A Social Construction Approach to Computer Science Education

Philip Machanick
School of ITEE
University of Queensland
St Lucia, Qld 4068
Australia
philip@itee.uq.edu.au
phone +61-7-3365-2766; fax 3365-4999

Abstract

Computer science education research has mostly focused on cognitive approaches to learning. Cognitive approaches to understanding learning do not account for all the phenomena observed in teaching and learning. A number of apparently successful educational approaches like peer assessment, apprentice-based learning and action learning have aspects which are not satisfactorily explained by purely cognitive models. On the other hand, these approaches are stratagems rather than comprehensive theories, in that they do not apply in all cases. Education theories which explore learning beyond the cognitive domain such as social construction may provide additional insights into matters like teaching style, curriculum design and assessment practices. This paper proposes a start towards introducing social construction into computer science education, and proposes new directions for research, curriculum development and educational practice.

1 Introduction

A student doing a project meets his advisor regularly, asking probing questions as to what to do next. The advisor gives the student a low grade for turning in a thesis showing no insight. The student is mortified: "I did everything my advisor told me." The advisor meanwhile thinks this is the worst student she's seen in a long time, one who could not do a thing without help.

A department of computer science and electrical engineering decides that an introductory course on professional practice done by engineers is similar enough to one for computer science students that the two classes could be merged. The two groups of students do not get on with each other, and the experiment must be abandoned after the first trial.

Students in a computer science degree do Business School electives because they look delightfully easy. They barely make a passing grade, and don't know what went wrong. What went wrong was they were focusing on the technical aspects of questions, and missing the business angle.

What do all these examples (based on real situations) have in common?

They illustrate that a purely cognitive model of learning misses important aspects of how knowledge is constructed. In the first case, the student doing the project has a complete misconception about the relationship between the student and advisor. In the second, there was a distinct culture at that university of what it is to "be" an Engineer, or what it is to "be" a Computer Scientist (note the capitals). Mix the two groups, and they can't work together. In the final example, the CS students have failed to understand what is important in the business courses. They have focused on the domain of discourse with which they are familiar – the technical – and missed what the courses were really about: business practice.

To appear in *Computer Science Education, 2007. A pointer to the final published version with corrections will appear later. See the journal web site* `http://www.tandf.co.uk/journals/titles/08993408.asp`.

One of the issues raised in discussions leading up to Curricula 2001 (ACM and IEEE Computer Society, 2001) was the notion of how to produce graduates who can “think like a computer scientist”. This discussion was not taken further, but it could have led to exploring an often-overlooked aspect of education. Education is building the individual to hold some position in society – one where performance is often judged on a number of dimensions other than pure cognitive skills. Communication skills, for example, are often seen as lacking in technical professionals (Becker et al., 1997). A range of workplace skills, such as interpersonal skills, the ability to relate technology to corporate strategy and an understanding of group dynamics (Sawyer et al., 1998), for example, are important in contextualizing technical skills.

To build on the opening anecdotes, asking how to produce graduates who can “think like a computer scientist” is the wrong question. The right question is asking how to produce graduates who *are* computer scientists.

This is not a subtle difference. The focus on the cognitive domain ignores a wide range of issues relating to how knowledge is formed and applied, including communication, social interaction and styles of work.

Much recent educational research has shifted to the *social construction* model, which places much more emphasis on the social construction of knowledge, and less on the individual as a cognitive entity. While social construction is not a new idea (Berger and Luckmann, 1966), it has relatively recently become a mainstream approach, and has not been explored much in computer science education. This paper aims to provide a starting point for introducing social construction into computer science education: it outlines the theory, relates it to existing non-cognitive approaches, and provides some examples to illustrate how the theory can be applied.

The remainder of this paper is structured as follows. Section 2 outlines models of education, including how computer science educational research fits the more general area of education research. Section 4 offers a proposal on how computer science educational research could be moved closer to the mainstream, including both ideas on how educational approaches could change, and how the efficacy of these ideas could be evaluated. In conclusion, the paper summarises the key ideas presented and proposes a way forward.

2 Models of Education

It is useful to place research in computer science education in context with broader educational research. While much research reported in the computer science education literature is not reported as being within a specific theoretical framework, recognized models are likely to influence work even where these models are not made explicit. The aim in this section is to summarize recognized methodologies in computer science education, identify ideas which do not fit the common theory, and relate these methodologies and ideas to trends in broader research in education.

Constructivism is possibly the post popular theoretical approach in computer science education. It derives from the theories of Piaget (Piaget, 1971), who saw learning as occurring in distinct stages, particularly as a child went through stages of general understanding (Piaget, 1953).

The remainder of this section examines constructivism and other approaches to computer science education, then goes on to compare them with recent thinking in broader educational research.

2.1 Computer Science Education

Since constructivism is the most commonly cited approach to computer science education, a brief overview of the approach is presented here, followed by a summary of CSE approaches which cannot be explained purely by a cognitive theory.

From this start, it should be possible to relate gaps in CSE practice to advances in mainstream education theory.

2.1.1 Constructivism

Constructivism, derived from the psychological theories of Piaget (Piaget, 1953, 1971), is a cognitive theory of learning. It assumes that a learner has a mental model which may or may not be in essentially the right form, and may or may not be complete. Relatively easy learning occurs when the mental model is in essentially the right form. Learning becomes harder when the mental model is not in a suitable form – learning in this case essentially amounts to a *paradigm shift* – a change of world view (Kuhn, 1996).

The simpler kind of learning is often referred to as *assimilation*: new details are fitted into an existing model. Learning which requires changes to the model is referred to as *accommodation*. Accommodation is generally seen as occurring as a result of the model being found to be flawed: something does not fit (von Glasersfeld, 1995b).

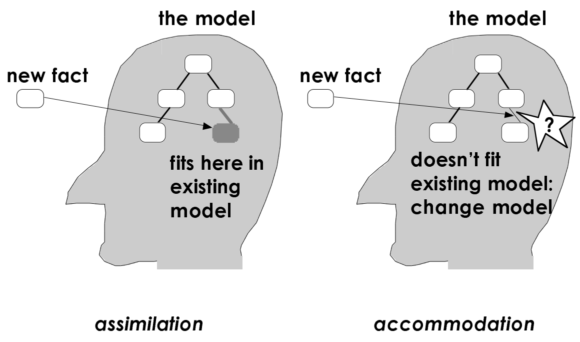

the model
new fact
fits here in existing model
assimilation
the model
new fact
?
doesn't fit existing model: change model
accommodation

Figure 1: Learning in Constructivism. *New facts which fit the existing model are slotted in (*assimilation*); those which do not fit require changes to the model (*accommodation*).*

The essential idea is illustrated in Figure 1.

In computer science education, constructivism has become an increasingly popular model of teaching and learning (Ben-Ari, 1998). One of the major influences in this popularity is the advocacy of teaching programming skills via Logo to young children using Piaget's theories (Papert, 1993). Constructivism is an intuitively appealing idea to computer scientists: we debug programs by starting from a conceptual model of what the program is doing, and test hypotheses based on this model. If the program is incorrect, the hypothesis is false. Sometimes, the model is wrong – sometimes the model is right, but there is a bug in implementation. Either way, we debug *something* when an inconsistency is found. It is appealing to computer scientists to think of learning as following a similar process. The learner's internal model has a bug in it, and the process of correcting this bug is not too dissimilar to finding out why a program doesn't work.

Relate this debugging idea to assimilation and accommodation. If a learner's existing model is essentially right, the "bug" in understanding is a small error like an off by one error in designing a loop and can be fixed by a small adjustment (assimilation). If the "bug" is at a deeper, more conceptual level – requiring accommodation – it is more like an incorrect choice of algorithm or data structure, one that does not fit the problem. Programmers generally debug at both levels so it is appealing for a computer scientist who thinks in such terms about teaching programming to extrapolate this sort of logic to teaching everything.

As noted before, much literature on computer science education is not specific on the educational theory, but the constructivism model is often assumed, or at least cognitive models underly assumptions of how learning takes place (Robins et al., 2003).

Purely cognitive models of learning are in wide use. For example, the still widely used Bloom's Taxonomy (Bloom, 1956) is a pure cognitive model: it does not aim to capture mechanisms for knowledge creation outside of the individual's final understanding.

Constructivism is increasingly cited in computer science education papers as providing a conceptual framework for the cognitive development of the learner (Schulte et al., 2003). Approaches to learning based on the notion of "debugging understanding" (Thomas et al., 2004) are common, and some have gone as far as to attempt to identify students' "mental models" as a starting point for understanding where they are going wrong (Götschi et al., 2003).

Constructivism is a powerful tool for understanding teaching and learning but, given the fact that it does not capture aspects of teaching and learning outside the cognitive domain, it is worth exploring other options.

2.1.2 Stratagems With Non-Cognitive Aspects

One idea which is reported to improve students' ability to understand what they are doing is to use test-driven development with rapid feedback on code quality as a way to facilitate reflection on what they are doing (Edwards, 2004).

The notion of students as "apprentices" – working on part of a project being run by an experienced practitioner, for example (Kölling and Barnes, 2004) is one with a growing following. The idea is that the "apprentice", rather than working on some artificially simplified problem, sees a real problem, but gets to work on a small part of it, possibly under supervision of an "expert". The idea of an apprentice has been extended to essentially cognitive tasks in the form of “cognitive apprenticeship”, which retains the essential idea of learning by working with an “expert” (Collins et al., 1989).

Related to these ideas is challenges to the traditional model of engineering education. Traditionally, engineering programs start from building a theoretical foundation and only later move to application of the theory to real problems. This traditional model has a lot of benefits, but is very artificial as compared, for example, to a practitioner attempting to solve a problem in an area where new theory applies.

The idea of learning by doing, with a “surprise” leading to deeper understanding (Schön, 1995) has some similarity to the “debugging understanding” idea of constructivists, but the difference is in seeing learning as being most effective when not abstracted from real problems. This idea has been taken up in a reform of the electrical and computer engineering curricula at Carnegie Mellon University (CMU): students are introduced to engineering alongside basic sciences, so they can see what the sciences are for (Director et al., 1995).

Yet another idea which has developed a following is peer assessment (Brookes and Indulska, 1996; Gehringer, 2001; Hübscher-Younger and Narayanan, 2003). This kind of approach is not necessarily popular with students (Rada et al., 1993), but even in cases where student attitudes are not positive, positive educational outcomes have been observed (Machanick, 2005).

What do all of these ideas have in common?

They all move away from the notion of accumulating knowledge in isolation from how it will be applied. Test-driven development is an actual methodology used in the real world. Even when it is started early, is not an artifice to encourage novice learning, but a good technique which applies even after the learner’s understanding has advanced. The “apprentice” idea again emphasises that novice participation is in solving “real” problems, but as a “peripheral” participant, with gradual evolution to a member of the community of “experts”. A “community” means any group a student may aspire to identify with: professional software engineers, academics, project managers, open source developers, etc. This kind of evolution from a beginner to an expert has been called “legitimate peripheral participation” (LPP): the idea is that the beginner’s role is not some artificial exercise, but a step towards integration with the expert community (Lave and Wenger, 1991). Introducing engineering early to make the science more “real” is again closer to expert practice. An expert engineer may occasionally have to review some detail of mathematics or physics while doing a complex design: this “learning” is in the context of solving an engineering problem. Finally, peer assessment gives a learner something closer to the “expert” view of the subject (the “expert” could be an instructor, or a job supervisor). By doing peer assessment, the learner sees the work from the expert’s perspective, not just that of the learner.

None of these ideas, however useful each may be, represents a general theory of education, but rather an interesting stratagem which points to the need to find a model which is not purely cognitive. LPP or apprenticeship models, while appealing, do not cover all learning situations – it is hard to conceptualise some learning situations in this mould (Ben-Ari, 2004).

2.1.3 Summary

Constructivism has developed a strong hold on thinking in computer science education, but it does not address non-cognitive aspects of learning. Ideas like gradually drawing a beginner into a community of experts and learning theory in conjunction with practice cannot be described purely in cognitive terms.

Given that there are good ideas which do not fit the popular model of constructivism, it is worthwhile exploring the space of educational theories which go beyond the cognitive domain. That is not to say that constructivism should be thrown out, but its limitations should be recognised.

2.2 Social Construction

Consider again the opening anecdotes – for example, that of trying to combine a computer science and an engineering class. The two groups of students had completely different perceptions of the course. The engineering students gave it a reasonable rating on the post-course survey, while the computer science students rated it as below average. There were persistent reports of the CS students feeling slighted at being grouped with engineers. Conversely, engineering lecturers didn’t want CS students in their class. None of these objections appeared to be grounded in any objective basis for criticising the course, the lecturers, or the students.

This brings us back to the notion of what it means to *be* a computer scientist (see Section 1).

What exactly does this mean?

One aspect is observing possibly unstated conventions. We have many conventions which are explicitly stated, such as code layout, naming conventions, etc. However, some conventions – such as drawing standard components of a logic circuit in a standard layout – are usually picked up by observing what others do, rather than explicitly taught. While in this case, conventions can be developed for *secondary notation* (Petre, 1995) of this type, there are other cases where writing down every detail is difficult, and learning proceeds best from experience – preferably in interaction with more expert practitioners.

Where it comes to the practice of a discipline – the practical steps to reach a solution – it is difficult to codify all steps, and to identify all strategies which every individual with a different learning style might employ. How many computer science educators have encountered the student, attempting to solve a programming problem staring at a blank sheet of paper, with the question, "Where do I *start*?"

Another issue is understanding how to interact with more senior or more experienced people in the field. When is it appropriate to ask for help? When are you expected to work things out for yourself? Expectations vary according to the field. A novice medical practitioner, for example, would be foolish not to seek advice on a life-threatening issue. A junior programmer who has forgotten a common algorithm would do better to look it up than to ask a manager.

Dealing with these questions leads us to an educational theory which is achieving growing acceptance.

2.2.1 Situating Social Construction

Cognitive science-based approaches to education have evolved from constructivism, which places the learner in isolation, to social constructivism, which recognises the social role of learning (Pear and Crone-Todd, 2002; Hundhausen, 2002). Social constructivism still ultimately views knowledge in cognitive terms, though it recognizes the role of interaction with others in the learning process.

The social construction model goes further in placing the social aspect of learning at the centre. Instead of focusing on the development of individuals' cognitive models, the focus is on how the individual interacts with a community in developing their understanding. Confusingly, Papert defines something quite different as "constructionism": learning by constructing an entity (Papert, 1991). Constructionism is a variant on social constructivism, emphasizing learning by doing.

Advocates of the social construction model argue that constructivism in any form has too limited a definition of knowledge – that it's constrained to defining knowledge as what's represented in someone's head, without acknowledging the role of communication in defining knowledge (Gergen, 1995; Rowlands and Carson, 2001).

Much of the education literature on social construction is inaccessible to the non-specialist, unaccustomed with the terminology of phenomenology. The basic idea, though is rather simple:

The effect of a learning event depends on how you experience it.

Phenomenology, among other things, argues that phenomena are defined by how they are encountered (Husserl, 1971). There is evidence that learning is highly related to the place and context of encountering a new fact or insight. Medieval illuminated manuscripts were designed with this concept in mind. Decorated initial letters of words and other decoration of the printed page were designed as aids to the memory, because a reader of a book might have use of it once in a lifetime. The graphical layout of the page was an aid to memorising the text (Clanchy, 1993). Encountering the text within a context of specific decoration on the page is an example of a phenomenon encountered in a particular context.

2.2.2 Social Construction in Practice

An example of the distinction social construction makes from other learning models is its view of the lecture. A lecturer may go through a lengthy process of knowledge discovery – reading books, searching for other sources, building on experience, etc. – then stand in front of a class and speak authoritatively, without any evidence of the preparation that went into the lecture. The class is not given an effective insight into how to learn the material for themselves, because they do not see the process the lecturer went through to acquire the knowledge (Gergen, 1995).

Figure 2 illustrates how a learner is gradually brought closer to the community of experts, by starting as an outsider, and gradually becoming more adept. The learner is depicted as looking towards the "community" of which they will ultimately form part. As new facts, methodologies, and so on, are encountered, the learner reflects on them, and gradually builds a capacity to become one of the "experts".

How can the learner be more involved in knowledge creation, in a way which emphasizes the learning process, understanding how to work from encountering knowledge to building ones own internal model, and gradually

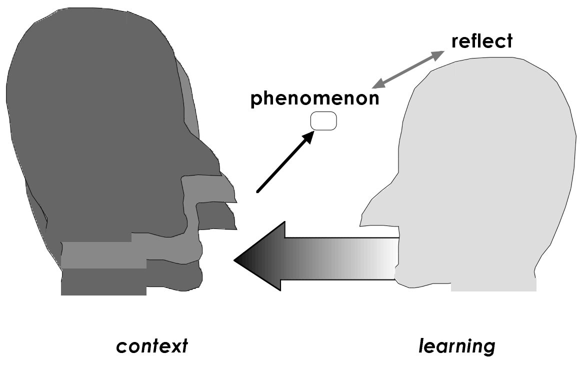


Figure 2: The Social Construction model. *The learner gradually becomes closer to the community of the adept, through encountering phenomena, and building them into their own model, through* reflection. *The cognitive aspect, though still a component of learning, has to be seen in the context in which it is developed.*

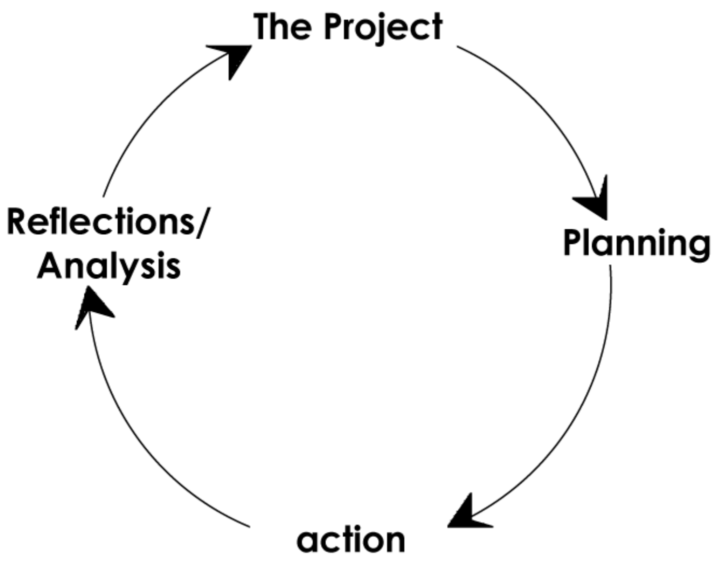


Figure 3: Action Learning Cycle (Bunning, 2001).

moving from peripheral to full participation in the community of experts? Legitimate peripheral participation (LPP) (Lave and Wenger, 1991) fits this model well, but what other approaches could apply where LPP does not fit well?

A specific approach to implementing the social construction model is called *action learning*. Action learning, like LPP, is a stratagem which can potentially be applied within other theories of education.

Action learning empasizes a cycle of starting from a desired outcome ("project"), planning an approach to solving the problem, applying the plan ("action") and reflecting on the outcome. If the problem is not yet solved, the cycle is repeated (Bunning, 2001). Action learning fits computer science education well, because many computer science problems can be formulated in the style of a problem to be solved, with the possibility of forming a plan which can be tried out, and evaluated ("reflection"). The step of reflection is easy to leave out – or pay lip service to – with programming problems, for example, because there is a temptation to make random changes to the program until it appears to behave correctly.

Figure 3 illustrates the general idea.

Action learning is a good fit to the social construction model. The feedback cycle emphasizes keeping the learner's experience in the loop. In attempting to learn something, the student tries to formulate a model, does something, then thinks about what has happened. The emphasis is on the student's experience in encountering the new concept. This experience can include working with a group, or an "expert" (an academic or tutor).

Action learning in its simplest form appears to be quite similar to the notions of learning in constructivism, in which the learner has to develop a mental model by a combination of assimilation and accommodation – respectively, fitting new information into a mental model, or adjusting an existing mental model for information which doesn't fit. In constructivism, a learner is seen as forming hypotheses to test existing mental models, a process which gradually leads to refinement of the mental model (von Glasersfeld, 1995a).

However, the social construction model places more emphasis on the learner's environment. Phenomenology looks at the relationship between the learner and phenomena they encounter (Ramsden, 1988). Thus, the action learning cycle should not look only relate to cognition, but also aspects of learning such as developing the skill of working in a group, or understanding what a user wants.

The value in being specific about using social construction is that it provides a framework for generalising from stratagems like LPP, and finding alternative stratagems where these do not apply. Further, understanding these approaches in terms of the social construction model helps us to pay attention to what the student is doing, how they relate to others, how they work and how they communicate, rather than focusing solely on cognitive measures of their work.

3 Application to Computer Science

To take these ideas further, let us consider how the action learning model can be applied in computer science education more generally, as a tool for applying the social construction model.

A key issue is to ensure that students actually *reflect* on what they are doing. The importance of reflection derives from the phenomenological underpinnings of the social construction model: we understand subjective experiences in terms of the context in which we encounter them. These subjective experiences, or "phenomena", become internalised by reflecting on them (Husserl, 1971).

It is a common experience of computer science instructors that students sit at the computer programming by trial and error (Edwards, 2004): if the compiler reports a problem, hack at the code until the error isn't reported. If the program crashes, flail about changing the code until it stops crashing. Such attempts at "fixing the problem" clearly do not involve much reflection. As related to Figure 3, the students are starting out with some idea of the project, going straight to action, and trying different actions until they achieve an apparently correct outcome. That little learning has taken place may only later be apparent, when the students are required to take a test, or solve a different problem requiring the same skills.

To make the discussion more concrete, let us consider a few of examples. The first is designing software using a class hierarchy, the next is teaching data structures and algorithms, and the last is teaching computer architecture. These examples are chosen to illustrate areas which place different demands on teaching and learning. The first is based on experience of teaching several classes, and the latter two examples illustrates some of the difficulties in attempting to apply the principles of social construction in practice.

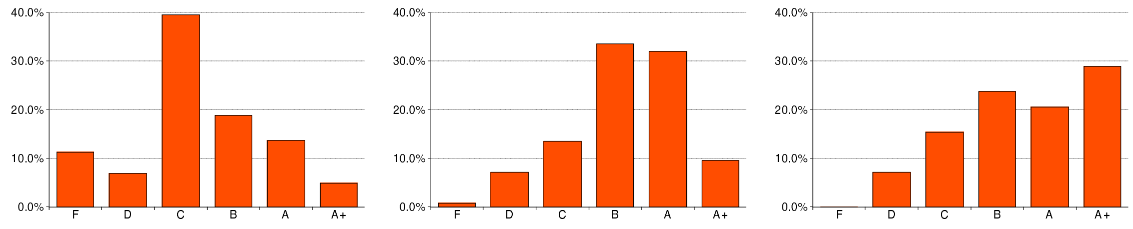
3.1 Programming with Classes

Many introductory texts are slow to introduce inheritance (Decker and Hirshfield, 1995; Kamin et al., 2002; Horton, 2002), most likely because it's seen to be a difficult concept. Approaches in which the empasis is on diving quickly into standard libraries (Collions, 2005) – clearly the way object-oriented programming was intended to be learnt (Goldberg and Robson, 1983) – are less common.

The problem with teaching object-oriented programming without starting from inheritance and libraries is that students develop a programming style at odds with the philosophy of object-oriented programming, and then are expected to unlearn this non-reuse style of programming. The issue here is very much one of building a student up to what an object-oriented developer does differently – not purely a matter of developing particular knowledge (in which case, it's less clear that the order of learning matters).

In terms of phenomenology, students have encountered object-oriented programming in a particular form, and that form becomes the one they learn.

What are the alternatives? Let's consider two options, which have stimulated debate, but not led to a clear outcome (a definite case that one is better than the other).



(a) 2002 (b) 2003 (c) 2004

Figure 4: Results compared across three years. *The 2003 class had the quizzes.*

One approach is to introduce object-oriented programming from day 1, as an approach in which standard libraries are built upon (Machanick, 1998). However, this approach is difficult in a first course, as there is a conflict between introducing basic syntax to build non-trivial examples, and exploring the library. This approach worked in Smalltalk-80 because the basic syntax was extremely simple, and programming constructs such as iteration and selection were built up out of library components (Goldberg and Robson, 1983). With languages like C++ and Java, this approach is more difficult, and strategies like subsetting the language or using toy dialects defeat the object of exposing students as early as possible to something "real".

Another approach is to start with a language with very simple syntax and semantics but which allows sophisticated concepts to be explored quickly. This is the philosophy adopted in various Scheme-based texts (Abelson and Sussman, 1996; Harvey and Wright, 1999). One can of course argue against starting with a language like Scheme because (practically) no one uses it for real projects. However, with the background of programming in any other language, an object-oriented course can start immediately with exploring the class library, and leave issues like iteration to picking up differences from previously-known languages.

The point here is not to enter into a language war, or to advocate or oppose a specific order of learning concepts. The point is that learning something in an artificial way which does not expose the learner to "expert" thinking and approaches can lead to an incorrect start, and a pattern of work which is hard to break.

A social-construction approach to evaluating these alternatives could be a useful addition to previous debate about which approach is "better" – and to opening up debate on other alternatives.

3.2 Data Structures and Algorithms

In 2003, I attempted to apply social construction principles in a single aspect of a Data Structures and Algorithms course. After each weekly tutorial, students were required to take a short quiz, and mark each others' attempts. The notion was to force reflection, and put students in the position of an "expert".

This approach was not very popular with students (Machanick, 2005). One comment on a survey illustrates the reaction of the class:

> *Waste the time in tutorials => Tutes rushed . . . basically means you should learn at home before coming to tutorials => Defeats the purpose. Marking is also pointless*

Being forced to "learn at home"? Clearly this is not what students do . . . but it is what a lecturer often has to do when preparing a course in a new area.

Grade distributions from comparable classes over three years are presented in Figure 4. At the University of Queensland, a numeric grading scheme with A+ represented as 7 and a low fail as a 1 is represented here as grades of F through A+ (with F representing a grade of 1 or 2). What is most obvious about the distributions is that the class in which quizzes were run had a distribution closer to normal. A possible interpretation of the variation in shape of the distributions is that the 2003 class forced all students to work closer to their ability, rather than some failing to do well because they did not keep up. However, there is a limit to how much to read into these results because different lecturers were involved each year, even if the material was similar.

Although students did not much like the quiz idea, it likely had some benefits, most notably making them better prepared for tutorials (Machanick, 2005).

However, it would be worth reconsidering the strategy to make it less unpopular. Although a non-cognitive aspect is included in the approach – placing the students in the role of "experts" marking each other's work, the context was not sufficiently natural to appeal to students, illustrating the difficulty of finding suitable stratagems to articulate the social construction approach.

3.3 Being a Computer Architect

The last experience is of running a computer architecture course with the intent of exposing the class to what it meant to *be* a computer architect. The idea was that a student with a reasonable general computer science or engineering background with some exposure to logic design could be drawn into a design project, and have to pick up missed background through working on the project. The course was structured around evaluating alternatives to a new processor design. Lectures were a smaller fraction of the course than usual (1 hour a week, out of a total of up to 5 contact hours a week), with frequent reflection breaks, where students were expected to think about what they had just been told, and to attempt to apply their new knowledge. The project required that the students choose from a list of vaguely specified topics, and narrow their choice down to something which would advance the big-picture goals of the overall project.

As with any real industry or research project, students had a number of checkpoints and design reviews, in which they were subjected to interviews, or had to give presentations. They also had to write a paper, and to review each others' work. Each of these points in the project was intended to provide an opportunity for reflection, in keeping with the action learning approach to social construction.

The biggest problem with this course was that students did not feel ready at the start to choose which project to work on, and had difficulty deciding where to start. As a consequence (the course was an elective so they had the choice to drop it), about two thirds of the class dropped the course. Those who stuck with it generally did well. Only one student who did not persist with all phases of the project failed overall.

A general observation is that the project forced student participation to a degree not possible in a course based more strongly on lectures. The difficulty students had with getting started on the project reflected too big a jump from being outside the circle of the adept to taking responsibility for using their knowledge.

In the next run of the course, students will be given more assistance in making an initial choice of project, and part of the grading scheme will reflect their ability to come to grips with defining what they have to do in the project.

3.4 Summary

The social construction model offers some challenges to conventional wisdom about education. It requires that we re-evaluate the role of lectures. It requires that we think in general about how students *encounter* new knowledge. It does not necessarily invalidate any previous approaches, but it does provide a framework within which to reflect on what we do, and to try to think through how to improve our approach. It provides a basis for rethinking why some approaches work, while others do not. Most significantly, it provides a framework for understanding how non-cognitive aspects of education could be tackled.

The discussion presented here on how we could rethink our approach to teaching object-oriented programming indicates some of the potential for using social construction to stimulate debate. The alternative approaches outlined – finding ways of learning class libraries first in an introductory course, versus starting with another programming model before going object-oriented – are not new, but it's useful to have a new conceptual framework in which to evaluate these ideas.

As the data structures and algorithms example illustrated, finding a suitable stratagem depends on the nature of the material and the approach which seems natural to students. Peer review can work well in some situations, but if it is seen as an artifice, it is not only likely to be rejected by the students, but also fails to meet one of the objectives of the social construction approach: introducing knowledge in a context where students can participate meaningfully in developing their understanding.

The computer architecture course illustrates more radical transformation of a course to fit the social construction approach. The success of the new approach will be the subject of future evaluation, once teething problems have been eliminated.

4 A Proposal

Reforming curriculum in terms of a social construction approach requires clarity on how teaching and learning needs to change. For this reason, more emphasis is placed in this paper on changing education strategy rather than on curriculum reform. However, it is still useful to give some consideration to how curriculum design could change.

Changes to how course material is delivered are reasonably easy to propose in the abstract, but experience will be required to develop a range of practical approaches, to extend the stratagems already identified in this paper.

The biggest challenge is in defining new research directions, as the style of measurement has to change if the focus is moved from cognitive approaches.

The remainder of this section takes up these issues.

4.1 Curriculum Reform

A social-construction model suggests that the basis for knowledge delivery be challenged at a more fundamental level than the traditional approach of tinkering with content. Social construction (and, to a lesser extent, constructivism), challenges the transmission model of learning, that learners are passive recipients of knowledge. Further, the notion of a learner gradually moving from peripheral to full participation implies a stronger emphasis on what kind of person a learner aims to be – not merely an emphasis on the knowledge they need to pick up.

The follow-up to Curricula 2001, Curricula 2005, attempts to define a broad range of disciplines including hardware engineering, computer science, information systems and information technology as regions of a 2-dimensional space, with a vertical axis labelled from hardware up through various software layers to organisational issues and information systems, and a horizontal axis varying from theory to application deployment and configuration (ACM et al., 2005). The problem with characterising knowledge in this way is that it does not take into account very different styles of work in different fields. An engineer installing a computer system will tend to take a very different approach to a technician if something fails, for example. Providing both groups with the same educational experience for overlaps in content may fail, as in the opening anecdote, where CS and engineering students didn't mix.

The most important issue for curriculum reform raised by the Social Construction theory is that we should look beyond content as defining curriculum, but consider also the way in which students encounter new knowledge as part of the curriculum.

4.2 Educational Strategy

It should be made clear at this point that doing "real" things should be distinguished from a "real-world" focus. In an academic institution, developing intellectual rigour, preparing students for a research career and establishing principles which may rarely apply but which could become the base for future discoveries are important components of the learning experience.

Some of these aspects of higher education may appear to be remote from what is done in the workplace, but are important parts of preparation for professional practice.

However, making apparently useless aspects of education "real" by showing how expert practitioners work can help to make students value skills like abstraction, reasoning from first principles and combining knowledge from multiple sources. These relatively "academic" skills may not apply in routine work, but add value to a qualification from a good tertiary institution, as opposed to a trade skill (such as learning how to use or maintain a particular piece of software).

The big challenge in an environment of limited resources is to change the way the teacher and learner interact. Ideally a lecture should be an exchange of ideas, with the lecturer playing the role of the expert, and the class the novices or apprentices, attempting to join the world of knowledge construction. In practice, with large classes and highly diverse student populations, this sort of dialogue is very difficult to achieve. However, a new insight into the nature of the task can help alter emphasis and detail of how we approach the problem. For example, in a lecture, explaining how the knowledge which went into the lecture was acquired is a step towards breaking down the notion of the lecturer as an unassailable authority (Gergen, 1995). Using peer assessment in a way which emphasises learning to become an "expert" is another example. Using students directly to grade each other's work can result in negative response, e.g., because the students don't accept another student's judgment as "expert" enough. One approach which can change that perception is if the grading is itself assessed. For example, in an advanced computer architecture course, an approach I have used with some success is to have students write a paper in imitation of the style of submitting a paper to a journal:

- *initial draft* – submit an initial draft which is not assessed but which is passed to other members of the class for review (the lecturer also writes a review)
- *refereeing* – review papers of other members of the class (the quality of refereeing is assessed by the lecturer)

- *final draft* – based on reviews, write a final version for assessment

The value in this approach is that the students get to see the work of others in the role of an “expert” – a referee judging someone else’s work. Their initial draft is assessed by other members of the class but solely for feedback (or formative assessment (Brown et al., 1997)). The review process allows for reflection at two levels: looking at someone else’s work as a way of understanding strengths and weaknesses in general of related kinds of work, and thinking about feedback from reviewers of their own paper. An exercise of this kind relates well to the action learning cycle of Figure 3.

More broadly, whenever designing or evaluating an educational approach, it is worth looking for the following issues:

- *how knowledge is encountered* – is it artificial, or a bland presentation of facts – or is it made real by relating to a problem or motivating example?
- *reflection* – are learners encouraged by the nature of the exercise to reflect on what they are doing, or can they get away with trial and error, copying out a likely solution, or some other unthinking solution strategy?
- *engagement with knowledge community* – how can students be made to become active participants in learning?

Sometimes it is hard to make these principles apply in practice. Stratagems like action learning, peer assessment and apprentice-like approaches give pointers to what is possible, but other approaches are needed where these do not apply.

4.3 Research Directions

Some existing approaches to computer science education already fit the social construction model, even if they are not explicitly designed as such. Accordingly, approaches like peer assessment, problem-based learning and apprentice-based schemes would be interesting to re-assess as social construction approaches. Questions worth asking include:

- Has reflection taken place? If so, how effective was it? If not, how can it be made more effective?
- Have students participated in knowledge construction? How can this be measured or made more effective?
- Is the way students encounter knowledge a reasonable reflection of future learning experiences? If not, how can this experience be made more authentic?
- Has the educational experience changed the nature of the student’s work? Has it exposed them to what it means to *be* whatever kind of professional their program aims to develop? How do their new skills apply to their future work?

In the case where a situation does not clearly fit a popular stratagem, it is necessary to consider how the theory could apply. In some cases, this may be more difficult than others. Even in case of a class of 500 students where the sole resource available is lecture sessions, some imagination can involve the class more, though it would be more profitable to focus initial efforts on smaller classes, lab sessions, tutorials, or other contexts where smaller numbers are involved.

One of the biggest challenges is how to make students more active learners. Starting early with problem-focused introductory courses, in which real problems can be solved from the start, is one option. An apprentice model (working on a small aspect of a big problem) or using a programming language which allows rapid progression to real problems is another example. Measuring the effectiveness of these strategies provides some interesting challenges. Again, examining the effect not only on modified courses but across other courses would be interesting.

Another important consideration is that changing teaching and learning styles requires changes to assessment models. Assessment drives student behavior (Gibbs, 1999), and an assessment approach which still focuses on measuring cognitive outcomes will undermine an attempt at including non-cognitive aspects in teaching and learning. For example, if ability to work in a team is important, an indirect measure like a poor grade for a badly executed team project may be less useful than a more direct assessment of ability to work in a team. Finding new measures of student performance would be a useful direction for new research in CSE.

There are two levels at which it could be interesting to measure changes resulting from adopting a social-construction approach. The first is direct measurement of educational outcomes in any altered course, and the second is broader measurement of whether student approaches to learning have altered. For example, if a course encourages reflection, it would be interesting to see if reflection becomes a habit. One strategy for encouraging reflection is to have students keep a reflective diary (Fekete et al., 2000). If this is done in one course, do students continue the practice in another? Would enforcing use of reflective diaries for a few courses lead ultimately to other improvements in student behaviour like thinking through programming problems, rather than trial-and-error coding?

A big challenge in general when introducing a change is student acceptance if no one else is making the change. If for example all courses involved some aspect of peer assessment, the issue of students finding it strange would be avoided. Any measurement of the effect of changes therefore has to take into account the possibility that students are prejudiced against change.

5 Conclusion

This paper has examined the need for new models in computer science education, starting from the fact that there are non-cognitive aspects to education, which are not addressed by the most popular theory, constructivism. Further, a variety of stratagems exist, which have been shown to have some success, despite going outside the purely cognitive domain.

The social construction model has been proposed as having some useful properties for learning computer science concepts, as well as being a good fit to existing non-cognitive approaches.

The remainder of this section summarizes key ideas from the paper, proposes a way ahead and wraps up with overall conclusions.

5.1 Key Ideas

The key idea of phenomenology as it applies to social construction of knowledge is that learning depends on how something new is encountered. An important aspect of the social construction model is a combination of shared experience and reflection. The model therefore does not see knowledge as strictly existing inside a person's head, but existing as a consequence of interaction with a community.

Action learning is one approach to implementing the social construction model, in which a cycle of planning, implementing and reflecting is emphasized. This approach is a good fit to much learning in computer science, especially programming-related courses.

Much of this aligns to ideas already found to work in computer science education (such as apprentice-like learning), so the challenge is to extend the benefits of these ideas to areas where they are not as obvious a fit.

5.2 Way Ahead

An obvious starting point is to re-evaluate existing strategies in terms of the social construction model, starting from ideas most similar in concept. A good next step would be to examine ideas which work and ideas which do not, to try to gain some insights into whether promoting the social construction model will increase the number of approaches which work.

If this initial investigation shows promise, the next stage is to pursue broader changes in curriculum and educational strategy, based on social-construction thinking. An important part of such change is rethinking student assessment, in line with an increased focus on non-cognitive aspects of education.

This paper has outlined some areas in which this process could be applied; the challenge to computer science educators is to take these ideas forward to new areas. It would be an interesting start to re-open debate about how best to teach programming: whether to start with object-oriented concepts or introduce them later, or other alternatives such as are outlined in Curricula 2001 (ACM and IEEE Computer Society, 2001).

5.3 Concluding Thoughts

The social construction model is gaining acceptance in the broader educational community. Despite having originated in the 1960s – with its roots in phenomenology going back to the early part of the twentieth century – it is not well known among computer science educators.

Given that there are problems in computer science education which have not been solved using previous approaches, it is worth exploring a new avenue. Social construction is a growing movement among educators, which does not necessarily mean it applies in all areas. However, it appears to be a good fit at least to some areas of computer science education, and therefore worth further exploration.

Acknowledgments

I would like to thank Gloria Dall'Alba and Jennifer Vadeboncoeur for introducing me to and clarifying the ideas behind current theories of education.

References

Abelson, H. and Sussman, G. J. (1996). *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA, 2nd edition.

ACM and IEEE Computer Society (2001). *Computing Curricula 2001 Computer Science*. ACM/IEEE-CS. `http://www.computer.org/education/cc2001/final/index.htm`.

ACM, IEEE Computer Society, and Association for Information Systems (2005). *Computing Curricula 2005: The Overview Report*. ACM/IEEE-CS/AIS, draft edition. `http://www.acm.org/education/Draft_5-23-051.pdf`.

Becker, J. D., Insley, R. G., and Endres, M. L. (1997). Communication skills of technical professionals: a report for schools of business administration. *SIGCPR Computer Personnel*, 18(2):3–19.

Ben-Ari, M. (1998). Constructivism in computer science education. In *Proc. twenty-ninth SIGCSE Tech. Symp. on Computer Science Education*, pages 257–261, Atlanta, Georgia, United States.

Ben-Ari, M. (2004). Situated learning in computer science education. *Computer Science Education*, 14(2):85–100.

Berger, P. L. and Luckmann, T. (1966). *The social construction of reality: a treatise in the sociology of knowledge*. Doubleday, New York.

Bloom, B. S., editor (1956). *Taxonomy of Educational Objectives: Book 1 Cognitive Domain*. Longman, London.

Brookes, W. and Indulska, J. (1996). Teaching internet literacy to a large and diverse audience. In *Proc. second Australasian Conf. on Computer Science Education*, pages 7–15, The Univ. of Melbourne, Australia.

Brown, G., Bull, J., and Pendlebury, M. (1997). *Assessing Student Learning in Higher Education*. Routledge, London.

Bunning, C. (2001). Turning experience into learning. In Steffe, L. and Gale, J., editors, *Action learning at work*. Gower, Aldershot, Hampshire.

Clanchy, M. (1993). *From memory to written record, England 1066-1307*. Blackwell, Oxford, 2nd edition.

Collins, A., Brown, J. S., and Newman, S. E. (1989). Cognitive apprenticeship: Teaching the crafts of reading, writing and mathematics. In Resnick, L. B., editor, *Knowing, Learning, and Instruction*, pages 452–494. Erlbaum, Hillsdale, NJ.

Collions, W. J. (2005). *Data Structures and the Java Collections Framework*. McGraw Hill, Boston, 2nd edition.

Decker, R. and Hirshfield, S. (1995). *The Object Concept*. PWS, Boston.

Director, S., Khosla, P., Rohrer, R., and Rutenbar, R. (1995). Reengineering the curriculum: design and analysis of a new undergraduate electrical and computer engineering degree at Carnegie Mellon University. *Proc. of the IEEE*, 83(9):1246 –1269.

Edwards, S. H. (2004). Using software testing to move students from trial-and-error to reflection-in-action. In *Proc. 35th SIGCSE Tech. Symp. on Computer Science Education*, pages 26–30, Norfolk, Virginia, USA.

Fekete, A., Kay, J., Kingston, J., and Wimalaratne, K. (2000). Supporting reflection in introductory computer science. In *Proc. thirty-first SIGCSE Tech. Symp. on Computer Science Education*, pages 144–148, Austin, Texas, United States.

Gehringer, E. F. (2001). Electronic peer review and peer grading in computer-science courses. In *Proc. thirty second SIGCSE Tech. Symp. on Computer Science Education*, pages 139–143, Charlotte, North Carolina, United States.

Gergen, K. J. (1995). Social construction and theeducational process. In Steffe, L. and Gale, J., editors, *Constructivism in Education*, pages 17–39. Lawrence Erlbaum Associates, Hillsdale, NJ.

Gibbs, G. (1999). Using assessment strategically to change the way students learn. In Brown, S. and Glasner, A., editors, *Assessment Matters in Higher Education*, pages 41–53. Society for Research into Higher Education and Open University Press, Buckingham, UK.

Goldberg, A. and Robson, D. (1983). *Smalltalk-80: The Language and its Implementation*. Addison-Wesley, Reading, MA.

Götschi, T., Sanders, I., and Galpin, V. (2003). Mental models of recursion. In *Proc. 34th SIGCSE Tech. Symp. on Computer Science Education*, pages 346–350, Reno, Navada, USA.

Harvey, B. and Wright, M. (1999). *Simply Scheme: Introducing Computer Science*. MIT Press, Cambridge, MA, 2nd edition.

Horton, I. (2002). *Beginning Java 2*. Wrox, Birmingham.

Hübscher-Younger, T. and Narayanan, N. (2003). Constructive and collaborative learning of algorithms. In *Proc. SIGCSE Tech. Symp. on Computer Science Education*, pages 6–10.

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

Husserl, E. (1971). “Phenomenology,” Edmund Husserl’s article for the *Encyclopaedia Brittanica* (1927): revised translation by Richard E Palmer. *J. British Society for Phenomenology*, 2:77–90.

Kamin, S. N., Mickunas, M. D., and Reingold, E. M. (2002). *An Introduction to Computer Science Using Java*. McGraw Hill, Boston.

Kölling, M. and Barnes, D. J. (2004). Enhancing apprentice-based learning of Java. In *Proc. 35th SIGCSE Tech. Symp. on Computer Science Education*, pages 286–290, Norfolk, Virginia, USA.

Kuhn, T. S., editor (1996). *The Structure of Scientific Revolutions*. University of Chicago Press, Chicago, 3rd edition.

Lave, J. and Wenger, E. (1991). Legitimate peripheral participation in communities of practice. In *Situated Learning: Legitimate Peripheral Participation*, pages 89–117. Cambridge University Press, Cambridge.

Machanick, P. (1998). The abstraction-first approach to data abstraction and algorithms. *Computers & Education*, 31(2):135–150.

Machanick, P. (2005). Peer assessment for action learning of data structures and algorithms. In *Proc. 7th Australasian Computer Education Conf. (ACE2005)*, pages 73–82, Newcastle, Australia.

Papert, S. (1991). Situating constructionism. In *Constructionism*, pages 1–12. Ablex, Norwood, NJ.

Papert, S. (1993). *Mindstorms*. Basic Books, New York, 2nd edition.

Pear, J. J. and Crone-Todd, D. E. (2002). A social constructivist approach to computer-mediated instruction. *Computers & Education*, 38(1-3):221–231.

Petre, M. (1995). Why looking isn’t always seeing: readership skills and graphical programming. *Commun. ACM*, 38(6):33–44.

Piaget, J. (1953). *The origin of intelligence in the child*. Routledge, London.

Piaget, J. (1971). *Psychology and epistemology*. Grossman, New York.

Rada, R., Ramsey, P., and Michailidis, A. (1993). Educational perspectives in collaborative hypermedia. In *Proc. 1993 ACM Conf. on Computer science*, pages 304–309, Indianapolis.

Ramsden, P. (1988). Studying learning: Improving teaching. In Ramsden, P., editor, *Improving Learning: New Perspectives*, pages 13–31. Kogan Page, London.

Robins, A., Rountree, J., and Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172.

Rowlands, S. and Carson, R. (2001). The contradictions in the constructivist discourse. *Philosophy of Mathematics Education J.*, 14. `http://www.ex.ac.uk/~PErnest/pome14/rowlands.pdf`.

Sawyer, S., Eschenfelder, K. R., Diekema, A., and McClure, C. R. (1998). Coporate IT skill needs: a case study of BigCo. *SIGCPR Computer Personnel*, 19(2):27–41.

Schön, D. (1995). The new scholarship requires a new epistemology. *Change*, 27(6):27–34.

Schulte, C., Magenheim, J., Niere, J., and Schäfer, W. (2003). Thinking in objects and their collaboration: Introducing object-oriented technology. *Computer Science Education*, 13(4):269–288.

Thomas, L., Ratcliffe, M., and Thomasson, B. (2004). Scaffolding with object diagrams in first year programming classes: some unexpected results. In *Proc. 35th SIGCSE Tech. Symp. on Computer Science Education*, pages 250–254, Norfolk, Virginia, USA.

von Glasersfeld, E. (1995a). Constructivist approach to teaching. In Steffe, L. and Gale, J., editors, *Constructivism in Education*, pages 3–15. Lawrence Erlbaum Associates, Hillsdale, NJ.

von Glasersfeld, E. (1995b). Sensory experience, abstraction, and teaching. In Steffe, L. and Gale, J., editors, *Constructivism in Education*, pages 369–383. Lawrence Erlbaum Associates, Hillsdale, NJ.