Introductory Programming and the Didactic Triangle

Anders Berglund
UpCERG, Uppsala Computing Education Research Group
Department of Information Technology
Uppsala University
Uppsala, Sweden
Anders.Berglund@it.uu.se

Raymond Lister
Faculty of Information Technology
University of Technology, Sydney
Sydney,
NSW, Australia
raymond@it.uts.edu.au

Abstract

In this paper, we use Kansanen's didactic triangle to structure and analyse research on the teaching and learning of programming. Students, teachers and course content are the three entities that form the corners of the didactic triangle. The edges of the triangle represent the relationships between these three entities. We argue that many computing educators and computing education researchers operate from within narrow views of the didactic triangle. For example, computing educators often teach programming based on how they relate to the computer, and not how the students relate to the computer. We conclude that, while research that focuses on the corners of the didactic triangle is sometimes appropriate, there needs to be more research that focuses on the edges of the triangle, and more research that studies the entire didactic triangle.

Keywords: didactic triangle, phenomenography, object-oriented programming.

1 The teaching of introductory programming is still a problem

Programming is hard to learn, and hard to teach. These problems are frequently acknowledged within the computing education community and are confirmed in several studies (e.g. Bennedsen, 2008; Berglund & Lister, 2007; Pears et al., 2007; Robins, Rountree, & Rountree, 2003). This paper begins by discussing previous research efforts that have tackled this problem. We then argue that, while previous research has advanced our understanding of how students learn to program, it is time to broaden our collective research focus. We will argue for a wider, more systematic focus on the complete teaching picture, instead of focusing upon parts of the picture. We will base our argument on new research findings, as well as other research sources, and will, as a conclusion, sketch the direction in which we propose that research now move.

We start (in section 2) by presenting Kansanen's (1999) didactic triangle, which is a model aimed at analysing and describing the entire teaching and learning situation. We then (in section 3) discuss separately each of the three corners of the triangle – teachers, students and programming. In section 4, we discuss the edges of the triangle – the relationships between teachers and programming, between students and programming, and between teachers and students. In sections 3 and 4, our analysis will focus on ambiguities and problems in the components and/or their relationships. In section 5, we then return to the complete didactic triangle, and apply the research results described in sections 3 and 4. In section 6, we conclude with our proposals for future work.

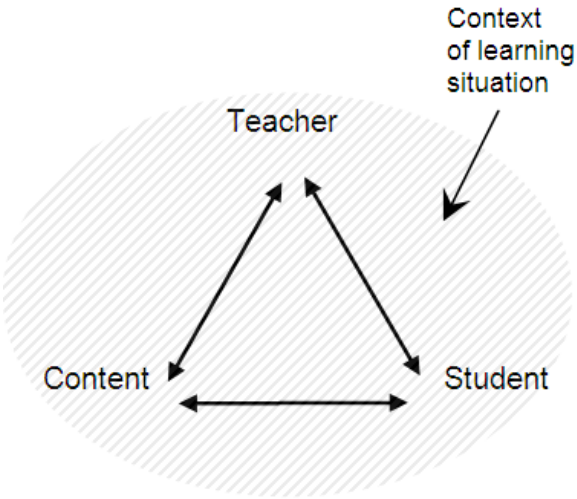

Figure 1: The Didactic Triangle (Kansanen, 1999; Kinnunen, forthcoming)

2 The Didactic Triangle

A teaching situation can be analysed and described in terms of its three main components: the student, the teacher and the content. These entities and their interaction can be illustrated in a didactic triangle, as shown in Figure 1 (Kansanen, 1999). When applying the model to the teaching of introductory programming, Kinnunen (forthcoming) argues that the context of the teaching situation must be taken into account, as teaching

does not occur in a vacuum; this is represented by the shading in Figure 1.

Although the didactic triangle should be seen and analysed as a whole – that is its raison d'être – we claim, based on Kansanen & Meri (1999), that it is often fruitful to precede (but not replace) analysis of the whole triangle with analysis of its components. The following two sections discuss the components of the didactic triangle, as best we can, given that each component by necessity has relations to its neighbours and the context.

The triangle is an analytic tool for, in the case of this paper, improving our awareness of issues often taken for granted and left implicit in discussions about programming.

3 The Corners of the Didactic Triangle

In this section, we discuss content, students and teachers in isolation from one another. Such a tight focus can sometimes be useful and appropriate in research, but it can also be a naïve perspective. Many of the topics discussed here in section 3 will need to be revisited in section 4, when we consider larger parts of the didactic triangle.

3.1 The "Content" Corner of the Triangle

Discussion about the learning and teaching of programming is often conducted without explicit reference to students and teachers. Instead, the focus is on the technology. Such a discussion rests upon the plausible, but often implicit, assumption that the simpler a technology is, the easier it is to learn. (In this paper, we ignore the difficulty of measuring simplicity.) Consequently, a discussion on how to simplify the technology can proceed without explicit reference to students and teachers. For example, Kolling, Koch and Rosenberg (1995) enumerated 10 requirements for a first year teaching language, where all 10 requirements related to principles of simplicity and transparency in programming languages. Students are only mentioned in one of the requirements, and only then in passing. That is not to say that the 10 requirements are wrong, or inappropriate, but merely that the 10 requirements offer a limited perspective on a complex issue.

The remainder of this section discusses briefly some other examples of where the focus is on the "content" corner, and where the student and teacher remain in the implicit background.

3.1.1 Languages and Language Wars

ACM Java Task Force (2006) was convened in 2004 with the following charter:

> To review the Java language, APIs, and tools from the perspective of introductory computing education and to develop a stable collection of pedagogical resources that will make it easier to teach Java to first-year computing students without having those students overwhelmed by its complexity.

The Task Force identified four significant challenges that instructors face teaching Java:

- Static methods, including "main"
- Lack of a simple input mechanism
- Conceptual difficulty of the graphics model
- GUI components inappropriate for beginners

Some other researchers (e.g. Grandell et al., 2006) have gone further, to advocate not using Java, and instead using other object-oriented languages, such as Python. There argument is that these other languages are simpler, and therefore (in their view) simpler to learn.

3.1.2 Tools

It is only natural that, when computing scientists are faced with a problem in their teaching, they look for a software solution. For example a very popular notion among computing educators is that students struggle with programming because they have difficulty visualizing how the algorithms work. That notion has spawned a plethora of visualization tools. As Stasko and Hundhausen (2004) explained, prior to a shift in the mid-1990's, the focus in that research was on the technology, not the students:

> The notion of using illustrations and pictures to explain computer algorithms and programs is nearly as old as computer programs themselves. … Initial research in the field of algorithm visualization was dominated by efforts to build algorithm visualization software systems and to expand the capabilities and expressiveness of these systems. … system paradigms, specification paradigms, types of views, and the like...

In section 5.2, we will examine what Stasko and Hundhausen then went on to say about the subsequent work on visualization. As a preview, we now paraphrase them, in terms of the nomenclature of this paper: they wrote that subsequent research broadened to examine larger parts of the didactic triangle.

3.2 The "Student" Corner of the Triangle

In this subsection, we discuss theories of student learning, without reference to the specific content of what is learnt by the student. Also, the theories discussed here assume that the teacher has no influence on the behaviour of the student. As noted at the commencement of section 3, such a perspective of the student can be naïve, and we shall revisit these topics later.

3.2.1 Deep and Surface Learning

In the 1970s, early phenomenographers identified two different approaches that students bring to learning. In the "deep" approach, students attempt to develop a genuine understanding of what they are studying, while students using the "surface" approach merely seek to complete the tasks set by the teacher (Marton & Booth, 1997).

While the notions of deep and surface approaches to learning are now well known, these notions are often understood and articulated in an incorrect, naïve fashion, where students are represented as being by nature "deep learners" or "surface learners". That is, the students are described without reference to either the content they are learning, or their teachers. As Biggs (2003) and many others have noted, both the teacher and the content have a profound influence on whether students adopt a deep or surface approach to learning.

3.2.2 Student motivation

In research on student motivation, the distinction between *intrinsic* and *extrinsic* motivation is commonly made. For an overview, see Entwistle (1998) or Ryan & Deci (2000).

While "intrinsic" and "extrinsic" are often used by teachers to describe the motivations of their students, the terms are often used naïvely, perhaps even incorrectly. In the naïve use of "intrinsic" and "extrinsic", it is the "student" corner of the didactic triangle that is in focus. That is, the motivation of student is described as an intrinsic property of the student, not as a reaction to the teacher or content being learnt.

The educational psychology literature on motivation reveals that student motivation is a complex issue that involves more than just one corner of the didactic triangle. Ryan & Deci divide extrinsic motivation into four subcategories, of which only one, "external regulation", describes what most teachers mean by "extrinsic motivation". Two of the extrinsic subcategories "identification" and "integration" are actually part of what most teachers naively refer to as "intrinsic" motivation. When many teachers use the terms "intrinsic" and "extrinsic", they are really referring to what Ryan & Deci refer to as the locus of causality, "internal" or "external".

3.3 The "Teacher" Corner of the Triangle

In this subsection, we discuss how teachers view their general role, independent of the specific content and specific students they teach.

During the last twenty years, studies in teaching and learning in higher education have to an increasing degree focused on the role of teachers (see for example Boyer, 1997; Kember, 1997; Ramsden, 2003; Trigwell, Prosser, Martin, & Ramsden, 2005), but such studies are sparsely represented compared to studies focusing on students. In computing education, there are very few studies of the teacher.

3.3.1 Content- versus Student-Centred

Fox (1983) identified four personal theories of teaching on the basis of his many anecdotal encounters with polytechnic teachers, from a variety of disciplines. The four personal theories formed pairs, with one pair being content- or teacher-focused and the other pair being student-focused.

Within the broad content- or teacher-focused category, Fox identified the sub-categories of 'transfer' and 'shaping'. In the first of these sub-categories, the student is viewed as a container into which the knowledge is to be poured. In the second sub-category, the student is viewed as a raw material to be moulded, or turned by some other 'manufacturing' process into a finished product.

Within the broad student-focused category, Fox identified the sub-categories of 'travelling' and 'growing'. In the 'travelling' sub-category, the teacher views the student as someone undertaking a journey, where discipline knowledge is the landscape, and the teacher is a guide. To define the 'growing' sub-category, Fox resorted to quoting Northedge (1976):

> In this case we conceive of the teacher as a gardener with the student's mind, as before, an area of ground.

There have been a number of studies – among them Dunkin (1990) and also Samuelowicz and Bain (1992) – using a variety of research methods, that have drawn broadly similar conclusions to Fox. In a meta-study, Kember (1997) found that numerous studies in this area showed a reliable distinction between teacher-centred/content-oriented and student-centred/ learning-oriented.

3.3.2 Research on Teachers within Computing

Lister et al. (2007) conducted a phenomenographic study of computing academics' understanding of teaching, and found categories consistent with Kember's meta-study.

Pears et al (2007) discusses the attitudes of teachers towards the students' success or failure in learning to program. They confirm that teachers often, when discussing the students, focus their arguments on themselves as teachers, not upon the students.

4 The Edges of the Didactic Triangle

In this section, we discuss the relationships between content and teachers, between content and students, and between teachers and students. With regard to content, we focus upon computer programming, particularly object-oriented programming.

4.1 Programming and Teachers

In section 3, we discussed teachers without reference to what they taught. At the university level, most academics do not separate the act of teaching from what they teach. As Bowden and Marton (1998, p. 143) expressed it:

> *… being good at teaching means that you are good at teaching something. You cannot teach in general and the way in which you deal with the particular content you are dealing with is what matters.*

Rowland (2000, p. 120) expressed a similar sentiment, and went further to warn of the dangers of making such a separation:

> *… a focus on generic approaches to teaching, and theories of learning, can lead to a separation of teaching method and subject matter. Academics or educational developers come to be seen as experts in how to teach but ignorant about what to teach … like experts of love who have no lover.*

Shulman has written extensively on this topic, introducing the concepts of *pedagogical content knowledge* (Shulman, 1987) and *pedagogy of substance* (Shulman, 1989).

4.1.1 Understandings of OO Programming

In April 2004, there was a vigorous e-mail discussion on the ACM SIGCSE members' mailing list (SIGCSE, 2004a, 2004b), concerning object-oriented programming (that discussion has since been regularly reprised in the

mailing list, on a smaller scale). Two quotes from that e-mail discussion illustrate the differences in how members of the SIGCSE community understand object-oriented programming:

> [Some email messages in this discussion imply] that selection and repetition are no longer necessary in object-oriented programming. I've been in this field for about 25 years, so maybe that makes me old, but I don't see how you can write a graphics render, a system simulation (eg: waiting-line simulation), an operating system, etc. without repetition and selection. (Jeffrey J. McConnell)

The second posting was written by Carl Alphonce:

> Selection and repetition are fundamental, but if statements and for loops are not. How selection and repetition are expressed in different paradigms differs. In OO polymorphism is the primary means of achieving selection. (Carl Alphonce)

Clearly, these two teachers have different understandings of OOP. In an earlier paper (Lister et al., 2006), we carried out a phenomenographic analysis (Berglund, 2006; Marton & Booth, 1997) of this April 2004 e-mail discussion. In this subsection of this paper, we extend our earlier work with a further analysis of the same data. This new analysis revealed three fundamentally different understandings of what OOP "is":

OO1. OOP is an extension of procedural programming
OO2. OOP is something fundamentally new
OO3. OOP transcends OO1 and OO2

Table 1 summarizes our findings. The next three subsections elaborate upon that summary.

4.1.1.1 OOP extends procedural programming

This category is illustrated by the earlier quote by Carl Alphonce (above) and by the following quote from Stan Warford:

> Java has the assignment statement. It has the if statement. It has loops. It has recursion. It has arrays. Your statement would be more convincing if it had none of these features because they were abstracted away at a lower level. But as long as it has them and students must use them it seems that traditional algorithmics (and by implication mathematics) must remain at the heart of CS. (Stan Warford)

4.1.1.2 OOP is something fundamentally new

In the second category, polymorphism and the interaction between objects are important. Also, the programming methodology of OO focuses on extending classes (i.e. inheritance) and refactoring rather than algorithm development and writing code from scratch. This understanding is illustrated in the following contribution to the discussion:

> I agree with your comments that if, while and repeat are not fundamental concepts, but rather selection and repetition are the fundamental concepts that may be represented by if and while. I can readily use and teach these concepts using polymorphism and recursion. (Richard Thomas)

4.1.1.3 OOP transcends OO1 and OO2

The third category presents an integrated perspective of OOP. This category transcends the idea that either procedural programming or object-oriented programming is more fundamental than the other. It goes further than a mere unification of categories OO1 and OO2, and contains relationships between the two previous understandings. The following contribution to the discussion illustrates part of that understanding:

> Philosophically, we must decide whether successively higher levels of abstraction provided by OO software development environments have caused algorithmic thinking and mathematics to become non-fundamental. (Stan Warford)

Important aspects of the categories	**Categories**		
	OO1. *OOP is an extension of procedural programming*	**OO2.** *OOP is something fundamentally new*	**OO3.** *OOP transcends OO1 and OO2*
Selection	IF-clause	Polymorphism	} The aspects that distinguishes OO1 and OO2 are not relevant. They are simply variations on a single theme
Program execution	Sequential execution of algorithms	Interaction between objects gives algorithm	
Role of objects	Containers of data and behaviour, created empty	The core concept. This is where "everything happens"	
Development	Writing code from scratch	Completing a framework	
Working methods	Problem solving and algorithm development.	Extending and refactoring	

Table 1: Teachers Understandings of Object-Oriented Programming

Warford then continues his email with a comment on an earlier statement by another discussant:

> I find it interesting that you would consider the Turing Machine, at the very lowest level of abstraction, to be fundamental and OO programming at the highest level to also be fundamental, while algorithmic reasoning with if, while, and arrays to be not so fundamental.

Warford acknowledges the thinking represented by the two previous categories, by discussing algorithmic thinking and OO programming. He then goes further in that he evaluates certain aspects of them. To be able to compare the understandings represented by the two categories, he has to see both of them from "the outside". Thus, we have identified a third category that transcends OO1 and OO2. This category takes a "bird's eye" perspective in that it sees the two previous understandings as variations on the same theme.

4.1.2 Mathematics or Software Engineering?

In the SIGCSE-mailing discussion, a further variation among teachers' understandings of OOP was revealed in their discussion of the relationship between OOP and mathematics, on one hand, and the relationship between OOP and software engineering (SE) on the other. In the discussion, Michael Kölling made the following claim:

> I think the only way this can eventually be resolved is that separate degrees are being taught in what are now regarded as sub-areas of computing (computer science versus software engineering being the obvious ones, but there will be more). (Michael Kölling)

Later, Conrad Cunningham addresses the question in the following way:

> This dispute gets to the heart of what software and computer science are all about. It is also one battle in a war that rages up and down modern intellectual history, the war between mathematical and physical worldviews.
>
> They have been fought in civil engineering: Do we design bridges based on mathematical models, or based on experience, aesthetics, and intuition? (Conrad Cunningham)

Two main positions about what underlies OOP were present in the data:

Underlying1. Mathematics underlies OOP
Underlying2. Software Engineering underlies OOP

The first category (Underlying1) offers a theory-driven perspective, stating that the fundamentals of CS are of a mathematical character. From this perspective, good teaching emphasises the theoretical, or mathematical.

The second category (Underlying2) gives voice to software engineering aspects. It is a people oriented perspective, summarised below by Michael Kölling:

> I also want students to learn to work in a programming team, read other people's code, assess quality of code in terms of maintainability and adaptability, and reason about quality trade-offs.
>
> […] There just are not many problems out there anymore that are solved by recluse individuals in a dark cellar room. (Michael Kölling)

4.1.3 Teacher Familiarity with the Content

As part of the SIGCSE-members email discussion, Stuart Reges made the following point:

> … if the material isn't straightforward for a lifelong computer scientist to teach, then can it really be all that fundamental? (Stuart Reges)

In a paper commenting upon the SIGCSE-mailing list discussion, Bruce (2005) acknowledged that not all computing academics have the background to teach OOP:

> [Some academics] … are simply thrown into an introductory Java course, even though their main experience is with procedural programming. Quite naturally, they will tend to teach most of the course the way they always have, including object-oriented topics where they occur in the text or syllabus, but without rethinking the overall approach of the course. … [The teachers of OOP] need to have developed programs larger than those assigned in introductory courses to have a real understanding why the organization supported by the object-oriented style is valuable. Once that understanding is there, the style can be taught more effectively to novices, even on relatively small programs. (Kim Bruce)

The technical background of teachers, and how it affects their teaching, is a relatively unstudied area of computing education research. We are aware of only two studies in this area. The first study was a biographical analysis done, by Carsten Schulte, as part of an ITiCSE 2006 working group (Lister et. al, 2006). Schulte analysed only two biographies, one from a OOP advocate and one from an OOP sceptic. One observed difference in the two biographies was that the OOP advocate had made a commitment to OOP before teaching it, whereas the OOP sceptic had found himself teaching OOP, not by his own decision, but as a result of an institutional decision. The second study (Liberman, Kolikant and Beeri, 2009) was of a high school teacher, who knew procedural programming, but who had been called upon to teach OOP while still learning it herself.

4.1.4 The Objectivist Perspective

By considering the relationship between programming and the teacher, a conversation ensues that is more rich than the conversation that ensues from considering each by itself. However, any conversation that is restricted to programming and teachers, and ignores the student, will inevitably become a conversation in the objectivist tradition. In that tradition, the curriculum and the pedagogy are constructed in such a way as to be most meaningful to the teacher, not the student. One example of objectivist pedagogy was articulated by Gries (2008), in his six principles for teaching objects first:

1) Reveal the programming process, in order to ease and promote the learning of programming.
2) Teach skills, and not just knowledge, in order to promote the learning of programming.

3) Present concepts at the appropriate level of abstraction.
4) Order material so as to minimize the introduction of terms or topics without explanation: as much as possible, define a term when you first introduce it.
5) Use unambiguous, clear, and precise terminology.
6) Introduce names for entities under consideration.

Gries' pedagogy is expressed in terms of content. The student is implicit in Gries' pedagogy. The constructivist perspective, of building upon what a student already knows, is not present in Gries' pedagogy.

4.1.5 OOP and Students

This section gives some examples of recent research on how students learn OOP.

4.1.5.1 Quantitative Studies

Butler and Morgan (2007) surveyed several hundred students in an object-oriented introductory programming unit. Students nominated the difficulty of several topic areas, on a 7 point scale (with 7 as the hardest). The topic areas were *Algorithms, Syntax, Variables, Decisions and Loops, Arrays, Methods, OO Concepts, OO Design,* and *Testing*. The average response was highest for *OO Concept* and *OO Design*.

Ma et al. (2007) investigated the mental models of assignment held by 90 students who had completed 70 hours of classroom learning in an introductory programming class. They found that approximately one third of the students held non-viable mental models of value assignment and over 80% of students held non-viable mental models of reference assignment.

4.1.5.2 Qualitative Studies

Eckerdal and Thuné (2005) performed a phenomenographic study to determine how students experienced the OOP concepts of object and class. They found that the students experienced an object as:

1. A piece of code.
2. As something that is active in the program.
3. As a model of some real world phenomenon.

They found that the students experienced a class as:

1. An entity in the program, contributing to the structure of the code.
2. As a description of properties and behaviour of the object.
3. As a description of properties and behaviour of the object, as a model of some real world phenomenon.

Eckerdal and Thuné described the educational implications of their research as follows:

> For the Java educator, one challenge is to construct an educational environment which facilitates for students to reach a rich understanding of the concepts object and class. To this end it is important to know the different ways in which students (as opposed to experts) typically experience these concepts. Our phenomenographic study has given such insight. Next, the educator needs to identify what variation the students have to discern in order to become aware of aspects belonging to a rich understanding of these concepts.

4.1.6 Student Motivation and Programming

Earlier, we discussed student motivation from the perspective of the "student" corner of the didactic triangle. That simple perspective has been problematized and questioned – it has been argued that what students are studying has an important role in motivation (see for example Salili, Chiu, & Hong, 2001).

Hansen and Eddy (2007) have taken the interesting step of surveying their students and directly asking them to rate their engagement with, and frustration with, the various assignments the students do across three courses. They found that frustration and engagement do vary according to the type of task given to the students.

4.1.7 Student Learning of CS in a context

Few education research projects have discussed learning of computer science in a context. One example can be found in Berglund (2005), which identifies complex relationships between the learning and the learning environment in a distributed project course in computer systems. Kinnunen (forthcoming) studies introductory programming courses and proposes models to analyse the full picture of a teaching and learning situation, with the ultimate aim of improving the teaching of programming.

Other projects take a more practical, and less research focused approach. For example, in a project by Tew, McCracken, & Guzdial (2005), exercises are remodelled and the course reorganised to better suit the students study interests.

4.1.8 Students and Teachers

The relationship between teachers and students is a neglected area of computing education research.

Hitchens and Lister (2009) reported on a focus group study of the attitude toward lectures by computing students. One of the outcomes of the focus groups was the importance the students attributed to a positive personal relationship with the lecturer. This is illustrated in the following comments by two students from the focus groups:

> … what makes a good lecture is more the lecturer and his attitude towards giving the lecture. … I've noticed that I've walked out of lectures thinking 'oh that's a good lecture' actually when the lecturer's happy more or less.

> … don't get me wrong because older people can be really happy and really energetic and really passionate. But, you know… I think they get older so they just don't care. They just want to hurry up and teach and get out of there.

The "feeling", or climate, in a class-room was studied by Barker & Garvin-Doxas (2004). They argued, based on their empirical investigations, that the CS class-room can be experienced as a male-dominated impersonal environment with guarded behaviours.

The teacher–student relationship is two-way, but there is probably less research on computing teachers' attitudes to their students than there is on the reciprocal relationship. Kutay and Lister (2006) conducted focus groups with

computing teachers and, amongst other issues, asked how teachers felt about their students. One focus group participant made the following statement about the importance of the emotional connection with students:

> … If you want to be a good teacher, you really have to show the students … that you are passionate about the things you are teaching. The students can very quickly discover the fraud, so you must actually show your love of that material, if that comes across I think half the battle is won.

4.1.9 Teacher- versus Student-Centred

Kember (1997) argued, from a phenomenographic perspective, that the student-centred approach is more advanced, or more complex, in that it presupposes the teacher-centred approach. To focus on the student a teacher must be capable of taking a step 'outside' herself and seeing her acts not as an aim in itself, but in relation to the student. The rather few studies that have quantified these orientations with individual teachers confirm that the student-focused orientation is less common than the teacher-focused one.

The insights from Kember's work tell us that the attitude of the teacher is an important factor in determining how she teaches. It would be interesting to explore what it is that leads some teachers to take the step to seeing their teaching, and the object of their teaching, from the perspective of their students.

5 The Complete Didactic Triangle

In the previous two sections, we have explored parts of the didactic triangle, and its implications for teaching programming. In this section, we consider the whole.

Our first observation, flowing from the previous sections, is that it is hardly surprising that there is not a consensus in our community as to what OOP is and how to teach it, when there are so many different perspectives stemming from different foci on different portions of the didactic triangle. Thus, we teachers "invite" our students to join a community of practice (Wenger, 1998) when the community itself does not share an understanding of what is OOP. Similar results have been found in a study based on questionnaires (Bennedsen & Schulte, 2007).

5.1 Content- versus Student-Centred, again

With the perspective gained from the didactic triangle, perhaps the concepts of content- versus student-centred teaching should not be seen as being in competition. Instead, as the content-centred orientation falls on the Teacher-Programming edge of the didactic triangle, while the student-centred orientation falls on the Teacher-Student edge of that triangle, perhaps it would be more profitable to see them – not mutually exclusive, but instead – as equally necessary aspects of the complete teacher. The teacher who is an expert in their subject but who cannot communicate with her students is perhaps no more or less effective a teacher than the talented communicator who simply doesn't know the content.

5.1.1 Learning as Entering a Culture

As computer scientists and academics we are part of, and carry, a certain culture, with its own values and norms. These values and norms need to be made explicit in the discourse of teaching. Booth (2001) presents learning of programming as a entering a community with its own ways of thinking.

Contrasts between our culture and the students' culture, based on their own experiences of home computers and games, are highlighted in the work of Kolikant (2005), where she argues that errors in students' programmes can have cultural reasons: "Correctness" means something different to students than what it means to us, their teachers.

Liberman, Kolikant and Beeri (2009) is a study spanning the entire didactic triangle. One aspect of the study is the way in which a teacher grapples with her own uncertainty with OOP concepts, all the while attempting to respond quickly to student questions.

In this culturally-oriented research work, all aspects of the didactic triangle are explicit. Not only is the technology and the student explicit in this research, but so is the culture to which the teachers belong.

5.2 Tools, again

Earlier, we discussed the summary, written by Stasko and Hundhausen (2004), of work in program visualization. They described how, prior to the mid-1990s, the focus in that work was on the technology, not the students, but then they went on to add:

> In the mid-1990's, the focus of algorithm visualization research shifted markedly. Rather than concentrating on the development of algorithm visualization technology … researchers began to turn their attention to the pedagogical effectiveness of the technology. The evaluation of algorithm visualization technology became paramount as researchers began to question their intuitions about the utility of algorithm visualizations as learning aids.

Since the mid-1990s, this style of research in algorithm visualization has encompassed the entire didactic triangle. Not only is the technology and the student explicit in this research, but so also are the (previously taken for granted) intuitions of the teachers.

6 Conclusion

Much of the work presented at CS education conferences today focuses upon details or "small picture" issues, such as specific teaching tools or tips and tricks (Simon, 2007a, 2007b; Valentine, 2004). These projects emphasise the corners of the didactic, but these projects serve important needs by offering platforms for discussions between teachers, and by disseminating good teaching experiences. However, our students face *other problems* than those which are addressed by these projects.

Based on the research presented earlier, we argue that an alternative line of research ought to be prioritized. Our arguments are given below:

1. *Educators do not have a shared picture of the fundamentals of object-oriented programming.* Certainly, a discussion is valuable and is a sign of life in the community, but the question of what OOP is might overshadow the question of what our students need and what and how we should teach them.
2. *We tend to base our teaching on our own needs, or our assumptions about the students' needs.* In this paper, we have discussed *how* our knowledge of both our students and ourselves is limited.
3. *We know very little about our students' world and our students' motivations.* We need to meet our students where they are, in order to make our teaching accessible to them, and thereby meaningful. Currently, we only approach them on their terms to a very limited extent
4. *We tend to focus on details instead of the bigger picture*.

In short: we teach and research that which we find important, but what is important to us may not be as important to our students and their learning. Computing education research needs to broaden its focus. Although it is tempting for us to explicitly prescribe certain lines of research, we cannot, and should not, do this – research should offer surprises. But we can hint at which forms of projects we believe are of less importance, and we can nominate research directions that we judge as more important.

The development of new teaching tricks, new "single-user" tools or other "detail-oriented" projects are generally of a limited value. They often prove be beside the point, as these projects are normally based upon the teacher's perception of importance, rather than on the world of the students

When studying student learning, we suggest that researchers should frame their questions in terms of the students' point of view, not the teacher's point of view. However, we also suggest that researchers study the teacher as much as the student.

Many of us, as CS education researchers, have our research training from computer science (or other sciences), we are trained in an "objectivist" or "positivistic" tradition (Cohen, Manion, & Morrison, 2000). Naturally, we bring this competence with us when doing research in CS education. Complementing this research with alternative approaches, stemming from the social sciences, opens new research questions for inspection and would, for example, invite to a further exploration of the students' learning context. (Berglund, Daniels, & Pears, 2006)

Acknowledgements

Raymond Lister's work in computing education research is partially funded by an Associate Fellowship awarded to he and Jenny Edwards from the Australian Learning and Teaching Council (formerly the Carrick Institute).

References

ACM Java Task Force (2006) *Version 1.0*. http://jtf.acm.org/ [Accessed September 2008]

Astrachan, O., Bruce, K., Koffman, E., Kölling, M., & Reges, S. (2005). "Resolved: Objects Early Has Failed". *SIGCSE'05, February 23-27, 2005, St. Louis, Missouri, USA.*, 451-452.

Barker, L., & Garvin-Doxas, K. (2004). Making Visible the Behaviors that Influence Learning Environment: A Qualitative Exploration of Computer Science Classrooms. *Computer Science Education, 14*, 119-145.

Bennedsen, J. (2008). *Teaching and learning introductory programming – a model-based approach* (PhD thesis): Oslo University, Oslo, Norway.

Bennedsen, J., & Schulte, C. (2007). *What does "objects-first" mean? An international study of teachers' perceptions of objects-first.* In proceedings of Seventh Baltic Sea Conference on Computing Education Research, Koli National Park, Koli, Finland. 21 - 30.

Berglund, A. (2005). *Learning computer systems in a distributed project course: The what, why, how and where* (Uppsala Dissertations from the Faculty of Science and Technology Vol. 62). Uppsala, Sweden: Acta Universitatis Upsaliensis.

Berglund, A. (2006). Phenomenography as a way to research learning in computing. *Bulletin of the National Advisory Committee on Computing Qualifications, BACIT, 4*(1).

Berglund, A., Daniels, M., & Pears, A. (2006). Qualitative Research Projects in Computing Education Research: An Overview. *Australian Computer Science Communications, 28*(5), 25 - 34.

Berglund, A., & Lister, R. (2007). *Debating the OO debate: Where is the problem?* In proceedings of Seventh Baltic Sea Conference on Computing Education Research, Koli National Park, Finland. 171 -174.

Biggs, J. B. (2003). *Teaching for quality learning university.* Buckingham: Open University Press/Society for Research into Higher Education. (Second edition)

Booth, S. (2001). Learning to program as entering the datalogical culture: A phenomenographic exploration. In *9th European Conference for Research on Learning and Instruction (EARLI), 28th August–1st September 2001, in Fribourg, Switzerland*.

Bowden, J. & Marton, F. (1998) *The University of Learning: Beyond Quality and Competence in Higher Education*. London: Kogan Page.

Boyer, E. (1997). *Scholarship Reconsidered: Priorities the Professoriate*). Hillsdale, NJ, USA.: Jossey-Bass.

Bruce, Kim. Controversy on how to teach CS 1: a discussion on the SIGCSE-members mailing list *ACM SIGCSE Bulletin*. Volume 37, Issue 2 (June 2005) 111-117.

Butler, M. Morgan, M. (2007) Learning challenges faced by novice programming students studying high level and low feedback. Proceedings of the Conference of the Australasian Society for Computers in Learning in Tertiary Education (ASCiLiTE). Singapore, December 2-5. http://www.ascilite.org.au/conferences/singapore07/procs/procs/butler.pdf [Accessed September 2008]

Cohen, L., Manion, L., & Morrison, K. (2000). *Research Methods in Education (5th Edition)*). London, UK: Routledge Falmer.

Denning, P. (2008) *The Computing Field: Structure*. In Wah, B. W. (Ed.) (2008) Wiley Encyclopedia of Computer Science and Engineering. Wiley-Interscience

Dunkin, M (1990): The induction of academic staff to a university: processes and products. *Higher Education* 20:47-66.

Eckerdal, A. and Thuné, M. 2005. *Novice Java programmers' conceptions of "object" and "class", and variation theory*. In Proceedings of the 10th Annual SIGCSE Conference on innovation and Technology in Computer Science Education (Caparica, Portugal, June 27 - 29, 2005). ITiCSE '05. ACM, New York, NY, 89-93.

Entwistle, N. (1998). Motivation and approaches to learning: Motivating and conceptions of teaching. In B. Brown, S. Armstrong & G. Thompson (Eds.), *Motivating students* (pp. 15-24). Lodon, UK: Kogan Page.

Fincher, S, and Petre, M. (2004) *Computer Science Education Research*, Taylor & Francis.

Fox, D (1983): Personal Theories of Teaching. *Studies in Higher Education*, 8(2):151-163.

Grandell, L., Peltomaki, M., Back, R.-J. and Salakoski, T. (2006). *Why Complicate Things? Introducing Programming in High School Using Python*. In Proc. Eighth Australasian Computing Education Conference (ACE2006), Hobart, Australia. CRPIT, **52**. Tolhurst, D. and Mann, S., Eds. ACS. 71-80.

Gries, D. (2008). *A principled approach to teaching OO first*. In Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (Portland, OR, USA, March 12 - 15, 2008). SIGCSE '08. ACM, New York, NY, 31-35.

Hansen, S. and Eddy, E. 2007. Engagement and frustration in programming projects. *SIGCSE Bull.* 39, 1 (Mar. 2007), 271-275.

Hitchens, M. and Lister, R. (2009). *A Focus Group Study of Student Attitudes to Lectures*. In Proc. Eleventh Australasian Computing Education Conference (ACE 2009), Wellington, New Zealand. CRPIT, **95**. Hamilton, M. and Clear, T., Eds. ACS. 93-100.

Kansanen, P. (1999). Teaching as Teaching-Studying-Learning Interaction. *Scandinavian Journal of Educational Research, 43*(1), 81 - 89.

Kansanen, P., & Meri, M. (1999). The Didactic relation in the teaching-studying-learning process. In *TNTEE Publications* (Vol. 2, pp. 107 - 116).

Kember, D. (1997). A reconceptualisation of the research into university academics' conceptions of teaching. *Learning and instruction, 7*(3), 255 - 275.

Kinnunen, P. (forthcoming). *The instructional process in an introductory programming course from students', teachers', and organizations' point of view - the system theoretical approach to the higher education at Helsinki University of Technology (prel.. title)* (PhD theses in Computer Science). Espoo, Finland: Helsinki University of Technology.

Kinnunen, P., McCartney, R., Murphy, C., & Thomas, L. (2007). *Through the eyes of instructors: a phenomenographic investigation of student success.* In proceedings of The Third International Computing Education Research Workshop, ICER, Atlanta, GA, USA. 61 - 72.

Kolikant, Y. B-D. (2005). *Students' alternative standards for correctness.* In proceedings of ICER '05: Proceedings of the 2005 international workshop on Computing education research, Seattle, WA, USA. 37–43.

Kölling, M., Koch, B., and Rosenberg, J. (1995) *Requirements for a first year object-oriented teaching language*. In proceedings of the twenty-sixth SIGCSE technical symposium on Computer science education, Nashville, Tennessee, USA. pp. 173-177.

Kölling, M. (2007). *I object*. From http://www.bluej.org/mrt/docs/objection.pdf

Kölling, M., Quig, B., Patterson, A., & Rosenberg, J. (2003). The BlueJ System and its Pedagogy *Computer Science Education, 13*(4), 240 - 268.

Liberman, N., Ben-David Kolikant, Y., and Beeri, C. (2009) *In-service teachers learning of a new paradigm: a case study*. In Proceedings of the Fifth international Workshop on Computing Education Research Workshop (Berkeley, CA, USA, August 10-11, pp. 43-50.

Lister, R., Berglund, A., Clear, T., Bergin, J., Garvin-Doxas, K., Hanks, B., et al. (2006). Research Perspectives on the Objects-Early Debate. *SIGCSE Bulletin Inroads, 38*(4), 173 - 192.

Lister, R., Berglund, A., Box, I., Cope, C., Pears, A., Avram, C., et al. (2007). Differing Ways that Computing Academics Understand Teaching. *Australian Computer Science Communications, 29*(5), 97-106.

Ma, L., Ferguson, J., Roper, M., and Wood, M. (2007). *Investigating the viability of mental models held*

by novice programmers. In Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (Covington, Kentucky, USA, March 07 - 11, 2007). SIGCSE '07. ACM, New York, NY, 499-503.

Marton, F., & Booth, S. (1997). *Learning and awareness*. Mahwah, New Jersey, USA: Lawrence Erlbaum Associates.

Northedge, A (1976): Examining our implicit analogies for learning processes. *Programmed Learning and Educational Technology* 13(4):67-78.

Pears, A., Berglund, A., Eckerdal, A., East, P., Kinnunen, P., Malmi, L., et al. (2007). *What's the Problem? Teacher's experience of student learning.* In proceedings of 7th Baltic Sea Conference on Computing Education Research, Koli Calling, Koli, Joensuu, Finland. 207-211.

Ramsden, P. (2003). *Learning to teach in higher education* 2nd ed.). London, UK; New York, NY, USA: Routledge.

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

Rowland, S. (2000) *The enquiring university teacher*. Buckingham: SRHE and Open University Press.

Ryan, R., & Deci, E. (2000). Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions. *Contemporary Educational Psychology, 25*(1), 54 - 67.

Salili, F., Chiu, C., & Hong, Y. (Eds.). (2001). *Student Motivation: The Culture and Context of Learning*. New York, MY, USA: Kluwer Academic.

Samuelowicz, K, & Bain, J (1992): Conceptions of teaching held by academic teachers. *Higher Education*, 24:93-111.

Shulman, L. S. (1987) Knowledge and Teaching: Foundations of the New Reform. *Harvard Educational Review*, 57:1(Feb) pp. 1-22. Also included in Shulman (2004), as chapter 5 (same title), pp. 83-.113

Shulman, L. (1989) Towards a Pedagogy of Substance, *AAHE Bulletin*, 41(10) pp. 8-13. Also included in Shulman (2004), as chapter 7 (same title), pp. 127-138.

Shulman, L. S. (2004) *Teaching as community property: essays on higher education*. San Francisco: Jossey-Bass, 2004.

SIGCSE. (2004a). SIGCSE-MEMBERS Archives March 2004, Week 3. Retrieved February, 2008, from http://listserv.acm.org/scripts/wa.exe?A1=ind0403c&L=SIGCSE-members

SIGCSE. (2004b). SIGCSE-MEMBERS Archives March 2004, Week 4. Retrieved February, 2008, from http://listserv.acm.org/scripts/wa.exe?A1=ind0403d&L=SIGCSE-members

Simon. (2007a). A classification of recent Australasian computing education publications. *Computer Science Education, 17*(3), 155 - 169.

Simon. (2007b). *Koli Calling comes of age: an analysis.* In proceedings of Seventh Baltic Sea Conference on Computing Education Research (Koli Calling), Koli National Park, Finland. 119 - 126

Simon, S., Carbone, A., de Raadt, M., Lister, R., Hamilton, M., and Sheard, J. 2008. *Classifying computing education papers: process and results.* In Proceeding of the Fourth international Workshop on Computing Education Research (Sydney, Australia, September 06 - 07, 2008). ICER '08. ACM, New York, NY, 161-172.

Stasko, J and Hundhausen, C. (2004) Algorithm Visualization. Chapter 17 in Fincher and Petre (2004).

Tew, A. E., McCracken, M., & Guzdial, M. (2005). *Impact of Alternative Introductory Courses on Programming Concept Understanding.* In proceedings of 2005 international workshop on Computing education research Seattle, WA, USA. 25 - 35.

Trigwell, K., Prosser, M., Martin, E., & Ramsden, P. (2005). University teachers' experiences of change in their understanding of the subject matter they have taught. *Teaching in Higher Education, 10*(2), 251-264.

Valentine, D. (2004). *CS educational research: a meta-analysis of SIGCSE technical symposium proceedings* In proceedings of the 35th SIGCSE technical symposium on Computer science education Norfolk, Virginia, USA 255-259

Wenger, E. (1998). *Communities of Practice. Learning, Meaning, and Identity*). Cambridge, UK: Cambridge University Press.