5th International Conference on Higher Education Advances (HEAd'19)
Universitat Politècnica de València, València, 2019
DOI: http://dx.doi.org/10.4995/HEAd19.2019.9393

About Students' Abstractions
Evaluation of Items Requiring Abstract Thinking Competence

Daniela Zehetmeier
Department of Computer Science and Mathematics, Munich University of Applied Sciences and Department of Informatics, Technical University of Munich, Germany

Abstract
Abstract thinking is one of the most important competences in computer science. When starting my research, there was no complete definition of the competence nor was there a tool to assess first-semester students' competence level. Thus, I developed a competence model of abstract thinking, which allowed me to derive an assessment tool. In this work, I will present first insights gained by analyzing the tests of 134 incoming students of computer science and scientific computing. The analysis confirms the assumption that incoming students often lack in this essential competence. Moreover, the overemphasis of the data aspect of classes in object oriented programming can be confirmed for university level education. Further investigations will follow. In the future, the insights gained can be used to develop teaching units or whole teaching concepts.

Keywords: *Abstract Thinking; Assessment; Evaluation; Introductory Programming; Novice Programming; Computing Education.*

1. Introduction

Literature and teaching experiences indicate that the competences of abstract, logical and analytical thinking are highly important in computer science (Dörge, 2012, Computer Engineering Curricula 2016, 2015, Society, Bourque, & Fairley, 2014). They are an essential prerequisite to acquire computing competences. Nevertheless, lecturers often observe a lack in these competences among first-semester students (Kramer, 2007, Thurner, Böttcher, & Kämper, 2014). Besides logical and analytical thinking, abstract thinking is less researched.

When looking at publications regarding the measurement of abstract thinking, like Kurtz (Kurtz, 1980) or Or-Bach & Lavy (Or-Bach & Lavy, 2004), it is noticeable that there is no definition stated the tool is based on, the tool is often not explicitly presented or the sample sizes are very small. And many times, the methods applied are not appropriate to be applied in first-semseter classes were the lack already exists, as they require experts knowledge concerning software development. When used at the beginning of students studies, I would mainly measure their professional knowledge. An approach that is appropriate for my intended setting is described by Bennedsen & Caspersen (Bennedsen & Caspersen, 2006). However, their study led to unexpected results and needs further revision. Thus, there is still a research gap. Consequently, an assessment tool called *Abstract Thinking Assessemnt (ATA)* has been developed, based on a well-research competence definition. The target group of the assessment is the population first-semester students in computer science or related topics, and thus, does not require any computer science specific knowledge.

2. Fundamentals

In order to interprete the data, it is crucial to know the definition underlying all problems (called items) in the ATA. All items are based on the competence model described in (Zehetmeier, Böttcher, Brüggemann-Klein, & Thurner, Defining the Competence of Abstract Thinking and Evaluating CS-Students' Level of Abstraction, 2019). It consists of the following three components:

- Identify commonalities in order to summarize them and to determine differences to normalize them, e.g. by parametrisation.
- Decide which information is essential for the given purpose and which is not.
- Create theoretical relationships between items or processes.

Additionally, it is important to know how students' answers to the 24 open-ended questions have been interpreted. This was done using a coding manual. It describes two independent perpectives: *correctness* and *level of abstraction*. Hence, two codes are assigned to each answer. The coding manual used is shown in Table 1. More details regarding the development and application of the coding manual can be found in (Zehetmeier, Böttcher, Brüggemann-Klein, & Thurner, 2019).

Table 1: Coding manual for dimension correctness and level of abstraction
(Zehetmeier, Böttcher, Brüggemann-Klein, & Thurner, Defining the Competence of Abstract Thinking and Evaluating CS-Students' Level of Abstraction, 2019)**.**

	Category	**Description**	**Score**
Correctness	Empty	Empty response field	0
	Regardless	The answer is without regard to the question. Buzzword	0
	False	An answer is false, if it is deficient or partly deficient from a professional perspective.	0
	Correct	An answer is correct, if it is accurate and complete from a professional perspective.	1
Level of Abstraction	Empty	Empty response field	0
	Regardless	The answer is without regard to the question. Buzzword	0
	Concrete	The answer describes the given examples using everyday or well-known terms.	1
	Specific	The answer depicts a rule or a rule set, which can only be applied to the given examples.	2
	Generic	The answer depicts a rule or a rule set, which can be applied to the given examples and beyond that.	3

3. Analysis

Data collection took place in winter semester 2018/19 right after the beginning of students' studies using the ATA. The population that forms the basis for these analysis are 134 first-year students in the bachelor programs of computer science and scientific computing at the Munich University of applied sciences. All assessments have been coded by me. Based on these data, known hypothesis and misconception regarding abstract thinking are investigated.

3.1 Deficites in the Competence of Abstract Thinking

Thurner et al. (Thurner, Böttcher, & Kämper, 2014) report a deficit in the competence of abstract thinking among the first-year students in computer science or related topics. So far, no tool existed to collect data and to verify this hypothesis. With help of the newly developed ATA and the data collected, the hypothesis can now be verified.

For the evaluation, the *Levels of Abstraction* have been transformed into dichotomous scores 0 and 1. Students achieved a point, if their answer is at least on a specific level. Level specific describes answers containing rules or rule sets, but can only be applied to the given examples. Answers are valued 0 if they describe an abstraction by using unspecific terms, describing the given examples or express actions step by step.

Figure 1 depicts an exploratory analysis of students' competence of abstract thinking. The x-axis depicts the percentage threshold representing the percentage of answers that needs to show abstract thinking competence. For each threshold it is evaluated how many of the students exeed the threshold. Consequently, every entry in the heatmap represents the the percentage of students exceeding the threshold.

According to commonly chosen 50% threshold, 73% of the students would exeed the threshold. This indicates a rather small deficit among the cohort. However, this analysis solely focusses on the level of abstraction and not on the correctness of the abstraction build. By including this facet the picture is a different one (see Figure 2).

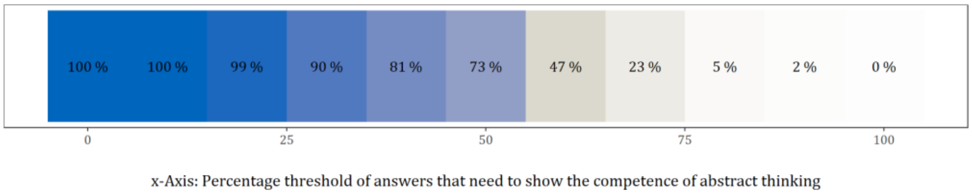

Figure 1: Development of student populations' success rate, if the threshold of answers that need to show the competence of abstract thinking is increased continuously from 0% to 100%.

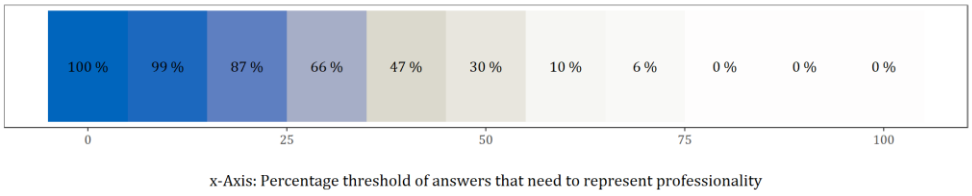

Figure 2: Development of student populations' success rate, if the threshold of answers that need to be correct and that show the competence of abstract thinking is increased continuously from 0% to 100%.

When again using the 50% threshold, only 30% of the students are able to exeed the threshold. This analysis reveals a major deficit in students' initial competence. This might be one explanation for the failure rate in the end-of-term exam, which could lead to the high drop out most computer science programs are facing with.

3.2 Misconception of Overemphasising Data Aspect of Classes

Besides hypotheses, there are known misconceptions in literature that are interesting for educational research in computer science. One that is related to abstract thinking competence is that students at school often overemphasize the data aspect of objects (Humbert, 2006). For university education it is interesting whether students at university still have this misconception as this influences how object-oriented programming is introduced.

The ATA contains two questions that allow insights into the type of characteristics students use intuitively to describe collections of similar objects. Derived from the competence model students need to describe commonalities and differences among objects and processes. For static artefacts, questions focus on summarising depictions under one umbrella term or on naming common details. Either way, students need to identify common and different characteristics.

A typical task in computer science that requires this part of the competence is the development of classes. Programmers need to find an appropriate name for the class representing several objects. Moreover, each class containes attributes and methods describing common characteristics.

Without any computer science specific knowledge students cannot write proper classes in a specific programming language, but they are able to find an umbrella term or list common characteristics of given objects. The ATA asks students to name characteristics of several instances of (1) lego bricks and (2) vehicles (two items). Their answers were categorised into five categories:

Table 2: Coding manual for answers concening the 'Vehicle-Item'.

Category	**Example(s)**
Missing / Regardless	*Empty*, “I don’t know”, Car
Component	Tires, Windows, Seats
Attribute	Colour, Weight, HP
Behaviour	Drives, consume fuel
Others	Number of Tires, Car type, Carries people

Most of these categories can be mapped to a specific programming construct, e.g. components are translated into association and behaviour is represented by methods. With this analysis I want to evaluate, which type or construct students spontaneously use to describe object. As depicted in Figure 3 students most often choose components or attributes as characteristics. This indicates that the misconception of overemphasizing the data storage aspect of objects (Humbert, 2006) is also present in the student puopulation. Consequently, students do not focus on the dynamic aspect of objects intuitively.

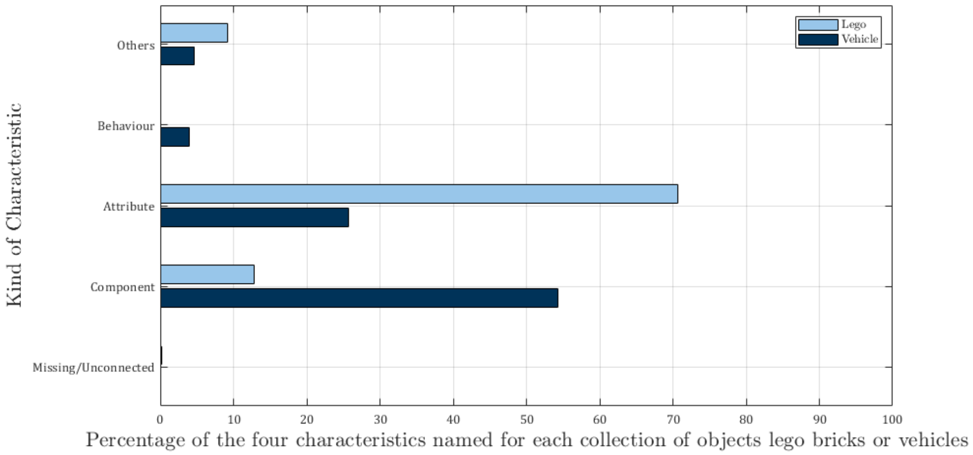

Figure 3:Percentage of the type of characteristic students used to describe several.

4. Conclusion and Future Work

The data reveals that students in computer science are not well prepared regarding their competence of abstract thinking. They are either able to describe "abstractions" (not necessarily on a high level of abstraction) correctly or they are able to specify an abstraction on a high level of abstraction, but building a correct abstraction on a high level seems challenging for them. However, such abstractions form the basis for the subsequent implementation in a specific programming language.

Thus, I agree with Or-Bach and Lavy (Or-Bach & Lavy, 2004) to discuss both, modelling and implementation in the lecture. I would suggest to put more emphasis on the modelling at the beginning of a CS-1 class, as the model is the basis for the implementation. Lectures need to put more emphasize on teaching abstract thinking competence and the processes behind, since this is one of the fundamental mental processes in the modelling phase. A useful technique to teach mental processes is cognitive apprenticeship (Collins, Brown, & Holum, 1991).

The analysis of the data also revealed that students intuitively use attributes and components to describe characteristics of similar objects. They rarely name behavioural characteristics. This finding should influence the introduction to object-oriented programming. Commonly, attributes and methods are introduced early and quickly, whereas the topic components occur later in the curriculum and often with extensive explanations. Due to the findings, components could be taught much earlier in the curriculum as students are familiar with the concept. Lecturers should put more emphasis on teaching methods and spend more time on this topic.

Acknowledgement

This work was supported by the German Federal Ministry of Education and Research (BMBF), grant no. 01PL16025, as part of the “Qualit¨atspakt Lehre” (“Teaching Quality Initiative”) program. Thank you for your support.

References

Bennedsen, J., & Caspersen, M. E. (2006, 6). Abstraction Ability As an Indicator of Success for Learning Object-oriented Programming? *SIGCSE Bull., 38*, 39-43. doi:10.1145/1138403.1138430

Collins, A., Brown, J. S., & Holum, A. (1991). Cognitive apprenticeship: Making thinking visible. *American educator, 15*, 6-11.

Humbert, L. (2006). Informatische Bildung: Fehlvorstellungen und Standards. *MWS -- Münsteraner Workshop zur Schulinformatik 2006*, (pp. 37-46).

Kurtz, B. L. (1980). Investigating the relationship between the development of abstract reasoning and performance in an introductory programming class. *ACM SIGCSE Bulletin, 12*, pp. 110-117.

Or-Bach, R., & Lavy, I. (2004, 6). Cognitive Activities of Abstraction in Object Orientation: An Empirical Study. *SIGCSE Bull., 36*, 82-86. doi:10.1145/1024338.1024378

Thurner, V., Böttcher, A., & Kämper, A. (2014, 4). Identifying base competencies as prerequisites for software engineering education. *Global Engineering Education Conference (EDUCON), 2014 IEEE*, (pp. 1069-1076). doi:10.1109/EDUCON.2014.6826240

Zehetmeier, D., Böttcher, A., Brüggemann-Klein, A., & Thurner, V. (2019). Defining the Competence of Abstract Thinking and Evaluating CS-Students' Level of Abstraction. *Conference on Software Engineering Education and Training (CSEE&T).*