Debating the OO debate: Where is the problem?

Anders Berglund
Uppsala Computing Education Research Group
Department of Information Technology
P.O. Box 337
SE- 751 05 Uppsala
Sweden
Anders.Berglund@it.uu.se

Raymond Lister
Faculty of Information Technology
University of Technology Sydney
P.O. Box 123,
Broadway 2007, Sydney
Australia
raymond@it.uts.edu.au

Abstract

In this paper we discuss problems related to the teaching of object-oriented programming (OOP). We argue that more research on how the computer science teacher understands OOP would be beneficial. Our argument takes its point of departure in three sets of studies: (1) an ongoing study on how computer science teachers understand core concepts of OOP, (2) a study of how the teaching of OOP is discussed within the CS community, and (3) a set of studies that discuss the different ways in which CS teachers experience their teaching. This paper reports on an ongoing study of the different ways in which computing science teachers understand object-oriented programming, and what they mean when use the term *objects first*. The phenomenographic research approach has been applied to the analysis of a discussion that occurred in the SIGCSE-members mailing list. Two understandings of *objects first* have been identified: (1) as an extension of imperative programming, and (2) as conceptually different from imperative programming. These two understandings are illustrated via the differing ways in which computing science teachers use the term *polymorphism*.

Keywords: Object-oriented programming, objects-first, phenomenography.

1 Introduction

Object-oriented programming (OOP) is hard for students to learn and hard for teachers to teach. The learning and teaching of OOP is often discussed at Computing Education Research (CER) conferences, and articles about it appear regularly in the CER journals.

Many attempts have been made to improve the learning and teaching of OOP. Among theses initiatives, to name but a few, are: environments for teaching Java (Kölling, Quig, Patterson, & Rosenberg, 2003); research into the students' experience of learning OOP (Bruce et al., 2004; Eckerdal & Berglund, 2005); explorations through the lenses of various learning theories (Ben-Ari, 2001, 2004; Robins, Rountree, & Rountree, 2003); analysis of the properties of languages (Mannila, Peltomäki, & Salakoski, 2006); and changes to teaching approaches (Pedroni & Meyer, 2006). Despite these many initiatives, the learning and teaching of OOP remains a problem, with low passing rates and high attrition rates.

2 Broadening the perspective on the debate

In this paper we argue that our teaching community would benefit by broadening the current debate, which emphasises the technology of OOP and the learning of CS students, to include an examination of ourselves, the teachers, and our own understanding of what OOP means. We will base our argument on an ongoing project on the various ways in which CS teachers understand some core concepts in OOP, as illustrated in an analysis of a 2004 SIGCSE mailing list debate on objects early (Lister et al., 2006).

2.1 How do CS teachers understand core concepts?

In the third and fourth weeks of March 2004, there was a vigorous discussion about teaching OOP on the SIGCSE-members mailing list (SIGCSE, 2004a & 2004b). A list of the 99 postings is given elsewhere (Lister et al., 2006b). The discussion focused on when object orientation should be introduced in beginners' programming courses: should objects be introduced early (*objects first*), or should objects be preceded by imperative programming (*imperative first*)? An ITiCSE working group, chaired by the two authors of this paper and Tony Clear, Auckland University of Technology, New Zealand, studied this discussion from several different research perspectives (Lister et al., 2006).

When reading and re-reading the mailing list postings, we came to realize that the discussants understood fundamental concepts in different ways and thus put different meanings into terms such as *objects first* and *polymorphism.* To explore this question, we decided to do a phenomenographic analysis of *how the list discussants understood objects first.*

2.1.1 Phenomenography

Phenomenography is an empirically based, pedagogically anchored research approach, aimed at exploring how something is understood by a group of people (Marton & Booth, 1997). The outcome of a phenomenographic

research project is an ordered description of the different meanings that the phenomenon under investigation (in our study *objects first*) has for the members in the group. Phenomenography is a qualitative, non-positivistic approach, which during the past few years has come to play an important role in CER (Berglund, 2006; Berglund, Box, Eckerdal, Lister, & Pears, 2008).

2.1.2 O*bjects first* is understood in two ways by the discussants

Our preliminary results show that *objects first* is understood in two different ways, corresponding to two phenomenographic categories. We wish to stress that these categories do not express the main theme of the debate, objects first or imperative first. They 'only' describe different meanings of the first of these stands.

Objects first can be understood

1. as an extension of imperative programming,
2. as something conceptually different from imperative programming.

2.1.3 Understanding the categories

The phenomenographic categories are constructs that summarise and 'abstract' different meanings. They do not illustrate individuals; an individual can see something in one or many ways. A category can be analysed into (or 'is constituted by') different Dimensions of Variation (DoV) or parameters, where each DoV can take certain 'atomic' values, or be un-instantiated.

Table 1 shows the two categories, their dimensions of variation and the values of these dimensions. Since the purpose of this paper is to encourage a debate, rather than to present the final outcome of a research project, we will here only illustrate the values of one DoV, *Polymorphism*, with two quotes, one corresponding to each value of the DoV, and thereby to different categories. A further discussion on the empirical data will be published in the future.

The first, by McConnell, illustrates *Polymorphism understood as different objects:*

> I still think that selection, repetition, variables, arrays, are still critical foundational CS1 topics. […] I firmly believe that students leaving a CS1 class should have these foundational topics first but with an understanding of objects. Decker and Hirshfield's "The Object Concept" had its problems, but I think it is a good book because: (1) its of a reasonable size, (2) it introduces objects early, (3) it concentrates on foundational topics, and (4) it introduces "advanced" object concepts, such as operator overloading, inheritance, and polymorphism at the end.

We interpret the second, by Joe Berger, as an indication of the understanding *Objects interact polymorphically*:

> Note that this is consistent with my thesis that polymorphism "means" that an object just knows what it is and behaves like it does without fuss or bother. […] Note that I don't explain all this to students unless they ask (rarely) in any early course. Polymorphism just "works" consistent with the "object is in control" metaphor …

As predicted by phenomenography, there is a hierarchical relationship between the categories. The second category is more advanced (in the phenomenographic sense) than the first, since the second presupposes the first – if imperative programming is not known to someone, then it is impossible for that person to view OOP as something different from imperative programming,. However, no one (at least today) needs to understand *objects first* in order to understand imperative programming. The phenomenographic theory predicts this kind of hierarchical structure. It indicates that the categories are related and thus that they are categories of the same phenomenon.

2.1.4 Our interpretation of the findings

We argue that these different ways of understanding *objects first* not only relate to teaching, but also, and more importantly, describe different ways of understanding object-orientation. We base this argument in the content of the categories, in our reading of the mailing list discussion and in the hierarchical structure of the categories. The arguments for *objects first* that are given along the lines of category one are not open to an interpretation of object-orientation as an interaction or a calculation on its own right. Thus they show a more delimited view of object-orientation.

Although these results are preliminary, we do not expect any important changes in our future work. We will investigate further concepts and might 'fine-tune' the description of the categories and elaborate on their constituents. We thus believe that the fact that *objects*

		Category 1. Objects first as an extension of imperative programming	**Category 2.** Objects first as something conceptually different from imperative programming
DoV1	Program execution	Objects are passive and are used when the program is run	Objects are active. Object interaction gives the algorithm
DoV2	Polymorphism	Polymorphism as different objects	Objects interact polymorphically
DoV3	Modifications	Modification as changing code	Modifications as adding to holes and hooks

Table 1. The dimensions of variation of *objects first*. In the column of each category, the values of the Dimensions of Variation (DoV) are shown.

first has two fundamentally different meanings, and the fact that there are corresponding underlying meanings in the interpretation of object-orientation, are stable.

The question we now ask ourselves is in what ways these contradictions of the understanding of object-oriented programming within our community influence our teaching and, ultimately, our students.

One could argue that our different interpretations do not constitute a problem for our students' learning, since they normally meet only one teacher and thus only one point of view. Our answer to this imaginary argument is related to the integrity of computer science. If it does not matter which of these interpretations our students meet, what is then the core of computer science?

2.2 Insights from an analysis of the objects early debate

In the phenomenographic (Marton & Booth, 1997) portion[1] of the previously mentioned multiple research perspective analysis of the e-mail discussion (Lister et al., 2006), we explored what the different arguments in the debate focused upon. The purpose was to reveal what the discussants 'talked about', and through this to better understand both the debate and its topic.

According to our analysis, the arguments in favour of an early introduction focused on three major, incommensurable, *themes*: (a) a narrow domain in focus, (b) a broad domain in focus, and (c) pedagogy in focus. Each of these themes could then be argued for, or understood, in four qualitatively different ways. For example, arguments in theme *a* could be categorised into the following categories: (a1) particular Java features, (a2) specific CS constructions, (a3) teaching, and (a4) students as students. In the paper describing these findings, we also demonstrate that the categories form a hierarchical structure and that the discussions in favour of imperative early can also be described according to a similar pattern.

In our conclusions, we argue that this structure helps in understanding the debate:

> With this structure of themes and categories, it is easy to see that several misunderstandings have their origins in a bad match between two arguments. That is, two participants might believe they disagree, when in fact they are arguing about different things. For example, one participant might be focusing on language features, while another is arguing about an aspect of pedagogy.
>
> The twelve categories, and the interrelationships of the categories, reveal the complexity of this discussion. It is not a discussion solely about programming languages or about how OO should be taught. Statements arguing that the 'solution' lies in a single concept (such as for example a new teaching tool) are oversimplifications. The reality is more complex. A broad range of questions need to be further analysed and discussed before a community consensus will emerge. (Lister et al., 2006, p. 156)

2.3 Teachers' experience of their teaching

In a meta-study, Kember (1997) reviews and condenses a set of independent studies on how teachers experience their own teaching. He states that this body of research shows a distinction between two broad orientations[2]: teacher-centred/content-oriented and student-centred/ learning-oriented. Recently these orientations have been confirmed for teachers in CS (Lister et al., 2007; Pears et al., in press).

Kember argues, 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[3] 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 important for how she teaches. It is worth exploring what it is that makes some teachers take the step to see their teaching and the object of their teaching from the perspective of their students. By learning about this development, we learn something important about the CS teacher.

3 Summary

We believe that our discussions in section 2 pose more questions than they answer about our community and our own relation to OOP. The common thread in these three discussions is the community. In section 2.1 we demonstrate that the members of the community, the teachers, understand key concepts in different ways. In the following section (2.2), our argument is that the debate that takes place between the members of the community is often carried out in a way that is too 'simple' to be capture the complexity of the issue. Finally, in section 2.3, we show that we know surprisingly little about the teacher, despite far-reaching evidence of her importance.

---

[1] This portion of the paper was mainly authored by Anders Berglund, with the support of Raymond Lister.

[2] Kember uses the term *orientation*. An alternative term, developed from the language of computer science and object-oriented programming, is *super-category.*

[3] We have chosen to refer to a teacher as "her" throughout this paper. Certainly, our claims are as valid (or as invalid) for a male teacher.

4 Open questions

Our insight, when working on the different constituents that form this paper, is that more research is needed concerning the CS teacher, her understanding of herself, her thoughts about her students, and the relationship between these entities. More precisely, what is it that we need to learn about the teacher? And how should such research be performed?

5 References

Ben-Ari, M. (2001). Constructivism in Computer Science Education. *Journal of Computers in Mathematics and Science Teaching, 20*(1), 45 - 73.

Ben-Ari, M. (2004). Situated Learning in Computer Science Education. *Computer Science Education, 14*(2), 85 - 100.

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., Box, I., Eckerdal, A., Lister, R., & Pears, A. (2008). *Learning educational research methods through collaborative research: The PhICER initiative.* In Simon & M. Hamilton (Eds.), Proceedings of the Tenth Australasian Computing Education Conference (ACE 2008), Wollongong, NSW, Australia. CRPIT, 78, 35 - 42.

Bruce, C., Buckingham, L., Hynd, J., McMahon, C., Roggenkamp, M., & Stoodly, I. (2004). Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university. *Journal of Information Technology Education, 3*, 143 - 160.

Eckerdal, A., & Berglund, A. (2005). *What Does It Take to Learn 'Programming Thinking'?* In Proceedings of the 1st International Computing Education Research (ICER) Workshop, Seattle, WA, USA, 135 -143.

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

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

Lister, R., Berglund, A., Box, I., Cope, C., Pears, A., Avram, C., Bower, M., Carbone, A., Davey, B., de Raadt, M., Doyle, B., Fitzgerald, S., Mannila, L., Kutay, C., et al. (2007). Differing Ways that Computing Academics Understand Teaching. *Australian Computer Science Communications, 29*(5), 97-106.

Lister, R., Berglund, A., Clear, T., Bergin, J., Garvin-Doxas, K., Hanks, B., Hitchner, L., Luxton-Reilly, A., Sanders, K., Schulte, C., & Whalley, J. (2006). Research Perspectives on the Objects-Early Debate. *SIGCSE Bulletin Inroads, 38*(4), 173 - 192.

Lister et al. (2006b) ITiCSE 2006 Working Group: Research Perspectives on the Objects-Early Debate. http://wwwstaff.it.uts.edu.au/~raymond/iticse06workinggroup/ [Feb. 2008]

Mannila, L., Peltomäki, M., & Salakoski, T. (2006). What about a simple language? Analyzing the difficulties in learning to program *Computer Science Education, 16*(3), 211 - 227.

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

Pears, A., Berglund, A., Eckerdal, A., East, P., Kinnunen, P., Malmi, L., McCartney, R., Moström, J. E., Murphy, L., Ratcliffe, M., Schulte, C., Simon, B., Stamouli, I., & Thomas, L. (in press). *What's the Problem? Teacher's experience of student learning.* In Proceedings of the 7th Baltic Sea Conference on Computing Education Research, Koli Calling, Koli, Joensuu, Finland.

Pedroni, M., & Meyer, B. (2006). *The inverted curriculum in practice.* In Proceedings of the 37th SIGCSE technical symposium on Computer science education Houston, TX, USA, 481 - 485

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

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

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

Conferences in Research and Practice in
Information Technology

Volume 88

Koli Calling 2007

AUSTRALIAN
COMPUTER
SOCIETY

Koli Calling 2007

Proceedings of the
Seventh Baltic Sea Conference on Computing Education Research,
Koli National Park, Finland, 15-18 November 2007

Raymond Lister and Simon, Eds.

Volume 88 in the Conferences in Research and Practice in Information Technology Series.
Published by the Australian Computer Society Inc.

Published in association with the ACM Digital Library.

Koli Calling 2007. Proceedings of the Seventh Baltic Sea Conference on Computing Education Research, Koli National Park, Finland, 15-18 November 2007

Conferences in Research and Practice in Information Technology, Volume 88.

Editors:
Raymond Lister
Faculty of Information Technology
University of Technology Sydney
Australia
E-mail: raymond@it.uts.edu.au

Simon
School of Design, Communication and Information Technology
University of Newcastle
Australia
E-mail: simon@newcastle.edu.au

Series Editors:
Vladimir Estivill-Castro, Griffith University, Queensland
John F. Roddick, Flinders University, South Australia
Simeon Simoff, University of Technology, Sydney, NSW
crpit@infoeng.flinders.edu.au

Publisher: Australian Computer Society Inc.
PO Box Q534, QVB Post Office
Sydney 1230
New South Wales
Australia.

Conferences in Research and Practice in Information Technology, Volume 88
ISSN 1445-1336
ISBN 978-1-920682-69-9

~~Printed April 2008 by Griffith University~~ Uni Print, Nathan Campus, Kessels Rd. Nathan, 4111, QLD, Australia.
Cover Design by Modern Planet Design, (08) 8340 1361.

The *Conferences in Research and Practice in Information Technology* series aims to disseminate the results of peer-reviewed research in all areas of Information Technology. Further details can be found at http://crpit.com/.