Threshold Concepts in Computer Science: Do they exist and are they useful?

Jonas Boustedt
Department of Mathematics, Natural, and Computer Science Högskolan i Gävle
S80176 Gävle, Sweden
bjt@hig.se

Anna Eckerdal
Department of Information Technology
Uppsala University
Uppsala, Sweden
Anna.Eckerdal@it.uu.se

Robert McCartney
Department of Computer Science and Engineering
University of Connecticut
Storrs, CT USA
robert@cse.uconn.edu

Jan Erik Moström
Department of Computing Science
Umeå University
901 87 Umeå, Sweden
jem@cs.umu.se

Mark Ratcliffe
Department of Computer Science
University of Wales
Aberystwyth, Wales
mbr@aber.ac.uk

Kate Sanders
Department of Math and Computer Science
Rhode Island College
Providence, RI USA
ksanders@ric.edu

Carol Zander
Computing & Software Systems
University of Washington, Bothell
Bothell, WA USA
zander@u.washington.edu

ABSTRACT

Yes, and Yes.

We are currently undertaking an empirical investigation of "Threshold Concepts" in Computer Science, with input from both instructors and students. We have found good empirical evidence that at least two concepts—Object-oriented programming and pointers—are Threshold Concepts, and that there are potentially many more others.

In this paper, we present results gathered using various experimental techniques, and discuss how Threshold Concepts can affect the learning process.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computers and Information Science Education—*Computer Science Education*

General Terms

Measurement, Experimentation

Keywords

Threshold Concepts, learning theory

SIGCSE'07, March 7–10, 2007, Covington, Kentucky, USA.

1. INTRODUCTION

Computer science is a young, rapidly changing discipline; we have had relatively few years to study the ways in which students learn and how to help them most effectively. Meyer and Land [13] have proposed using "Threshold Concepts" as a way of characterizing particular concepts that might be used to organize the educational process. The idea has the potential to help us focus on those concepts that are most likely to block students' learning [4]. A subset of the core concepts in a discipline, Threshold Concepts are defined by Meyer and Land [13] are:

- *transformative*: they change the way a student looks at things in the discipline.
- *integrative*: they tie together concepts in ways that were previously unknown to the student.
- *irreversible*: they are difficult for the student to unlearn.
- potentially *troublesome* (as in [15]) for students: they are conceptually difficult, alien, and/or counter-intuitive.
- often *boundary markers*: they indicate the limits of a conceptual area or the discipline itself.

This paper describes an ongoing project aimed at empirically identifying Threshold Concepts in computer science. In a multi-national, multi-institutional study, we have gathered data from both educators and students. The paper outlines experimental techniques, issues raised, and results to date. Information about how Threshold Concepts fit with other learning theories and how they can be put into context

with other ways to organize computer science concepts can be found in [5].

Section 2 describes the techniques we have used to gather data about potential Threshold Concepts from the perspective of both instructors and students. In Section 3 we present the results of a preliminary analysis of the data, including the identification of two Threshold Concepts in computing with some associated evidence. Section 4 looks at implications for educators and finally, in Section 5, we present our preliminary conclusions and discuss the future directions of this research effort.

2. DATA GATHERING AND ANALYSIS

The study began by gathering data from instructors in computer science. This was followed by a brief analysis and a much more in-depth study of graduating seniors.

2.1 The instructors: informal interviews and surveys

In June 2005, at the Conference on Innovation and Technology in Computer Science Education (ITiCSE), 36 instructors from nine countries were interviewed and asked for suggestions of concepts that meet the criteria for a Threshold Concept. These interviews were unstructured and done in a fairly conversational style.

From these, we gained much insight. First, the idea of Threshold Concepts is compelling: nearly everyone we spoke with was immediately interested. In total 33 concepts were suggested, with the most popular being: levels of abstraction; pointers; the distinction between classes, objects, and instances; recursion and induction; procedural abstraction; and polymorphism. Second, while some concepts came up again and again, there was no universal consensus.

In November 2005, we gave a poster and had discussions with researchers at the Koli Calling 2005 Conference on Computer Science Education in Finland [12] and used a questionnaire and interviews to gather data more systematically from conference participants. The results were similar to those collected at ITiCSE but it was quite apparent that instructors focus on "difficult to learn" more than any other aspects of the concepts they discuss.

2.2 The students: semi-structured interviews

Given the tentative list of concepts derived from instructors and the literature ([1, 2, 3, 17, 18] e.g.), we began to investigate the question of whether these—or any—concepts are experienced by students as thresholds.

We chose to initially interview graduating students, since they were more likely than novices to have mastered the relevant concepts, and to have developed some perspective. So far we have completed 16 interviews with students at seven institutions in a total of three countries. The script for these interviews is given in Figure 1.

We started by addressing the *troublesome* criterion, asking students for concepts they found difficult at first (places where they were initially "stuck"). From these, we selected one concept to pursue in depth and addressed the *transformative, integrative,* and *irreversible* criteria in that context. We did not examine the *boundary marker* criterion, as it is related more to disciplinary boundaries and less to individual experience. The interviewers agreed in advance on a list of five Threshold Concept candidates (control structures, thinking sequentially, parameters, objects, memory model) that had come up repeatedly in instructor interviews. If the student mentioned one of those, that concept was chosen.

1. Could you tell me about something where you were stuck at first but then became clearer? *(Subject answers <X>.)*

   The rest of this session will now focus on <X>.
2. Can I start by asking you to tell me your understanding of <X>?
3. Assume that you were explaining <X> to someone just learning this material, how would you do it?
4. Tell me your thoughts, your reactions, before, during and after the process of dealing with <X>.
5. Can you tell me what helped you understand <X>?
6. Can you describe how you perceived/experienced <X> while you were stuck and how you perceived/experienced it afterwards?
7. Based on your experience, what advice would you give to help other students who might be struggling with <X>?
8. Please tell me what other things you need to understand in order to gain a good understanding of <X>.
9. Can you tell me how your understanding of <X> has affected your understanding of other things?
10. Was your understanding of <X> something that you had to keep reviewing or having learned it once were you OK with it?
11. Describe how and in what context you have used <X> since you learned it?
12. Is there something more you want to tell me about <X>?
13. To finish the interview, can you tell me whether there are any other things where you were stuck at first but then became clearer? I promise I won't ask you about them in detail!

Figure 1: Script Excerpt for Student Interview

The aim of this deeper investigation was twofold. It enabled us to gather evidence as to whether specific concepts met the requirements for Threshold Concepts (Questions 1, 4, 6, 8, 9, 10 in Figure 1). In addition, it gave us data for an analysis (in progress) of graduating students' understanding of central concepts (Questions 2, 3, 5, 12 in Figure 1).

For analysis, the student interviews were transcribed verbatim and where necessary, translated into English by the interviewer.

3. THRESHOLD CONCEPTS IN COMPUTER SCIENCE

Of the concepts discussed in depth by the students, we selected two that seemed promising based on the interview content and closely examined the interviews regarding those concepts. For both concepts – object-oriented programming (OOP) and pointers – we found evidence that they satisfy the criteria for Threshold Concepts.

3.1 Object-oriented programming

Object-oriented programming is experienced as difficult both to teach [8, 16] and to learn [9, 19]. When Eckerdal and Thuné [6] interviewed first-year students after their first programming course on their understanding of object-oriented concepts, many students stated that they found the concepts troublesome despite great effort to learn them.

The interviews give further evidence of OOP as a Threshold Concept. One student subject discussed OOP as *troublesome* to learn:

> *Subject8:* Stuck at first – I would have to say the initial object-oriented programming. Knowing how classes communicate, how you communicate between classes and really understanding how objects work...

The researcher performing the interview asks the subject about the *integrativeness* of OOP later in the interview by asking if once they understood OOP, were there other things they then understood. The subject discusses a multi-threaded programming course.

> *Subject8:* Well, for instance, the class we did for software engineering, what we did right now, the server that I wrote, each client that connects to the server, I thought of it almost as an object, which it is basically. And then that client connection would be held on to while waiting for other connections. And then there'd be this huge array of connections. And then, I mean, that wasn't that difficult for me to grasp that concept just because we'd kind of went over it in class, but I just think understanding object-oriented programming helped me to understand that there was this group of objects, group of threads, group of clients, whatever.
>
> *Interviewer:* That they were all working together?
>
> *Subject8:* That were all working together, exactly. And understanding object-oriented programming, I think, made that easy to learn; easy to understand.

This subject learned about OOP, then later learned about multi-threaded programming, and perceived a real connection at a fairly abstract level.

Another student also indicated that object-oriented programming is *troublesome* to learn:

> *Subject6:* [...] object oriented programming was one thing for example that took a long time before ... it clicked. Why and how it should be used.

The researcher later seeks to find out from the same student if object-oriented programming is *irreversible* by asking if the student had to review the OOP material.

> *Subject6:* Yes, I need to review sometimes, this is completely clear [...], often it's just syntactical details, [...] basic stuff is there, I've mainly used Java so, sure I'm a little bit stuck in those tracks but I can usually bring everything with me and just transfer it to C++ for example, [...]
>
> *Interviewer:* The big stuff is there so to speak
>
> *Subject6:* Yes, I can get, sometimes it can take, or yes some mistakes before I remember that, "right, those", [...] I can forget, to make some mistakes in the beginning but as long as I've once known and done it correctly some time then it usually comes back.

The student explains that some syntactic details might be forgotten in a specific object-oriented programming language, but "the basic stuff is there" (the object oriented paradigm) and "it usually comes back", implying object-oriented programming is *irreversible*.

The quote also shows the close relation between the *irreversible* and the *integrative* aspects of this Threshold Concept. The student explains that he/she can use the knowledge gained from programming in one language and transfer it to another language.

Later in the interview the researcher discusses the *transformative* aspect of learning object-oriented programming by asking about the difference in how the student looks at problems and their solutions before and after:

> *Subject6:* Yes, it's like day and night, before I came here I had ... I couldn't ... abstract the problems on, well to a very small extent perhaps, but today it's ... I can identify the problems usually in a very short time, unless it's very complex and difficult to understand but today I only see small sub-problems and ... usually simple solutions to them. Before it was just one large program that ... I solved sequentially in some way and the programs looked like that ...

Discussing the same topic the interviewer later asks about problem solving, about what role OOP played:

> *Subject6:* It simplifies it, even if I don't use an OO language the OOP way of thinking can help a lot in ... in some way ... well, you can give a lot of data, you can give it some kind object status even if it doesn't have its own methods etc, in that way...

The student explains that he/she looks at programming in a completely new way after learning object-oriented programming. The knowledge has *transformed* how the student looks at problems. This is consistent with Luker [10], who argues that learning the object-oriented paradigm, "requires nothing less than [a] complete change of the world view."

3.2 Pointers

The second concept identified as a potential Threshold Concept was the use of pointers, particularly when used as parameters. That this concept can be *troublesome* is illustrated by the following excerpt:

> *Subject13:* And so when you implement pointers and see then you're like okay I need to figure out how I modify that and it affects the memory. And then if I reference the memory I get what back. And then you start passing the arguments. And you have to understand passing by reference or passing by value. And a lot of those were definitely big hurdles right in the beginning because I didn't – it was just – I guess too theoretical of a concept for me to really put in practical sense.

The student describes the difficulties in understanding pointers in general and how they work with parameter passing.

Another student affirms that pointers can be *troublesome*:

> *Subject3:* I know that pointers are something that a lot of students have trouble with. [...]

Not only can we see that this student found pointers *troublesome*, but also that they are *integrative* and *transformative*:

> *Subject3:* And I think once you've realized that a pointer is just pointing to a place in memory, it's just pointing to a location, that's all it is. Then I think everything will flow from that. Yeah, because you realize then that the object itself is just a place in memory too.
>
> *Interviewer:* So before you weren't even thinking about memory so much.
>
> *Subject3:* Exactly. I didn't at all. Like in Java, I didn't think about memory nearly so much. I mean I knew that certainly memory was allocated, that memory was allocated with its variables and attributes and that kind of stuff. But I didn't ... when I was writing a program, I didn't ever think about what was happening underneath. Especially garbage collection.

Once the student understood how pointers work he/she was able to use this knowledge to explain how objects and references are implemented, thus getting an improved understanding of how Java works.

In another part of the interview the same student described how the understanding of pointers has helped in other subjects:

> *Subject3:* Well, as I was saying, in the hardware class and in Operating Systems, we definitely discussed pointers and I used it both conceptually and also in, well not in in Operating Systems, but in the hardware class in assembly language, we definitely used pointers.
>
> We definitely were dereferencing all the time in assembly language, so when we were, for example, writing to an address register, we would have to dereference it in order to get at the address to find out what was going on at that particular address in memory. So, definitely I used it again and again.

The clearest statement that the concept of pointers is *unforgettable* comes from an interview with Subject13 when asked if the understanding gained needed to be reviewed to be remembered:

> *Subject13:* The syntax I would have to review, guaranteed. The syntax is –it's a little – it's syntax. But the basic idea of passing by reference or value; no, once I understood that I – every time it's mentioned I immediately know and understand – I can see a picture – a diagram in my head of what I'm supposed to do. What – how the effect will work. So the concept was not lost at all.

4. IMPLICATIONS FOR EDUCATORS

By their nature, Threshold Concepts have a number of implications for educators. These are key concepts that students must understand to make progress in computer science: failing to gain this understanding and the associated lack of progress can lead to frustration, a poor understanding on how the discipline fits together, and increased attrition of students. It is important, therefore, that we understand these concepts, and how they are learned, in detail. It is useful to know that students tend to become stuck on a particular concept, but the deeper understanding of the student experience—how students get unstuck, and why some students get unstuck (or perhaps never get stuck at all) while others remain stuck—should provide ideas on how to help students to make progress toward understanding that concept. Knowing what concepts a particular Threshold Concept integrates can provide the instructor with a context in which the concept might effectively be taught.

Davies suggests that when teachers proceed on the incorrect assumption that students have learned a Threshold Concept, it may cause students to go forward with surface-level learning:

> In the absence of this understanding students can only resort to learning surface routines and language in the hope that they can pass this off as real understanding.[4]

More interesting still, a result found in economics ([14], as reported in [13]) was that teaching Threshold Concepts in simplified ways can have similar bad effects: the simplified version may be treated as "ritualized" knowledge–a superficial understanding–that makes it *more* difficult for the student to ultimately learn the concept. What these results suggest is how important it is that teachers accurately monitor the level of understanding of these concepts by their students, expecially if "going through the threshold" (in a deep-learning sense) is necessary for their students to progress.

Looking on the bright side, however, Threshold Concepts can provide positive opportunities to instructors. First, they may help us manage the ever-growing curriculum. For example, Computer Curricula 2001 (CC2001) [7] included 63 core units, each made up of several topics. If we can identify a relatively small number of Threshold Concepts within the curriculum, instructors will be able to focus their efforts on helping students with those concepts. Second, because Threshold Concepts are integrative, an instructor can use them to help students see connections within the discipline that transcend individual course boundaries.

In some cases, it will be necessary to revisit the same Threshold Concepts over multiple courses. We saw evidence of this for both OOP and pointers. In object-oriented programming, we saw a student's understanding develop from seeing objects as a simple encapsulation mechanism to an appreciation for design patterns; for pointers (and the associated memory issues), the basic understanding learned in introductory programming became richer in subsequent courses. The students reported that crossing the thresholds for them was a gradual process, not necessarily an "aha" moment.

One more observation from the interviews that applies more generally than to Threshold Concepts, is that students (in retrospect) appreciate the value of individual work. Nearly all, when asked to give advice to other students who are having problems with a particular concept, included things like "you need to work on your own", "you need to sit down and think about what you're doing", and "sit down and work it out on paper and really understand what happens." Our own teaching experience is that this understanding is not shared by all novice students.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we describe Threshold Concepts. These are a subset of the core concepts in a discipline that are transformative, integrative, irreversible, and potentially trouble-

some. If instructors don't take care in introducing these concepts to our students and monitoring their understanding, students may fail to move on, or move on with only surface knowledge of the concept.

We also describe our initial empirical investigations into Threshold Concepts. We used both informal interviews and questionnaires to obtain data from instructors and scripted, semi-structured interviews to obtain data from students.

Based on these investigations, we present evidence that Threshold Concepts do exist in computer science. We have identified two Threshold Concepts, or perhaps broad areas within which thresholds exist: pointers and object-oriented programming. These were the terms our subjects used for concepts they identified, but these concepts – object-oriented programming in particular – are very broad. A close reading of the interviews suggests that more specific Threshold Concepts might include the way in which objects work together (i.e., concurrency), or the ability to see large problems as composed of a set of small sub-problems.

In future work, we plan to investigate these more specific concepts, and also some of the other candidate Threshold Concepts that have been mentioned in our data. One particularly intriguing example is the notion of translation from one representation to another – it is certainly pervasive within computer science, but only one of our instructor subjects (and none of the students) mentioned it as a candidate.

In addition, we have not yet analyzed the variations in understanding associated with the concepts discussed here. This analysis will provide an outcome space with qualitatively different ways to understand a certain concept (as in [11]), information that should be immediately useful to instructors.

An area that we will be investigating is whether the learning of a Threshold Concept is an identifiable stage that all learners go through, or whether it is more of an individual phenomenon. Some of our data suggest that the Threshold Concepts that a student identifies are strongly influenced by his or her learning experiences–e.g., the only two subjects who suggested "How a processor control unit works" had taken a course where they were required to design one. In addition, it is possible that a Threshold Concept may seem to be universal if the concept is overly broad, as mentioned above regarding to object-oriented programming: subjects may agree on the broad concept, but this may be due to their experiencing different more-specific concepts as thresholds. We will address these issues by undertaking interviews with a finer-grained focus on particular concepts.

We are also planning to interview novices, to see how their perspective compares with the graduating students'. Finally, once Threshold Concepts have been precisely identified, the next step will be to design curricula and assessment tools to help student cross these thresholds more easily.

6. REFERENCES

[1] ACM/IEEE-CS Joint Curriculum Task Force. Computing curriculum 1991. Report of the IEEE Computer Society and ACM, 1990.

[2] A. Biermann. *Great Ideas in Computer Science: a gentle introduction*. MIT Press, 1990.

[3] J. G. Brookshear. *Computer Science: an overview*. Addison Wesley, sixth edition, 2000.

[4] P. Davies. Threshold concepts: how can we recognise them? 2003. Paper presented at EARLI conference, Padova. http://www.staffs.ac.uk/schools/business/iepr/docs/etcworkingpaper(1).doc (accessed 25 August 2006).

[5] A. Eckerdal, R. McCartney, J. E. Moström, M. Ratcliffe, K. Sanders, and C. Zander. Putting threshold concepts into context in computer science education. In *ITiCSE-06*, pages 103–107, Bologna, Italy, June 2006.

[6] A. Eckerdal and M. Thuné. Novice Java programmers' conceptions of "object" and "class", and variation theory. In *ITiCSE-05*, pages 89–93, 2005.

[7] Joint Task Force on Computing Curricula. Computing Curriculum 2001, computer science volume. Report of the IEEE Computer Society and ACM, 2001. http://www.sigcse.org/cc2001/ (accessed 25 August 2006).

[8] M. Kölling. The problem of teaching object-oriented programming, part 1: Languages. *Journal of Object-Oriented Programming*, January 1999.

[9] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen. Early programming: A study of the difficulties of novice programmers. In *ITiCSE-05*, 2005.

[10] P. A. Luker. There's more to OOP than syntax. *SIGCSE Bull.*, 26(1):56–60, 1994.

[11] F. Marton and S. Booth. *Learning and Awareness*. Lawrence Erlbaum Ass., Mahwah, NJ, 1997.

[12] R. McCartney and K. Sanders. What are the "threshold concepts" in computer science? In T. Salakoski and T. Mäntylä, editors, *Proceedings of the Koli Calling 2005 Conference on Computer Science Education*, page 185, November, 2005.

[13] J. H. Meyer and R. Land. Threshold concepts and troublesome knowledge (2): Epistemological considerations and a conceptual framework for teaching and learning. *Higher Education*, 49:373–388, 2005.

[14] J. H. F. Meyer and M. Shanahan. The troublesome nature of a threshold concept in economics. 2003. Paper presented at EARLI conference, Padova. (As reported in [13]).

[15] D. Perkins. The many faces of constructivism. *Educational Leadership*, 57(3):6–11, 1999.

[16] E. Roberts. The dream of a common language: The search for simplicity and stability in computer science education. *SIGCSE Bull.*, 36(1):115–119, 2004.

[17] G. M. Schneider and J. L. Gersting. *An Invitation to Computer Science*. Brooks Cole, second edition, 1998.

[18] A. Schwill. Fundamental ideas of computer science. *Bull. European Assoc. for Theoretical Computer Science*, 53:274–295, 1994.

[19] L. Thomas, M. Ratcliffe, and B. Thomasson. Scaffolding with object diagrams in first year programming classes: Some unexpected results. In *SIGCSE-04*, 2004.