Defining the Competence of Abstract Thinking and Evaluating CS-Students' Level of Abstraction

Daniela Zehetmeier*
Munich University
of Applied Sciences
Lothstraße 64
D-80335 München
daniela.zehetmeier@hm.edu

Axel Böttcher
Munich University
of Applied Sciences
Lothstraße 64
D-80335 München
axel.boettcher@hm.edu

Anne Brüggemann-Klein
Technical University
of Munich
Boltzmannstraße 3
D-85748 Garching
brueggemann-klein@tum.de

Veronika Thurner
Munich University
of Applied Sciences
Lothstraße 64
D-80335 München
veronika.thurner@hm.edu

Abstract

Although it is commonly agreed that the competence of abstraction and abstract thinking is one of the most important competences in Computer Science, only a few of these sources define this competence and its processes in a precise manner. Furthermore there is a lack of instruments to test the competence of abstract thinking and to integrate it into teaching.

This work will start to close the gap concerning the competence of abstract thinking by deriving a theoretical description of the competence construct of abstract thinking, focusing on a Computer Science perspective. Furthermore, we will present a coding manual based on the model, which can be used to evaluate student assignments.

This coding manual is applied to examples of our teaching practice in order to demonstrate its validity.

1. Introduction and Overview

It is commonly agreed that the competence of abstraction and abstract thinking is one of the most important competences in computer science [1, 2]. This is because numerous concepts in computer science are abstract by nature [3, 4].

Associations, like IEEE, ACM and the German GI mention in their curricula for computer science that abstraction is a basic methodology and a key component. The US Advanced Placement Courses name abstraction as a central problem-solving technique and so does the Guide to the Software Engineering Body of Knowledge (SWEBoK [5]). Textbooks for teaching computer science refer to abstraction as a fundamental concept.

The ability to abstract is stated as one of the 12 most important competences of Software Engineers investigated by [6]. Keith Devlin [7] writes: "Once you realize that computing is all about constructing, manipulating, and reasoning about abstractions, it becomes clear that an important prerequisite for writing (good) computer programs is the ability to handle abstractions in a precise manner."

Previous investigations suggest that students lack in competences especially abstract thinking, which are necessary to study computer science or related topics successfully [8, 9]. Hence, it must be the goal of computer science education at universities, to develop abstract thinking within their students at an early stage.

Although there exist many scientific sources, which state the competence of abstract thinking as one of the most important competences in computer science, as mentioned above, only a few of these sources define this competence and its processes in a precise manner. Moreover, existing attempts to define it differ significantly from each other or highlight solely a single aspect of the competence.

In summary, there is a research gap concerning the competence of abstract thinking in computer science. Although the relevance of abstract thinking is commonly agreed, there is no consistent and complete definition of that competence. Nor has it been proven that the competence is a key for acquiring computer science skills and thus its relevance in the curriculum of computer science [10, 11].

As Kramer [8] states: "before we can control or effect, we must first measure" there exist no concepts for teaching the competence as there is no description and no possibility to measure the competence of abstract thinking. Only a sound model allows the development of a valid test instrument [12], which is missing as well. Consequently, statements about the competence of abstract thinking and its influence on the success in studying computer science remain hypotheses.

Our overall research goal is to develop a valid

*This author is also with Technical University of Munich, Boltzmannstraße 3, D-85748 Garching

URI: https://hdl.handle.net/10125/60202
ISBN: 978-0-9981331-2-6

instrument to assess the competence of abstract thinking that enables us to improve teaching concepts as well as to adopt these to students' needs.

This work describes the first step to achieve this goal by deriving a theoretical model of the competence of abstract thinking, focusing on a computer science perspective. Furthermore, we will present a coding manual based on the model, which we will use to evaluate existing student assignments. This is a pilot effort to establish the basis for an empirical study.

2. Literature Review

In a first step we conducted a literature review, which was guided by the recent work of Hazzan & Kramer [13] as well as Cetin & Dubinsky [14]. Additionally, we looked into English dictionaries and encyclopedias for the terms: abstraction, abstract, and abstracting. The goal was to find definitions for the competence of abstract thinking. Table 1 summarizes our findings by quoting relevant statements that can be seen as definitions or describe a process.

Besides that, there exist publications like [15, 16, 17] that state the relevance of abstract thinking, try to measure the competence or teach the competence, but do not clearly state a definition nor competence model. One example that gives a unclear definition is: *"In mathematics learning, the term abstraction is used in two senses: An abstraction is a mental representation of a mathematical object. Abstraction, without an article, is the mental process by which an individual constructs such an abstraction"*. [15, p. 31]

3. Conceptualization of Abstract Thinking

In order to measure and teach the competence of abstract thinking it is necessary to establish a common understanding of what happens during the process of abstract thinking. The goal of the conceptualization is to deduce a competence model based on the literature review.

In a first step we analyzed the quotes listed in the literature review above. Thereby we focused on the processes described. Often it was sufficient to look at the verbs used. We clustered similar process descriptions and ended up with four clusters.

The first group contains processes describing a summary of common features, as well as processes that include steps to identify and to normalize distinguishing properties. We labeled this cluster with *Commonalities and Differences*. Processes involving removal or reduction are comprised in the cluster called *Remove*. Our third cluster is entitled *Keep*. It assembles processes that mention focusing on relevant information, described by terms like essential and concentrate. The last one covers higher level processes leading to new structures or ideas (*Expand*).

We were almost satisfied with these four clusters. However, when solving problems that require a focus on essential information, *Remove* and *Keep* appeared to be two sides of the same medal. Merging them into one cluster illustrates that using certain details is a well-considered choice. Furthermore, we agree to Colburn & Shute [36], who describe the process of removing as hiding details, but not neglecting them, as they might be important in a lower level of abstraction. Thus, we suggest the term *Hide* instead of *Remove*, as we interpret the term of removing as an irreversible process. Consequently, we end up with three clusters by combining those two. The resulting second cluster is finally called *Hide & Keep*.

Fig. 1 shows a Venn diagram where the references are assigned to the clusters they address. Some statements are related to more than one cluster. Thus, they appear in the appropriate intersecting set.

The identified clusters build the three components of our model of abstract thinking. In summary the components are defined as follows (cf. Fig. 2):

- *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.*

Another aspect we think is relevant with regard to the competence of abstract thinking is the kind of artifacts, which are considered during the abstraction process. We distinguish between static entities and dynamic processes while abstracting [26]. In our opinion this facet will influence teaching, especially in Computer Science.

In the following subsections we illustrate the components of our proposed model and their relevant literature in more detail. Additionally, we give CS-related examples.

3.1. Commonalities and Differences

One process that is often associated with abstraction is identifying *commonalities and differences*. Weicker et al. [30] describe the ability to abstract as the skill to identify commonalities and to normalise

Table 1. Summary of our informal literature review presenting definitions and process descriptions.

Quote	Reference
"The process of considering something independently of its associations or attributes", "Consider something theoretically or separately from (something else)", "Extract or remove (something)"	[18]
"disassociated from any specific instance", "relating to or involving general ideas or qualities rather than specific people, objects, or actions", "[...] something that summarizes or concentrates the essentials of a larger thing or several things", "to consider apart from application to or association with a particular instance", "the act of obtaining or removing something from a source"	[19]
"one is able to draw conclusions or illustrate relationships among concepts in a manner beyond what is obvious. [..] Progressing beyond the tangible characteristics in order to conceptualize theoretical relationships between items or processes. [...] Abstract thinking occurs conceptually, categorically, and generally."	[20]
"drawing out of common features [...] represent the essential underlying relationships and the irrelevant aspects of the problem are ignored. "	[15, p. 31]
"a process of omitting all individuating features, and retaining only what is common to all of a set of resembling particulars" ([21] cited from [22])	[21]
Extraction of commonalities from several specific instances and corresponding categorisation.	[23]
"[a]bstraction is the transition from concrete to abstract, that is, to the set of commonalties.[...] An activity of vertically reorganizing previously constructed mathematics into a new mathematical structure.", "abstraction proceeds from a set of mathematical objects (or processes) and consists of focusing on some distinguishing properties and relationships of these objects rather than on the objects themselves. The product of abstraction consists of the class of all objects that have the distinguishing properties and enter into the distinguishing relationships"	[24]
"abstractions are constructed by assembling available ideas into new structures. The function of abstraction is not to provide generality but to facilitate the assembly process and to provide a different categorization. [...] particulars are recognized as instances of the same abstraction"	[25]
Can appear in two shapes: idealisation (process) and extraction (product, result)	[26]
"By abstracting, man isolates and, in the process of ascent, mentally retains the specific nature of the real relationship of things that determines the formation and integrity of assorted phenomena. [...] 'The abstract' usually has several characteristics - it is something simple, devoid of differences, fragmentary, and undeveloped."	[27]
"The act or process of leaving out of consideration one or more properties of a complex object so as to attend to others"	[28, p. 11]
Reduction of complexity or formalisation	[3, p. 427]
"Through abstraction we view the problem and its possible solution paths from a higher level of conceptual understanding. As a result, we may become better prepared to recognize possible relationships between different aspects of the problem and thereby generate more creative design solutions."	[29, p. 240]
The ability to abstract is the skill to detach circumstances from specific details and to focus on on the essential. Furthermore, it is the skill to identify commonalities and normalise differences. Additionally, the ability to think and work without a concrete reference is described.	[30]
"[...] process and result of generalization by reducing the information of a concept, a problem, or an observable phenomenon so that one can focus on the 'big picture' "	[31, p. 13-4]
"process of removing details to simplify and focus attention"	[8]
Common characteristics are carved out or an amount of events with common features is summarised.	[32, p. 212 ff]
Is the approach where the real world is reduced to a model, which highlights the substantial parts or were certain aspects of an issue are deliberately omitted	[4, p. 115].
"It is a cognitive means according to which, in order to overcome complexity at a specific stage of a problem solving situation, we concentrate on the essential features of our subject of thought, and ignore irrelevant details. [...] it enables the problem solver to think in terms of conceptual ideas rather than in terms of their details."	[33]
"a way to do decomposition productively by changing the level of detail to be considered. [...] The process of abstraction can be seen as an application of many-to-one mapping. It allows us to forget information and consequently to treat things that are different as if they were the same. We do this in the hope of simplifying our analysis by separating attributes that are relevant from those that are not."	[34, p. 11]
"Abstraction reduces information and detail to facilitate focus on relevant concepts. [...] It is a process, a strategy, and the result of reducing detail to focus on concepts relevant to understanding and solving problems. [...] The process of developing an abstraction involves removing detail and generalizing functionality. [...] An abstraction extracts common features from specific examples in order to generalize concepts"	[35, p. 15 f]
´´The most common, but not necessarily the most important meaning of abstraction of a concept in computer science and mathematics, is extraction, that is, the idea of considering common features of several (the more the merrier) examples and building a structure or category which has all of these features."	[14]

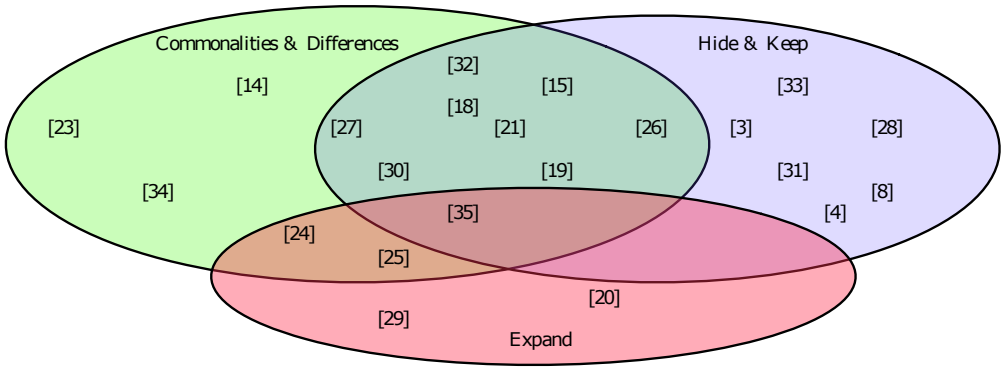

Figure 1. Venn diagram of references assigned to the clusters they address.

differences. Normalising differences is an aspect typical for computer science. This component of abstraction can also be found in the work of Liskov & Guttag [34] as abstraction by parameterization. In more detail this process is described as treating "things that are different as if they were the same".

This component finds further underpinning in: [22], [23], [15, p. 31] or [19].

A typical example for finding and summarizing commonalities or normalizing differences is to parameterize various similar entities in order to create them in a repeating concept. More concrete one would like to draw several equal objects, but spreading them along the x-axis. Thus, the x-coordinate of the object is a parameter and varies in every repetition.

3.2. Hide and Keep

Another aspect many definitions state includes the process of removing information to focus on the essentials [8, 18, 19, 28, 30, 35]. When revealing the essential features [8, 31, 34] it is important that this happens for a specific purpose.

This arises while defining classes in object-oriented programming based on given entities. Based on the purpose the resulting objects can look different.

3.3. Expand

A process that appears while abstract thinking is the assembling of available ideas into new and more abstract structures [25]. These kind of new structure appears to be easier to understand, but is increasingly difficult to develop [34].

The development of high-level programming languages is the more complex the higher the level of abstraction is. Nevertheless, high-level programming languages are more intuitive, easier to read and learn as they hide a lot of complex single steps.

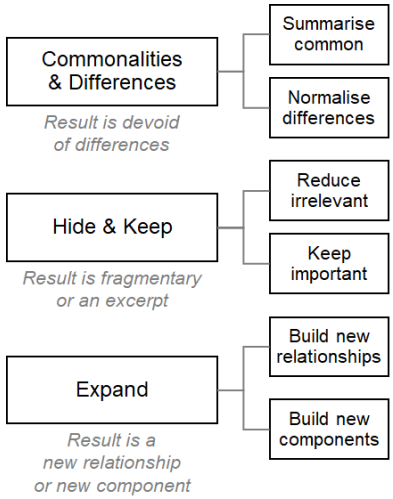

Figure 2. Resulting model after the clustering process stating the identified aspects of abstract thinking.

4. Rating Rationale

For each task used to measure the competence of abstract thinking a rating rationale is required. We developed one that guides evaluating staff through the qualitative rating process and helps to produce unbiased assessments.

In this section of the paper we present the development of a generic rating rationale that can be used for the evaluation of all kinds of tasks intended to measure the competence of abstraction. The results of the manual's application to a couple of tasks that can be used to evaluate students' abilities to abstract, are depicted in section 6.

We identified two perspectives to be considered, namely *correctness* and the *level of abstraction* a student proves to have achieved.

For both perspectives we have the rating category

empty for an empty answer or answers that are crossed out. The category *regardless* is used for answers that do not show any relation to the problem statement. A detailed description of the rating rationale can be found in Table 3 and Table 4.

Additional categories specific for the first evaluation dimension of *correctness* are *correct* and *false* (cf. Tab. 3). There must be a hard border between these two categories. Therefore we rate only completely correct answers as correct, which means that the answer is accurate and complete in a professional manner. Partly correct or wrong answers are rated as false.

The rating rationale for the *level of abstraction* is guided by the observation that people who think in an abstract manner are able to summarize or concentrate on the essentials of several objects or processes. Furthermore, abstract thinking is commonly seen to appear on various levels [13, 14, 29, 31, 34]. Thus, it seemed obvious that we have to find an adequate measure for the level of abstraction a student achieved.

Based on the literature, two opposed categories *concrete* and *generic* can definitely be defined [19, 24, 25]. Our teaching experience shows that students tend to only abstract given examples, which is more than concrete, but not the most generic solution. Thus, we introduced a third category *specific* between the poles *concrete* and *generic*.

Concrete answers are characterised by a practical and stepwise explanation. Furthermore, answers focus on the given examples and use well-known terms. When answers reveal a rule set, which can only be applied to the given examples, but is complete, the answer is *specific*. In the case that the described rule set can be applied beyond the given examples, the answers is rated as *generic*.

5. The Maze-Task

When we want to assess the degree to which our students have developed a certain competence – abstraction in our case – we must find ways to test it. This means that we have to find exercises whose solution requires mainly abstract thinking and which are mostly limited to abstract thinking in the ideal case. Of course no exercise can be limited to a single competence, as basic competences like reading and accuracy are required anyway.

As an example for an assessment of abstraction-related abilities we use an exercise from our first-semester course on software development in the Computer Science Bachelor curriculum. This course is similar to CS1.

The assessment is based on the implementation of a maze-generating software using the hunt-and-kill-algorithm described in [37]. The maze-generator "moves" on a grid of cells. Each cell has a right and a bottom wall (cf. Fig. 3). A maze is generated by erasing walls according to given rules.

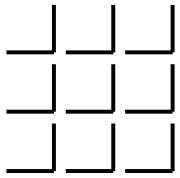

Figure 3. Field of cells.

Jamis Buck describes the algorithm as follows:

1. Choose a starting location.
2. Perform a random walk, carving passages to unvisited neighbors, until the current cell has no unvisited neighbors.
3. Enter "hunt" mode, where you scan the grid looking for an unvisited cell that is adjacent to a visited cell. If found, carve a passage between the two and let the formerly unvisited cell be the new starting location.
4. Repeat steps 2 and 3 until the hunt mode scans the entire grid and finds no unvisited cells.

A real benefit of this algorithm is that it does not require the concept of backtracking, like many others.

We wanted to put emphasis on the software design phase right from the first semester and we wanted to make expert thinking processes transparent. Based on positive experiences with using guiding questions representing decision points of experts' thinking in assignments for heterogeneous groups we wanted to apply this approach to algorithm development and object oriented design.

However, students are not expected to develop complex designs on their own yet. Thus, we tried to guide them through a thinking process for solving the given problem. This should help students develop their idea of the concept by following the guiding questions. After every design step, they can compare their result with our ideas.

5.1. Students' Prerequisites

This algorithm can be implemented based on two-dimensional arrays and is reasonably complex – which made it an ideal candidate for the point of time in that course. Jamis Buck [37] describes this algorithm

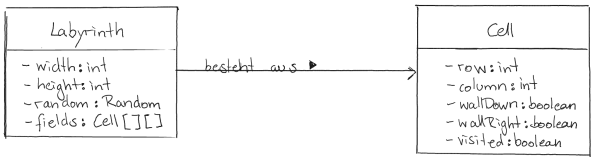

Figure 4. Intermediate result (UML-Diagram)

based on some subtle design details. Our first semester students already had basic experience with primitive data types, objects and control structures at the time of this exercise. However, they did not have any design experience yet.

5.2. The Assignment

Students had to first deduce a data model to implement the hunt-and-kill-algorithm described in [37]. The idea we presented to the students as an intermediate result after the steps towards the data model is depicted in Fig. 4.

After presenting the data model students work on the algorithm. Step by step they are guided from an over-viewing perspective describing the interaction of the major steps towards the algorithmic details. The tasks we evaluated deal with the relationship of adjacent cells and how to carve passages from one to another.

Students had to document and turn in their answers to the guiding questions. These documents provide insights into the students' thinking processes and reveal their pictures in mind. Furthermore, we are able to derive results regarding the students' level of abstract thinking.

5.3. Tasks to be Evaluated in Detail

The first question (1) asked students to mark adjacent cells of a given cell S in a grid (the expected solution is shown in *gray*).

	N	
N	S	*N*
	N	

Furthermore, they are asked to visualize possible corner cases.
(2) Which special cases can you identify for other starting cells S? Please draw them.

These visualizations should help students to deal with the following two open format answers:

(3) How can you identify adjacent cells on the grid?

S		

	S	

(4) Given two adjacent cells A and B. How do you know which wall is to be removed?

	B	
	A	

	A	B

The result of these questions should guide students towards the subsequent implementation task.

6. Evaluation of Students' Answers

We used the proposed rating rationale to evaluate the correctness of the answers and their respective level of abstraction. The evaluation process was conducted by the professor together with a research assistant.

When rating the answers, we need an understanding of which formulations are indicators for the categories. The following list describes formulations showing up in students' submissions:

`Concrete`

- anecdotal description
- using the words "above" and "beyond"
- description of moving into a compass direction
- description of the examples in own words
- common edge

`Specific`

- argumentation based on x- and y-coordinates but restricted to the given examples
- formulations like "value 1 greater" (or less)
- essential properties like adjacent are defined

`Generic`

- argumentation based on x- and y-coordinates without restrictions
- formulations using mathematical symbols to describe relations
- all essential properties are defined accurate and clear

Afterwards, we merged the evaluation of these documents with the accompanying implementation-task, as well as the final exam.

7. Results

The class of our degree program Bachelor Computer Science starting in winter 2017 fulfilled the assignment described after two months of their software development class. The course consists of four hours lecture per week and two hours lab session. Students have to solve ten programming problems during the semester. It is required to achieve a certain score in the lab sessions to be admitted to the final written exam. However, the performance in the lab does not influence their grades. During the lab sessions they get advice by the lecturers and tutors.

This year's class consisted of 15% female students. 35% of the students have already accomplished an apprenticeship. Only half of the students has a good math grade which is often seen as an indicator for programming abilities[1].

From this course, we received 46 valid hand-ins via Moodle. We detected no cases of copy. Based on these data, we can now get deeper insights into students' competence of abstract thinking.

Tasks (1) and (2) are quite easy such that almost 95% of the students solved them correctly. However, if a student named two times the border case as a corner case, it can be interpreted as an indicator for a lower level of abstraction, as both cases can be normalized to build one case.

Tasks (3) and (4) are more difficult, as they require the abstraction of relationships and processes. Students had to write a text, so we were able to analyze both the correctness and the achieved level of abstraction.

Figure 5 shows the histogram of number of students that correctly solved tasks (3) and (4). As task (3) is easier than task (4) the distribution of correct and false answers is as expected.

Figure 6 shows the a histogram of number of students that reached the different levels of abstraction. We can see that our students tend to think on a concrete level. The more complex a task is the more they are limited to thinking on a lower abstraction level. This result is worse than we had expected after two months of class and after having implemented some algorithms.

Finally Figure 7 shows the joint frequencies of levels reached for tasks (3) and (4) in a scatter plot. As expected, only a few students achieved a higher level in the more difficult task (4). Most students answered both

[1] A: 15%, B: 31%, C: 29%, D:8%, E:10%, 7% no information

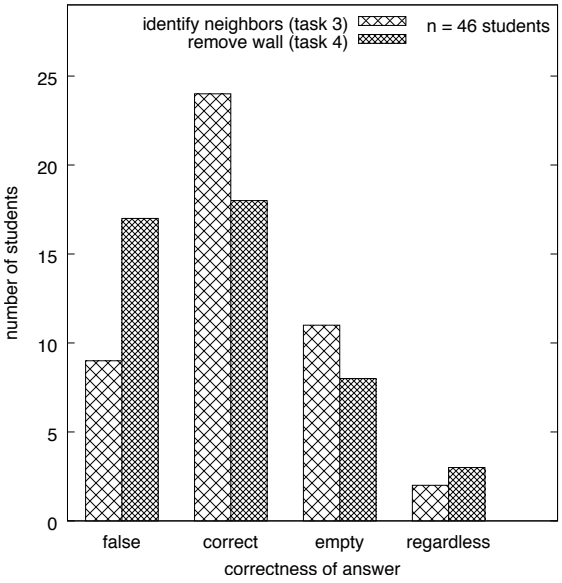

Figure 5. Histogram of the number of students that solved tasks (3) and (4) in a correct manner.

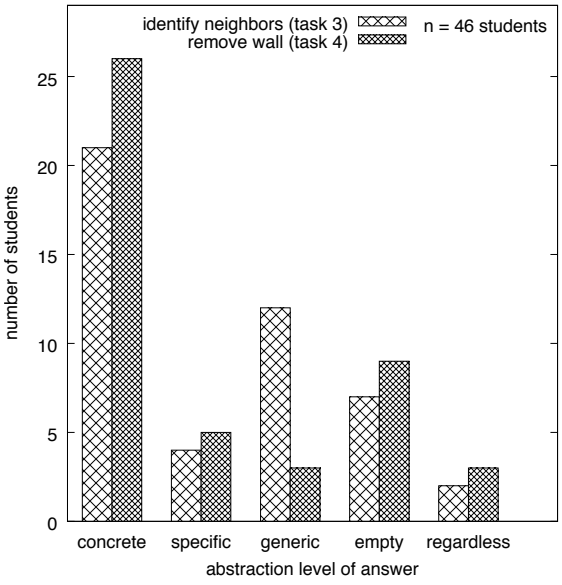

Figure 6. Histogram of the number of students that showed the respective level of abstraction in tasks (3) and (4).

Table 2. Rating for different examples of answers to Task (4), according to different rating perspectives. Translated by the authors while trying to maintain sloppiness in wording.

Answer	Correctness	Level
At the first: B(1,2), A(1,1) $\rightarrow$ same x-Coordinate, different y-Coordinate $\rightarrow$ $2 > 1$ $\rightarrow$ remove southern wall of B respectively the northern wall of A. Analogously for 2. example, but with western and southern wall.	False	Concrete
If going to the right, you have to remove the left Wall of B. *repeated 4 times with varying directions.*	False	Concrete
I don't understand the question. Every cell has only a right or a bottom wall. So I remove what I can remove. So in example 1 the bottom wall of B and in 2 the right of A, as the cells do not have a left and upper wall.	False	Concrete
At first you check the dependent position (e.g. xA + 1 = xB). Consequently you can see that A is right of B, so you can remove the left wall of A and the right wall of B.	False	Specific
You remove the wall where both cells have the same x- and y-coordinate.	False	Generic
If B is above or left of A, the lower respectively the right wall of B has to be removed. Otherwise, the lower respectively the right wall of A has to be removed.	Correct	Concrete
y-Position of B is greater then A: remove bottom wall of B x-Position of B is greater then A: remove rightWall of A.	Correct	Specific
If it concerns neighboring cells, I must remove a wall in A if the column or row index of A is smaller the one of B. Is it the column index, the right wall of A has to be removed, concerning the row index it is the bottom one. [*analogously for A index greater B*].	Correct	Generic

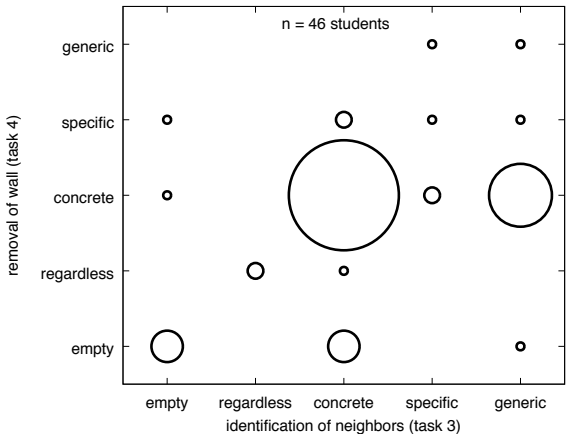

Figure 7. Joint frequencies of levels reached for tasks (3) and (4).

tasks on a concrete level. We expected more students to answer in a specific or even generic manner.

Our main findings are:

1. Only 28% of our students manage to answer both questions in a correct professional manner.
2. 60% of the students fail at task two.
3. Around 50% of the answers are on a concrete level, which means e.g. students use terms like beneath and above to describe relationships.
4. 25% of the students answer task one in an abstract way, whereas only 0.1% of the students manages an abstract description for the relationships in task four.
5. 50% of the students do not change the abstraction level during the tasks. A significant part of 25% changes to a lower level in task two.

8. Conclusion

In conclusion, we deduced a model for the competence of abstract thinking, based on current literature. The resulting definition of the competence construct is a contribution to close the research gap regarding the competence.

Based on the definition and common practice, a coding manual can be described to evaluate various open-ended questions that focus on abstract thinking. In a pilot-study, we used this coding manual to assess an assignment where students had to describe one aspect of an algorithm with the goal to implement it later on. The insights we gained by analyzing the results confirm our observations and will influence our future teaching.

In order to get deeper insights into the competence, we will increase this set of specific questions. We will then use these questions to build a questionnaire

to assess the competence in more depth in a larger empirical study. Consequently, teaching methods can be evolved that consider the insights gained by the assessments.

Acknowledgement

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

References

[1] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of software engineering*. Prentice Hall PTR, 2002.

[2] J. M. Wing, "Computational thinking," *Commun. ACM*, vol. 49, pp. 33–35, Mar. 2006.

[3] C. Dörge, *Informatische Schlüsselkompetenzen – Konzepte der Informationstechnologie im Sinne einer informatischen Allgemeinbildung (in German)*. PhD thesis, University of Oldenburg, 2012.

[4] W. Hartmann, M. Näf, and R. Reichert, *Informatikunterricht planen und durchführen*. Springer-Verlag, 2007.

[5] IEEE, "Software engineering body of knowledge, v3," 2013.

[6] Y. Sedelmaier, S. Claren, and D. Landes, "Welche Kompetenzen benötigt ein Software Ingenieur?," in *SEUH*, pp. 117–128, 2013.

[7] K. Devlin, "Special issue: Why universities require computer science students to take math," *Communications of the ACM*, vol. 46, no. 9, 2003.

[8] J. Kramer, "Is abstraction the key to computing," *Communications of the ACM*, vol. 50, no. 4, pp. 36–42, 2007.

[9] V. Thurner, A. C. Böttcher, and A. Kämper, "Identifying base competencies as prerequisites for software engineering education," in *Global Engineering Education Conference (EDUCON), 2014 IEEE*, pp. 1069–1076, April 2014.

[10] J. Bennedsen and M. E. Caspersen, "Abstraction ability as an indicator of success for learning object-oriented programming?," *SIGCSE Bull.*, vol. 38, pp. 39–43, June 2006.

[11] B. L. Kurtz, "Investigating the relationship between the development of abstract reasoning and performance in an introductory programming class," in *ACM SIGCSE Bulletin*, vol. 12, pp. 110–117, ACM, 1980.

[12] K. Koeppen, J. Hartig, E. Klieme, and D. Leutner, "Current issues in competence modeling and assessment," *Zeitschrift für Psychologie/Journal of Psychology*, vol. 216, no. 2, pp. 61–73, 2008.

[13] O. Hazzan and J. Kramer, "Assessing abstraction skills," *Commun. ACM*, vol. 59, pp. 43–45, Dec. 2016.

[14] I. Cetin and E. Dubinsky, "Reflective abstraction in computational thinking," *The Journal of Mathematical Behavior*, vol. 47, pp. 70–80, 2017.

[15] N. M. Seel, ed., *Encyclopedia of the Sciences of Learning*. Springer Science & Business Media, 2011. States open questions.

[16] R. Or-Bach and I. Lavy, "Cognitive activities of abstraction in object orientation: An empirical study," *SIGCSE Bull.*, vol. 36, pp. 82–86, June 2004.

[17] D. Nguyen and S. Wong, "Oop in introductory cs: Better students through abstraction," in *Proceedings of the Fifth Workshop on Pedagogies and Tools for Assimilating Object-Oriented Concepts*, 2001.

[18] O. E. Dictionary, "Oxford English Dictionary."

[19] Merriam-Webster.com, "Merriam webster dictionary," 2017.

[20] S. Loue and M. Sajatovic, *Encyclopedia of aging and public health*. Springer Science & Business Media, 2008.

[21] J. Locke, *An essay concerning human understanding*. T. Tegg and Son, 1836.

[22] J. Weinberg, "Abstraction in the formation of concepts," in *Dictionary of the History of Ideas: Studies of Selected Pivotal Ideas* (P. P. Wiener, ed.), Charles Scribner, 1973-1974.

[23] E. Rosch and C. B. Mervis, "Family resemblances: Studies in the internal structure of categories," *Cognitive psychology*, vol. 7, no. 4, pp. 573–605, 1975.

[24] R. Hershkowitz, B. B. Schwarz, and T. Dreyfus, "Abstraction in context: Epistemic actions," *Journal for Research in Mathematics Education*, pp. 195–222, 2001.

[25] S. Ohlsson and E. Lehtinen, "Abstraction and the acquisition of complex ideas," *International Journal of Educational Research*, vol. 27, no. 1, pp. 37–48, 1997.

[26] D. Davis, T. Yuen, and M. Berland, "Multiple case study of nerd identity in a cs1 class," in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, (New York, NY, USA), pp. 325–330, ACM, 2014.

[27] V. Davydov, "Soviet studies in mathematics education volume 2.," 1990.

[28] F. Moller and G. Struth, *Modelling Computing Systems: Mathematics for Computer Science*. Springer Science & Business Media, 2013.

[29] G. Voland, *Engineering By Design, 2nd*. Upper Saddle River, NJ, Pearson Prentice Hall, 2004.

[30] N. Weicker, B. Draskoczy, and K. Weicker, "Fachintegrierte Vermittlung von Schlüsselkompetenzen der Informatik," in *HDI*, pp. 51–62, 2006.

[31] I. C. Society, P. Bourque, and R. E. Fairley, *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. Los Alamitos, CA, USA: IEEE Computer Society Press, 3rd ed., 2014.

[32] P. Hubwieser, *Didaktik der Informatik Grundlagen, Konzepte, Beispiele*. Springer, 2007.

[33] O. Hazzan and J. Kramer, "Abstraction in computer science & software engineering: a pedagogical perspective," *Frontier Journal*, vol. 4, pp. 6–14, January 2007.

[34] B. Liskov and J. Guttag, *Abstraction and specification in program development*, vol. 180. MIT press Cambridge, 1986.

[35] The College Board, "Ap computer science principles," May 2017.

Table 3. Coding Manual for dimension correctness for tasks designed to measure the competence abstraction.

Correctness	
Empty	*Empty response field*
Regardless	The answer is without regard to the question. Buzzword
False	An answer is false, if it is deficient or partly deficient from a professional perspective.
Correct	An answer is correct, if it is accurate and complete from a professional perspective.

Table 4. Coding Manual for dimension level of abstraction for tasks designed to measure the competence abstraction.

Level of Abstraction	
Empty	*Empty response field*
Regardless	The answer is without regard to the question. Buzzword
Concrete	The answer describes the given examples using everyday or well-known terms.
Specific	The answer depicts a rule or a rule set, which can only be applied to the given examples.
Generic	The answer depicts a rule or a rule set, which can be applied to the given examples and beyond that.

[36] T. Colburn and G. Shute, "Abstraction in computer science," *Minds and Machines*, vol. 17, no. 2, pp. 169–184, 2007.

[37] J. Buck, *Mazes for Programmers: Code Your Own Twisty Little Passages*. Pragmatic programmers, Pragmatic Bookshelf, 2015.