2nd International Conference on Higher Education Advances, HEAd'16
Universitat Politècnica de València, València, 2016
DOI: http://dx.doi.org/10.4995/HEAd16.2016.2770

Teaching Abstraction

Böttcher, Axel[a]; Schlierkamp, Kathrin[a]; Thurner, Veronika[a] and Zehetmeier, Daniela[a]
[a]Faculty of Computer Science and Mathematics, Munich University of Applied Science, Germany.

Abstract
Many technical disciplines require abstraction skills, such as the ability to deduce general rules and principles from sets of examples. These skills are the basis for creating solutions that address a whole class of similar problems, rather than merely focusing a single specific instance. Experience shows that many freshmen students are ill equipped with these skills. Therefore, we developed an intervention that systematically teaches abstraction skills to students, and applied our approach to a cohort of freshmen students in computer science.

Keywords: *Abstraction, cognitive competencies, computer science, teaching*

1. Motivation

Thinking in an abstract manner is a key competency for computer scientists (Bucci, Long, & Weide, 2001) (Kramer, 2007), as well as in many other technical discplines. For example, moving from a notion of quantity, count or multiplicity to the representation of numbers in a computer requires some kind of abstraction. Similarly, a function that computes the sum of the values of its two parameters A and B is an abstraction that specifies the general mechanism for adding up two numbers from a given set, no matter what their values are. Finally, when analysts model a complex business process, they usually examine a set of specific scenarios, identify their commonalities and differences, intentionally drop irrelevant detail and mold the relevant parts into some kind of template that captures the very essence of the underlying business process in its "typical" form. Each of these examples involves a strong notion of abstraction.

Our experience, based on both formal tests and observation, shows that abstraction-related competencies are insufficiently developed in the vast majority of our freshmen students. Just to mention one example, the exercise given in Figure 1 was only solved by less than 30% of our freshmen-students. Even worse, most students are not even explicitely aware that there *is* such a skill as abstraction, and that it is highly essential for their chosen course of study. Needless to say, many of them don't know about their own proficiency (or the lack of it) in this area, either.

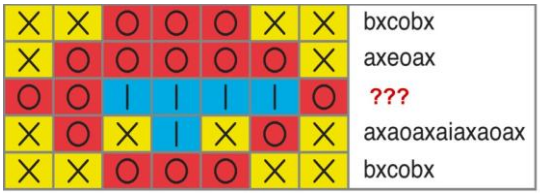

Figure 1. Example question from the 'Informatik Biber', an exercise that assesses abstraction skills of A-level students (BWINF, 2010).

However, abstraction skills are a key competence for computer scientists to be, and highly essential for learning reasonable programming skills. Therefore, in order to enable our students to successfully cope with the technical content that they are confronted with in their course of study, we have to teach them abstraction as a fundamental practical and cognitive competence.

One difficulty that arises in teaching abstraction is, that it is an "invisible" concept, meaning that there is no obvious way to make its notion tangible e.g. by suitable experiments (such as "dropping an apple" to visualize the notion of "gravity"). Another problem is that those people that are adept in the skill of abstraction usually apply it unconsciously. More precisely, experts tend to form abstractions from real world details without being able to specify the steps of the cognitive process that they applied. So

obviously, even if you *are* an expert in abstract thinking, it is difficult to teach this skill to novices if you are unaware of *how* you do it.

Therefore, in order to teach the skill of abstraction in a systematic way, experts first have to identify what they themselves do in the abstraction process. Once this is understood in sufficient detail, they can then move towards developing appropriate didactic approaches that help to evolve the skill of abstraction in their students. In addition, to measure abstraction skills we need suitable tests or tasks that focus on assessing abstraction skills on different levels of expertise.

2. Goals

To tackle these problems, we investigate how we can teach abstraction in a systematic way. Thus, this paper describes and evaluates an intervention for teaching abstraction that

- conveys an understanding of the *concept* of abstraction to our students,
- creates an awareness of what abstraction is, and why it is a necessary skill,
- makes the *cognitive process* of abstraction transparent,
- transfers to our students at least a basic understanding of this process as well as the ability to apply it, and
- helps students to assess their own current skill level in this area.

To specify the intended learning outcome in our students, we have to define which kind of skills we associate with different levels of expertise in abstract thinking. Therefore, in Table 1, we provide teaching goals that are formulated in accordance with the revised Bloom taxonomy for teaching and learning objectives (Anderson, 2001). Note that as our intervention addresses freshmen students and thus novices to the art, it only covers the four lower levels of expertise, i.e. Remember to Analyse.

Table 1. Teaching goals for the intervention on abstraction, according to the revised Bloom taxonomy.

Level	Teaching Goal: Students ...
Remember	… define the terms abstraction and concretion. … define the competence of abstract thinking.
Understand	… explain that computer science mainly deals with abstract concepts and that hence the ability of abstract thinking is essential in this domain. … reason that understanding of a domain requires an understanding of the underlying rules.
Apply	… derive concrete statements from a given simple abstraction. … extract the simple rule-set underlying a given set of concrete yet also simple examples.

Analyse	… apply a meta-strategy for finding an abstraction and solution strategy for a given abstraction task.
Evaluate	… evaluate the results of an abstraction process (i.e. the model or rule-set).
Create	… develop a meta model.

3. Related Work

Abstraction is "a process of omitting all individuating features, and retaining only what is common to all of a set of resembling particulars"
John Locke, to be found e.g. in (Wiener, 1973-1974)

Contributions towards abstraction in computer science (CS) curricula touch three categories:

- definition attempts or at least identification of abstraction in computer science
- the field of teaching abstraction abilities
- the field of testing them

Kramer states that abstraction is a key skill for computing (Kramer, 2007). He concludes that "we should focus more directly on ensuring that our teaching is effective and that computing professionals have adequate abstraction skills". As a basis, he recommends to measure students' abstraction abilities, both at the time when they apply for a place at colleges to study computer science, as well as annually throughout their college education process. But he does not give hints on how to achieve this. Kramer also mentions that many courses "rely on or utilize abstraction (...) but that it must be taught indirectly through other topics". Similarly, Bucci et al. (Bucci, Long, & Weide, 2001) observed that abstraction is severely shortchanged by current CS1/CS2 pedagogy. They give some examples of teaching support.

On the other hand, Hazzah and Kramer (Hazzan & Kramer, 2007) state that "abstraction should be introduced as an identifiable concept". This corresponds to our notion, namely to move away from the indirect teaching of abstraction to focusing it as a topic by itself. If teachers are not aware of their own abstract thinking, they might easily overlook concepts that need a detailed explanation, and take it for granted that students can understand these concepts on their own. However, we did not find any contributions towards teaching abstraction as a concept on its own, and making the abstraction process transparent to students.

Cook et al. have tried a systematic approach to teach abstraction for computer scientists to be, which is based on mathematical modeling. When basic math courses are taught in

parallel to introductory courses on software development and not up-front, an additional layer of complexity is introduced by the context of that specific mathematical subject.

Several ideas for tests that measure abstract thinking capabilities are proposed by Hazzan & Kramer (Hazzan & Kramer, 2007). However, they already require some basic knowledge of computer science which we cannot expect in our freshmen students.

4. Teaching Approach

Our teaching unit on abstraction is designed along the process depicted in Figure 2, based on Dietz & Dietz (Dietz & Dietz, 2011). Starting point is the idea to make students aware of their lack in abstract thinking. As soon as students are conscious of their incompetence and willing to face this deficit, it is possible to work with them to close the identified gap. The next step is to teach abstraction and to develop basic skills in this competence. Those skills must be practiced over and over, and improved until abstract thinking finally becomes an unconscious competence. Abstract thinking must become second nature to our students.

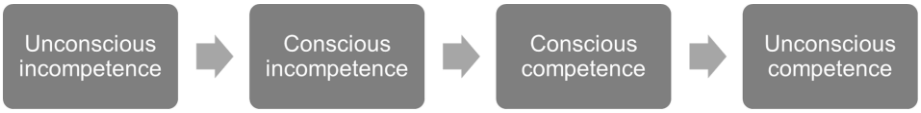

Figure 2. The different steps from unconsciuos incompetence to unconscious competence.

4.1 From unconscious incompetence to conscious incompetence

The first of two 90-minute teaching units started with an initial test to unveil the deficits in abstract thinking of our students, and to attract the students' attention for this topics.

4.2 From conscious incompetence to conscious competence

On the way to conscious abstract thinking, we tried to make transparent categorizations that unconsciously take place in everyday's life. As examples, we used verbs/nouns, round/angular shaped geometric ojects and car/bike brands, and asked our students to find the odd ones out. This demonstrates that everyone naturally uses fundamental abstraction skills. Next step was the discussion of the term "abstraction" and its common definitions. This was followed by exercises to find commonalities in sets of different entities like video game consoles, fruit, or geometric objects. Those commonalities define rule sets, which in general are the basis for the transformation into formalism of software.

On the other hand, it is important to be able to identify concrete examples from given sets of rules. Therefore, we asked students to find examples for the following three rule sets:

- primitive data types in Java
- control structures
- mandatory classes and electives in their degree course

Furthermore, in order to establish the relationship to the context of software development, we introduced the concepts of entities and behavior, which result in *boxes* (classes) and *blue prints* (class definitions).

In a second step, we introduced the concept of the abstraction of processes. As examples from everyday life, we used text formatting in a word processor, and sorting. As an exercise for the *abstraction* of processes, students had to define rules for calculating change in a vending machine. To practise *concretion* of processes, students had to apply the right-hand-rule for escaping from different mazes.

Deepen the conscious competence

A deep understanding of abstraction in the domain of computer science requires expert thinking. In order to understand the mental processes computer scientists implicitly apply when solving problems, we analyzed our thinking-steps while developing an algorithm with help of a think-aloud session of domain experts (Pea, 1986). Following the approach of Pólya (Pólya, 1973), we formalized the single steps we applied into a general approach. In order to give our students more guidance than the very general description of Pólya, we described our approach in more detail with respect to finding algorithms.

We used this approach to develop the second part of our intervention on abstract thinking, a 90 minute lecture where we focused on the problem of determining whether a given word is a palindrome (word that reads identically backward or forward) or not.

At the beginning, we introduced the task description including a short definition of palindromes. In addition, we defined the input and output of the desired algorithm. According to our approach, the first step is to find examples and non-examples. Subsequently, students were asked to informally describe an algorithm in natural language. This typically includes redundancy in terms of repeated compare-instructions on different character positions, as shown in Figure 3.

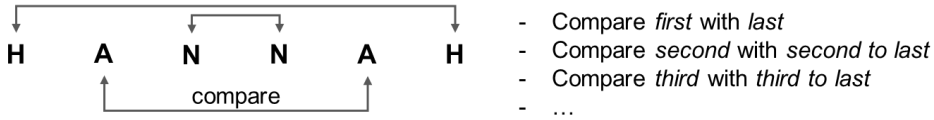

Figure 3. Illustration of the compare-instructions on different character positions of a palindrome.

The next step is to apply the following fundamental rules of formalization:

- Replace the textual sequence with an index
- Create decision trees and flow diagrams
- Use signal words like *if, while, until, repeat, …* for the process description
- Avoid redundancy wherever possible

This description is refined step by step, until the majority of our students understands how to transfer the process description into source code. In order to support students while formulating source code, we listed some general rules to help them:

- Wrap an algorithm into a method
- Map signal words to constructs of the programming language
- Convert examples to test-cases

5. First Results

In order to evaluate our special lecture, we designed a pre- and a post-test. For these tests, we used two similar questions as shown in Figure 4, as well as one question that occurred identically in both tests. Furthermore, we asked students to self-assess their competencies in concrete and abstract thinking on a scale of 1 (low) to 10 (very high).

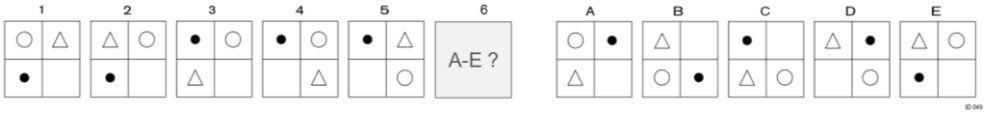

Figure 4. Task: how would the series on the left side continue?

The first test was handed out at the beginning of the first lecture, and students had 10 minutes to fill in the form. Then, we started with the lecture that took around 60 minutes. At the end of the lecture, we assessed students again to evaluate the impact of our lecture.

As students performed good in the pre-test on the questions (15 and 19 correct answers out of 21 participants) there was no significant improvement to measure. It seems that students have some basic abilities of abstract thinking at the beginning of their studies. However, this just works for common problems like the one illustrated in Figure 4. Nevertheless, if it comes to computer science related tasks like finding an algorithm (also cf. the exercise in Figure 1 for this issue), students often struggle. This is one reason, why we prepared the second part of the intervention, where we have explicitly presented the thinking process for developing an algorithm step by step.

The self-assessment of the students' confidence in their ability to think in an abstract way increased from a median of 5 to 6. At the beginning of their studies (October 2015), the same students estimated this ability with 3 to 4. Thus, this single lecture improved the ability of thinking abstractly much quicker than other lectures before.

To ask our students for feedback on the lecture, we distributed another questionnaire. Students stated that the unit was helpful to understand what abstraction and concretion is about, and that the intervention provided an introduction into the topic. Additionally, students realized that these skills are important for understanding problems from the area of computer science. As well, students stated that they felt supported individually, and that

they learned something new. However, they still felt not very confident about their individual competencies. For example, they stated that they are not sure if they are able to create abstract objects, relationships and rules by themselves for new tasks, or if they can depict complex facts clearly. All in all, they assessed themselves to be more skilled in concrete than in abstract thinking.

6. Conclusion and Future Work

We devised an intervention for teaching abstraction systematically, and tested it with our students of computer sciences. As a result, our students stated that they increased their knowledge about abstract and concrete thinking, but still struggle when solving new tasks. Therefore, they need to practice these skills regularly, to increase their proficiency in this area. Only with sufficient practice, our students' abstraction skills will evolve from a conscious competence to an unconscious one.

References

Anderson, L. W. (2001). *A Taxonomy for Learning, Teaching, and Assessing. A Revision of Bloom's Taxonomy of Educational Objectives* (1 ed.). (L. W. Anderson, D. R. Krathwohl, P. W. Airasian, K. A. Cruikshank, R. E. Mayer, P. R. Pintrich, . . . M. C. Wittrock, Eds.) New York: Longman.

Bucci, P., Long, T. J., & Weide, B. W. (2001). Do we really teach abstraction? *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education* (pp. 26-30). New York, NY, USA: ACM.

BWINF, G. f. (2010). Informatik Biber. *Informatik Biber*. Retrieved from http://informatik-biber.de/

Dietz, I., & Dietz, T. (2011). *Selbst in Führung.* Paderborn: Junfermann Verlag.

Hazzan, O., & Kramer, J. (2007, January). Abstraction in Computer Science & Software Engineering: a Pedagogical Perspective. *Frontier Journal, 4*(1), 6-14.

Kramer, J. (2007). Is abstraction the key to computing. *Communications of the ACM, 50*.

Pea, R. D. (1986). Language-independent conceptual 'bugs' in novice programming. *Journal of Educational Computing Research, 2*, 25-36.

Pólya, G. (1973). *How to Solve It: A New Aspect of Mathematical Method.* Princeton university press.

Wiener, P. P. (Ed.). (1973-1974). *The Dictionary of the History of Ideas: Studies of Selected Pivotal Ideas.* New York: Charles Scribner.