Technische Universität München
Fakultät für Informatik

Computer Science and Abstract Thinking

Concept, Assessment, Training

by Daniela Zehetmeier

Technische Universität München
Fakultät für Informatik
Fachgebiet Programmierung und Anwendung verteilter Systeme

Computer Science and Abstract Thinking

Concept, Assessment, Training

Daniela Zehetmeier

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Priv.-Doz. Dr. Georg Groh
Prüfer der Dissertation:	1.	Prof. Dr. Anne Brüggemann-Klein
	2.	Prof. Dr. Axel Böttcher, Hochschule München
	3.	Prof. Dr. Veronika Thurner, Hochschule München

Die Dissertation wurde am 17.06.2019 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 05.08.2019 angenommen.

Abstract

Researchers and lecturers commonly agree that the competence of abstract thinking is of utmost importance in computer science. It is therefore striking that few sources precisely and completely define abstract thinking and its cognitive processes. As a consequence, there is a lack of assessment tools for this competence. These issues mean that concepts to support students in developing higher levels of competence in a university setting cannot be evaluated and their development lacks empirical basis. In this dissertation, I narrow the gap between the general appreciation of abstract thinking competence and its teaching to first-year students through the following four contributions:

Firstly, there is the conceptualisation of abstract thinking competence, where a competence model is derived from theoretical descriptions and scientific literature. The final model consists of the three competence components: *Commonalities and Differences*, *Hide and Keep*; and *Expand*. These components summarise cognitive processes related to abstract thinking.

Secondly, there is the development of an assessment tool called Abstract Thinking Assessment (ATA) to evaluate students' competence of abstract thinking. The ATA is aimed at first-year students undertaking a bachelor's degree in computer science or related topics; the ATA considers and utilises current research. For each competence component of abstract thinking, one or more types of questions are presented and it is then explained how they have been implemented in different test items of the ATA. In order to evaluate students' answers objectively, a rating rationale as well as a detailed coding manual are developed. These recommendations guide evaluators through a qualitative rating process and help to obtain unbiased results.

Thirdly, there is the description of the application of the ATA in two studies *Main* and *Post*, and the analysis of the data collected. 134 first-year students of the bachelor's programmes in computer science and scientific computing participated in the *Main study*. First, the data are used to validate the assessment against common quality criteria, which confirms a good quality of the ATA. Second, the analysis of the student population gives lecturers an overview of their cohort and enables them to pick up students at the level they are in terms of learning. The analysis shows a very high heterogeneity regarding the competence of abstract thinking. Moreover, the data reveal that novice programmers focus on components and attributes when describing a set

of entities while neglecting their dynamic aspect. Furthermore, the comparison of students' results in the *Main* and *Post study* discovers that the development of students' competence is rather small within one semester, but significant. In order to give more evidence to the importance of teaching abstract thinking competence, known hypotheses on the lack of abstract thinking competence and its impact are verified. Analyses indicate that the lack in competence is rather small when considering the level of abstraction solely. However, if the correctness and the level of abstraction of a solution are combined in the evaluation, students' results show a significant deficit. Moreover, this thesis confirms what researchers and lecturers have long suspected: the competence of abstract thinking is necessary to acquire professional software development skills. However, it also reveals that the competence alone is not sufficient.

Fourthly, it is shown how all findings concerning the competence of abstract thinking and further research directly influence the development of future teaching practice. A teaching concept is designed that explicitly addresses the competence of abstract thinking integrated into a course on software development. This dissertation presents the development of teaching units and their realisation in class. In addition to the teaching material, the teaching style has been changed from a traditional approach to pair teaching. Lecturers can have different roles during the lecture that are outlined in this thesis. Amongst others, students state that the lecture picks them up at the right level of competence. Moreover, they are aware that abstract thinking is an important competence in software development.

In conclusion, this dissertation narrows the research gap regarding the competence of abstract thinking and gives an example of how to define, measure and teach competences in general. In addition, this thesis also significantly contributes to the ongoing debate surrounding abstract thinking competence.

Zusammenfassung

Forschende und Dozierende sind sich einig, dass die Kompetenz des abstrakten Denkens in der Informatik von größter Bedeutung ist. Es ist daher erstaunlich, dass nur wenige Quellen das abstrakte Denken und seine kognitiven Prozesse präzise und vollständig definieren. Infolgedessen fehlen Messinstrumente für diese Kompetenz. Diese Problematik führt dazu, dass Konzepte zur Unterstützung der Studierenden bei der Entwicklung höherer Kompetenzniveaus in einem universitären Umfeld nicht evaluiert werden können und keine empirischen Grundlagen für deren Entwicklung vorliegen. In dieser Dissertation schließe ich die Lücke zwischen der allgemeinen Wertschätzung der Abstraktionskompetenz und ihrer Vermittlung an Studienanfängerinnen und -anfänger durch die folgenden vier Beiträge:

Der erste Teil der Dissertation ist die Konzeptualisierung der Kompetenz des abstrakten Denkens, bei der ein Kompetenzmodell aus theoretischen Beschreibungen und wissenschaftlicher Literatur abgeleitet wird. Das endgültige Modell besteht aus den drei Komponenten: *Commonalities and Differences*, *Hide and Keep* und *Expand*. Diese Komponenten fassen kognitive Prozesse zusammen, die sich auf die Kompetenz des abstrakten Denkens beziehen.

Zweitens wird die Entwicklung eines Assessment-Tools namens Abstract Thinking Assessment (ATA) beschrieben, mit dem die Kompetenz der Studierenden im Bereich des abstrakten Denkens bewertet werden kann. Das ATA richtet sich an Studienanfängerinnen und -anfänger die einen Bachelorabschluss in Informatik anstreben. Es berücksichtigt und nutzt aktuelle Forschungsergebnisse. Für jede Kompetenzkomponente des abstrakten Denkens werden eine oder mehrere Arten von Fragen vorgestellt. Anschließend wird erklärt, wie diese in verschiedenen Testaufgaben des ATA umgesetzt wurden. Um die Antworten der Studierenden objektiv bewerten zu können, werden ein Bewertungsschema sowie ein detailliertes Kodierhandbuch entwickelt. Diese Handreichungen leiten Beurteilende durch einen qualitativen Evaluationsprozess und tragen dazu bei, unverfälschte Ergebnisse zu erzielen.

Im dritten Teil erfolgt die Beschreibung der Anwendung des ATA in zwei Studien *Main* und *Post* sowie die Analyse der erhobenen Daten. 134 Studienanfängerinnen und -anfänger der Bachelorstudiengänge Informatik und Scientific Computing haben an der Hauptstudie teilgenommen. Zuerst werden die Daten verwendet, um das Messinstrument gegen allgemein bekannte Qualitätskriterien zu prüfen, was eine gute Qualität

des ATA bestätigt. Zweitens gibt die Analyse der Gesamtheit der Studierenden den Dozierenden einen Überblick über ihre Kohorte und ermöglicht es ihnen, Studierende auf dem Kompetenzniveau abzuholen, auf dem sie sich befinden. Die Analyse zeigt eine sehr hohe Heterogenität in Bezug auf die Kompetenz des abstrakten Denkens. Darüber hinaus zeigen die Daten, dass sich Programmieranfängerinnen und -anfänger bei der Beschreibung einer Menge von Entitäten auf Komponenten und Attribute konzentrieren und dabei deren dynamischen Aspekt vernachlässigen. Darüber hinaus zeigt der Vergleich der Ergebnisse von *Main* und *Post Study*, dass die Kompetenzentwicklung der Studierenden innerhalb eines Semesters eher gering, aber signifikant ist. Um die Bedeutung der Lehre von abstraktem Denken besser zu belegen, werden bekannte Hypothesen zum Mangel an Abstraktionskompetenz und deren Auswirkungen überprüft. Auswertungen zeigen, dass der Kompetenzmangel bei alleiniger Betrachtung der Abstraktionsebene eher gering ist. Wird jedoch die Richtigkeit und der Abstraktionsgrad einer Lösung in die Bewertung mit einbezogen, weisen die Ergebnisse der Studierenden ein signifikantes Defizit auf. Darüber hinaus bestätigt diese Arbeit, was Forschende und Dozierende seit langem vermuten: Die Kompetenz des abstrakten Denkens ist für den Erwerb professioneller Fähigkeiten in der Softwareentwicklung notwendig. Es zeigt sich jedoch auch, dass die Kompetenz keine hinreichende Voraussetzung ist.

Viertens wird gezeigt, wie alle Erkenntnisse über die Kompetenz des abstrakten Denkens und weitere Forschung die Entwicklung der zukünftigen Unterrichtspraxis direkt beeinflussen. Es wird ein Lehrkonzept für das Modul Softwareentwicklung entwickelt, dass explizit auf die Kompetenz des abstrakten Denkens eingeht. In dieser Dissertation werden die Entwicklung von Unterrichtseinheiten und deren Umsetzung im Unterricht vorgestellt. Zusätzlich zum Unterrichtsmaterial wurde der Unterrichtsstil von einem traditionellen Ansatz zu Pair-Teaching geändert. Dozierende können während der Lehrveranstaltung unterschiedliche Rollen einnehmen, die in dieser Arbeit beschrieben werden. Die Studierenden geben unter anderem an, dass die Lehrveranstaltung sie auf der richtigen Kompetenzstufe abholt. Darüber hinaus sind sie sich dessen bewusst, dass abstraktes Denken eine wichtige Kompetenz in der Softwareentwicklung darstellt.

Zusammenfassend wird in dieser Dissertation die Forschungslücke in Bezug auf die Kompetenz des abstrakten Denkens geschlossen und ein Beispiel für die Definition, Messung und Förderung von Kompetenzen im Allgemeinen gegeben. Darüber hinaus leistet diese Dissertation einen wesentlichen Beitrag zur laufenden Debatte um die Kompetenz des abstrakten Denkens.

Contents

Introduction and Overview

It is commonly agreed that the competence of abstract thinking is one of the most important competences in computer science [3, 15, 20, 29, 34, 53, 66, 79, 84, 93, 99]. This is because numerous concepts in computer science are abstract by nature [20, 29, 41].

Associations like IEEE, ACM [90] and the German GI [33] state in their curricula for computer science that abstraction is a 'fundamental concept' and a 'key component'. The US Advanced Placement Courses name abstraction as a central problem-solving technique [89], and so does the Guide to the Software Engineering Body of Knowledge (SWEBoK [82]). Furthermore, textbooks for teaching computer science refer to abstraction as a fundamental concept [7, 46, 58].

Abstract thinking is stated as one of the twelve most important competences of software engineers investigated by Sedelmaier [79]. Dörge [29, p. 421] defines abstract thinking as a key competence in computer science, as it plays a role in all core areas of computer science. Keith Devlin [28] 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 incoming students lack in competences, which are necessary to study computer science or related topics successfully, especially in abstract thinking [9, 53, 55, 93]. Hence, it should be the goal of computer science education at universities, to develop abstract thinking within the students, right after they have been admitted to a university.

Although there exist many scientific sources asserting 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.

Kramer [53] calls for a sound understanding of the competence of abstract thinking. The author claims that it is essential to the future of the computer science profession to be able to test and teach abstract thinking. In general, only a sound competence model allows the development of a valid assessment method [52]. In a later publication [44] on how to assess abstract thinking, Hazzan and Kramer suggest question patterns for abstract thinking as a first attempt to tackle the challenge of measuring the competence of abstract thinking. Hazzan and Kramer did not follow up on their research, and a verified assessment method still appears to be missing.

There exist numerous concepts for teaching abstract thinking [13, 23, 44, 66, 68, 84]. I agree with Kramer that "before we can control or effect, we must first measure" [53]. As there exists no competence model and no possibility to measure the competence of abstract thinking, the effectiveness of teaching concepts is not validated. Additionally, statements concerning the competence of abstract thinking and its influence on the success in studying computer science remain hypotheses.

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 upon, there is no consistent and complete definition of the competence. Nor has it been statistically validated that the competence is a key for acquiring computer science skills. Thus, it has to be justified to prove the necessity of abstract thinking competence in the curriculum of computer science. Due to the lack of an assessment tool, the effectiveness of existing teaching concepts cannot be validated.

My overall goal is to improve the competence of abstract thinking among first-year students in computer science, which contributes to the improvement of university teaching in this field. For this purpose, I provide a competence model as well as a valid assessment tool to meet the demands of Kramer [53]. I focus on the study entry phase, as this phase forms the basis for the students' study progress and it shows a significant drop out [36, 42, 50, 93]. Moreover, this dissertation focusses on a single competence, abstract thinking, which is often said to be a prerequisite for studying computer science successfully.

Consequently, the following research questions and intermediate research questions for this work are:

How can the competence of abstract thinking be defined in the context of computer science based on current literature?

How can the competence of abstract thinking be measured in the context of computer science?

Is there a deficit in first-year students' initial competence of abstract thinking?

Is there a relation between the competence of abstract thinking and the acquisition of professional software development skills?

How can students' competence of abstract thinking be promoted in a class on software development?

Based on the research questions, the following goals of this work can be derived:

Theoretical competence model
Theoretical competence model of abstract thinking focussing on a computer science perspective to create the basis for assessment and teaching.

Valid assessment tool
Valid assessment tool to evaluate known hypotheses on the abstract thinking competence of first semester students in computer science.

Verification of the assessment tool
Verification of the assessment tool to ensure the quality of the assessment tool.

Confirmation of two crucial hypotheses
Confirmation of two crucial hypotheses on the competence of abstract thinking: deficit in the competence and impact of the competence on the acquisition of professional skills to close the research gap and to justify a teaching concept that additionally addresses the competence of abstract thinking.

Teaching concept
Exemplary teaching units to promote students' competence of abstract thinking based on the research results. And a teaching approach to additionally support the teaching units.

In this dissertation, I achieve these goals by proposing a theoretical model of the competence of abstract thinking, focusing on a computer science perspective. Furthermore, I present an assessment method and the accompanying rating rationale. Additionally, I conduct three studies to verify the instrument and to collect data. Based on the insights gained by student answers, known hypotheses are investigated. Moreover, implications for teachings abstract thinking integrated into a CS-1 class are presented.

Scope and overview of this work

In my thesis, I devise a competence model, an assessment tool, and a teaching concept for the competence of abstract thinking. Different researchers, such as Kramer [53] and Thurner et al. [93], hypothesise that this competence is essential, but underdeveloped, in incoming computer science students. Under the assumption that the competence is crucial, I investigate the abstract thinking competence of first-year students' using the proposed assessment tool. The competence model as well as the findings of the analysis serve as an impulse for developing an innovative teaching concept. The four main goals that I have identified above inform the structure of the dissertation.

Part I: Conceptualisation
How can the competence of abstract thinking be defined in the context of computer science based on current literature?

The first result of this thesis is a theoretically founded model of the competence of abstract thinking. A literature review forms the basis for the competence model, where relevant statements are collected, structured and synthesised. This part of the dissertation also includes theory regarding competences in general. The theoretically derived competence model is the foundation of the assessment method, as well as the teaching practice.

Part II: Operationalisation
How can the competence of abstract thinking be measured in the context of computer science?

At the beginning of the operationalisation, the target group and the planned setting are described in detail. From this, requirements for the assessment tool are derived. After that, the research method is chosen, based on a short evaluation of common methods with respect to the predefined requirements.

Subsequently, the assessment tool called Abstract Thinking Assessment (ATA) is developed. In the first step, performance indicators are derived from the competence model. They form the basis to evaluate and develop question patterns. They also represent typical situations where a specific competence is needed. Based on these general descriptions, concrete problems are deduced, which are called 'items' in assessment theory. As a result of discussion with experts, a subset of items forms the first version of the ATA. This undergoes a pilot study to find further improvements and evaluate the requirements derived from the setting. The findings obtained are used to revise the assessment tool and end up in the final ATA.

One of the requirements is that the answers can be judged by some kind of standard. Therefore, a general rating rationale is developed. This represents a blueprint for the evaluation of each item and is completed by sample answers from the pilot study. This rating rationale is then further refined for every individual task. All this together forms the coding manual, which is used to rate student answers. In order to prepare the rated answers for analysis, they are scored.

Part III: Analysis
Is there a deficit in first-year students' initial competence of abstract thinking?
Is there a relation between the competence of abstract thinking and the acquisition of professional software development skills?

The ATA is applied in a *Main* and a *Post study* at the Munich University of Applied Sciences. Students of the bachelor's programmes in computer science and scientific computing took part in the assessment in the early days of their first semester. The same students fill in the identical assessment at the beginning of their second semester. These data allow to verify hypotheses from literature and to get insights into the students' level of competence. Additionally, the collected data are used to analyse the quality of the assessment itself.

Part IV: Training
How can students' competence of abstract thinking be promoted in a class on software development?

The last part presents the development of a teaching concept for the module *Software Development I*. It explicitly addresses the competence of abstract thinking using an integrated approach. Several teaching units are described in detail from design to realisation in class. Additionally, the teaching style pair teaching has been introduced as part of the teaching concept. Useful roles lecturers can take are outlined. An evaluation of the results of the end-of-term evaluation completes this part.

Parts of this thesis have been published in:

– Designing Lectures as a Team and Teaching in Pairs; by Daniela Zehetmeier, Axel Böttcher and Anne Brüggemann-Klein. [100]
– Defining the Competence of Abstract Thinking and Evaluating CS-Students' Level of Abstraction; by Daniela Zehetmeier, Axel Böttcher, Anne Brüggemann-Klein and Veronika Thurner. [101]

Part I.

Conceptualisation

Modelling Abstract Thinking Competence

Conceptualisation

There exists a variety of reasons why research on competences is conducted. My research question *"Is there a deficit in first-year students' initial competence of abstract thinking?"* indicates a diagnostic purpose. But before this question can be answered, a competence model is needed as a basis [52]. The aim is to identify crucial factors of the competence of abstract thinking. This competence model then helps to reflect on student competences in a systematic way, and thus, forms the foundation for an assessment tool, which will be introduced in Part II.

This part describes the development of a competence model for abstract thinking in the context of computer science. It represents the competence in terms of specific basic skills and abilities. Hence, I reach a comprehensive understanding of abstract thinking in the context of computer science. The competence model is a theoretically deduced description of the competence. It is a synthesis of literature from various fields, like psychology and computer science, and is extended by professional experience. The underlying notion of competences is based on F. Weinert [98].

1. Literature Review: Competences in General and Definitions of Abstract Thinking Competence

Looking in more detail at abstract thinking, the term competence or a periphrasis of competence often occur. Competences have increased in importance not at least because of the Bologna Process [64]. Based on recent research, competences are attributed with the following characteristics: become visible through performance in real life situations, depend on the domain, and are learnable (see Section 1.1).

Once the term competence is clarified, the chapter continues with the evaluation of terms that appear in the context of the competence of abstract thinking, like purpose and level (see Section 1.2). The chapter ends with an overview of the current state of literature regarding definitions of the competence of abstract thinking. Although several scientific sources consider abstract thinking to be highly relevant, they do not state a definition or a competence model.

1.1. Term Competence

Over the last years, acquiring competences has been identified as a main goal of education at every level. In our modern and industrial society, professional qualifications cannot be limited to knowledge passed on from generation to generation [51, p. 8]. The statement of Keith Devlin affirms this evolution:

> *"A common view of education is that its main aim is the acquisition of knowledge through the learning of facts. [...] But it's simply not right. [...] The goal of education is to improve minds, enabling them to acquire abilities and skills to do things they could not do previously" [28].*

Since the Bologna Process [64] has been established in 1999, many people have developed an intuition what is meant by competences. However, most of them are not able to precisely define or clearly differentiate them [98]. Hence, it is helpful to establish a common understanding of competences based on recent literature. This section presents widely-used definitions and concludes with one that is used throughout this work.

Most scientific work interprets competence as a (specialised) set of knowledge, abilities, and skills that professionals use to reach a specific goal as e.g. defined in [30, 81, 98]. The competence definition that is probably most often cited originates from Weinert:

> *"First, this concept refers to the necessary prerequisites available to an individual or a group of individuals for successfully meeting complex demands. [...]*
> *Second, it should be used when the necessary prerequisites for successfully meeting a demand are comprised of cognitive and (in many cases) motivational, ethical, volitional, and/or social components.*
> *Third, the concept of competence implies that a sufficient degree of complexity is required to meet demands and tasks. Those prerequisites that can in principle be fully automatized can also be characterized as skills. The boundary between skill and competencies is fuzzy.*
> *Fourth, learning processes are a necessary condition for the acquisition of prerequisites for successful mastery of complex demands. This means much must be learned, but cannot be directly taught"[98, p. 62 f]*

A definition that is comparable with Weinert's was given by Shavelson:

> *"Competence (1) is a physical or intellectual ability, skill or both; (2) is a performance capacity to do as well as to know; (3) is carried out under standardized conditions; (4) is judged by some level or standard of performance as 'adequate,' 'sufficient,' 'proper,' 'suitable' or 'qualified'; (5) can be improved; (6) draws upon an underlying complex ability; and (7) needs to be observed in real-life situations" [81, p. 44].*

The most important aspect of Shavelson's definition is that competences are learnable and improvable. Additionally, the author also includes the fact that competences are demonstrated when meeting demands, like Weinert did before.

Erpenbeck gives a very general and vague definition of competences. The author indicates that competences comprise skills, knowledge and qualifications. However competences cannot be reduced to this. Competences have something in addition, which facilitates the ability to act in open, uncertain and complex situations [30, p. XII].

Hartig & Klieme [39, p. 131] describe competence as contextualised. This means that competences are abilities to master specific situations and demands. Furthermore, they state that competences are learnable. They can be acquired through experiences with certain demands and situations. And according to Schaper [76] the competence term for academia also includes the aspect of reflection, i.e. the awareness of the own competences.

Based on these well-known definitions, this work considers competences as learnable and that they can be improved through conscious support. Competences are linked to the mastery of specific demands and situations. Following this reasoning, demands and situations need to be analysed in order to define a competence. Moreover, criteria have to be defined to give evidence whether and to which extent demands are mastered and thus competences are present.

1.2. Context of Abstract Thinking

This section presents and categorises terms that appear in the context of abstract thinking. In everyday use, the term is often associated with the notion of difficulty or sophistication. When looking into literature, one can find several terms that are used synonymously to name the process of building abstract mental models. Additionally, most scientific sources state that an abstraction is formed for a particular purpose. However, their descriptions of a purpose vary. Additionally, three different dimensions for characterising the process of abstract thinking in more detail are identified: *artefact*, *number of artefacts* and *abstraction level*.

1.2.1. Terms

There exist two terms in literature that are used to describe the process: abstraction and abstract thinking. Abstraction is also used to characterise the result of the process [80]. This ambiguity causes discussions and incomprehension. In order to avoid misunderstandings and to be clear about the terms, I will use the term abstract thinking to refer to the mental process. How this process is defined in detail is described in the following (especially Chapter 2). In this work, the term abstraction is used to specify the mental representation, which is the final result of the process.

1.2.2. Colloquial Use

The first we hear from our students when talking about abstract thinking is that they associate it with being difficult. Furthermore, the term abstract is commonly used in phrases like "That's too abstract for me" to express that something is complex or sophisticated. This common feeling is even reflected in dictionaries. The dictionary Merriam Webster [62] defines abstract as "difficult to understand". This might be the case as abstractions are independent of specific instances [41, 62, 73].

Such an association of abstract thinking with being difficult has consequences for teaching this key competence in computer science. Students might have an unconscious barrier when learning the competence or might give up more quickly. This will gain further importance in Part IV.

1.2.3. Purpose

Along with the process of abstract thinking, the importance of the purpose is often highlighted [41, 43, 53, 82]. Most publications state a rather vague purpose, like "focus on the 'big picture'" [82] or "highlight the substantial parts" [41, p. 115]. Kramer [53] writes that "the level, benefit, and value of a particular abstraction depend on its purpose". When analysing the context of this statement, Kramer is much clearer about the purpose than other sources are. He uses the evolution of the London Underground map to illustrate the importance of a purpose during an abstract thinking process. Underground maps have the purpose of visualizing intersections between railway lines clearly, whereas the geographical distances are not relevant. However, if an abstraction is used for another purpose, in this case walking between sights in London, the abstraction gives misleading information.

This aspect is important when it comes to the design of the test instrument in Part II. Items need to state a purpose in order to guide the abstract thinking. Furthermore, it is of high importance to point out the role of the purpose when teaching abstract thinking (cf. Part IV). During abstract thinking, it is necessary to always keep an eye on the desired goal and check whether one is still on the right track. When working with existing abstractions, which were created for a certain purpose, it needs to be evaluated whether they are suitable or not to solve the problem.

1.2.4. Dimensions

When it comes to the development of items in Part II, the following dimensions should be considered to cover the entire spectrum of the competence. Additionally, this could help students to structure their mental processes and thus to support them while training the competence.

Artefact. One aspect that is relevant with regard to the competence of abstract thinking is the kind of *artefacts* that are considered during the abstraction process. Literature distinguishes between static entities and dynamic procedures while abstracting [25, 45]. In my opinion this facet will particularly influence the teaching of this competence. Especially in computer science, static entities and dynamic processes differ in their handling. This is reflected by the fact that in object-oriented programming there exist two concepts for implementing static and dynamic abstraction: attributes and methods.

Number of Artefacts. Another facet that influences the abstraction process itself is the *number of artefacts* at the beginning of the process. When looking at the example in the paper of Kramer [53] the abstraction of one single entity is developed. This process of focussing on one single object is also described by Davydov [27]. In summary, this procedure can be seen as finding the essence of a subject for a certain purpose. Other authors state that the abstraction process starts with two or more objects and aims to integrate them into one structure (cf. [45, 58, 67]). Hence, it is a many-to-one mapping in order to build an intersecting set.

Abstraction Level. According to SWEBOK Version 3 [82], while abstracting one focusses on one *abstraction level* of the problem or concept at a time (cf. also [43]). However, concentrating on one level does not mean the person does not know anything about the neighbouring levels. Furthermore, Kramer [53] adds that the level has to be chosen carefully to keep the important information and neglect the irrelevant. Guidance for selecting an appropriate level is given by the purpose [53]. When talking about abstract thinking and levels of abstraction, I see both as abstract thinking: building higher level abstractions or working with higher level abstractions and thereby staying on the same level. In contrast, I see building alternate abstractions [82], like transferring a class-diagram into code, not as a process in the sense of abstract thinking. I consider such processes as systematic translations.

1.3. Definitions of Abstract Thinking

In a first step towards a competence model of abstract thinking, I have conducted a literature review, which was guided by the recent work of Hazzan & Kramer [44] as well as Cetin & Dubinsky [19]. I started with looking up the terms *abstract*, *abstraction*, and *abstracting* in English dictionaries and encyclopaedias. The review continued with searching for *abstraction*, *abstract thinking*, and synonyms like *abstraction skills* and *thinking in an abstract way*, or translations into German (*Abstrahierfähigkeit*, *Abstraktionsfähigkeit*) in scientific publications.

The goal of the literature review was to find definitions that describe the competence of abstract thinking. Table 1.1 summarises my findings by quoting relevant statements that can be seen as competence descriptions. This collection is the starting point for deriving a competence model. In addition to publications that indicate a competence definition, there exist some [66, 68, 80] that state the relevance of abstract thinking, try to measure the competence or teach the competence, but do not (clearly) specify what is meant by the competence of abstract thinking, and thus, its underlying cognitive processes and demands. In a definition of this kind [80, p. 31]: *"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."*, the crucial concept of the mental process is underspecified. Furthermore, there exist publications that just mention abstract thinking as a process students need to be able to perform in certain situations. For example, students should be able to use this technique while solving problems [82] and e.g. master this fundamental design concept and principle [90]. This needs further clarification in order to develop a competence model.

	Quote	Reference
#1	“The process of considering something independently of its associations or attributes”, “Consider something theoretically or separately from (something else)”, “Extract or remove (something)”	[73]
#2	“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”	[62]
#3	“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.”	[61]
#4	“drawing out of common features [...] represent the essential underlying relationships and the irrelevant aspects of the problem are ignored. ”	[80, p. 31]
#5	“a process of omitting all individuating features, and retaining only what is common to all of a set of resembling particulars” ([59] cited from [97])	[59]
#6	“[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”	[45]
#7	“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”	[67]
#8	Can appear in two shapes: idealisation (process) and extraction (product, result) *(translated from German)*	[26]
#9	“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]
#10	“The act or process of leaving out of consideration one or more properties of a complex object so as to attend to others”	[65, p. 11]
#11	Reduction of complexity or formalization *(translated from German)*	[29, p. 427]
#12	“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.”	[95, p. 240]

Quote		Reference
#13	The ability to abstract is the skill to detach circumstances from specific details and to focus 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.	[96]
#14	"[...] 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' "	[82, p. 13-4]
#15	"process of removing details to simplify and focus attention"	[53]
#16	Common characteristics are carved out or an amount of events with common features is summarised.	[46, p. 212 ff]
#17	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	[41, p. 115].
#18	"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."	[43]
#19	"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. It is crucial to remember, however, that relevance often depends upon context. [...] Abstraction by parametrization allows us, through the introduction of parameters, to represent a potentially infinite set of different computations with a single program text that is an abstraction of all of them."	[58, p. 11]
#20	"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"	[88, p. 15 f]
#21	"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."	[19]
#22	dropping (or ignoring, or neglecting) spatial and temporal features with which one is not concerned.	[20]
#23	"somehow subtracting specificity, so that an incomplete description of the original concrete entity would be a complete description of the abstraction"	[57]

Table 1.1.: Summary of the literature review presenting definitions and process descriptions.

2. Development of the Competence Model

In a next step, I have analysed the quotes listed in the literature review (see Section 1.3). During the review, I focused on the processes described by each quote. Often, it was sufficient to look at the verbs that are used. Afterwards, I 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. This cluster is labelled with *Commonalities and Differences*. Processes involving removal or reduction are comprised in the cluster called *Remove*. The 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 (entitled *Expand*).

I was almost satisfied with these four clusters. However, when solving problems that require to 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 reflected choice, driven by a purpose. Furthermore, I agree with Colburn & Shute [20], who describe the process of removing details as hiding, but not neglecting them, as the information might be important in a lower level of abstraction. Thus, I suggest the term *Hide* instead of *Remove*, as I interpret the term of removing as an irreversible process. Consequently, I end up with three clusters by combining those two. The resulting second cluster is finally called *Hide & Keep*.

Figure 2.1 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.

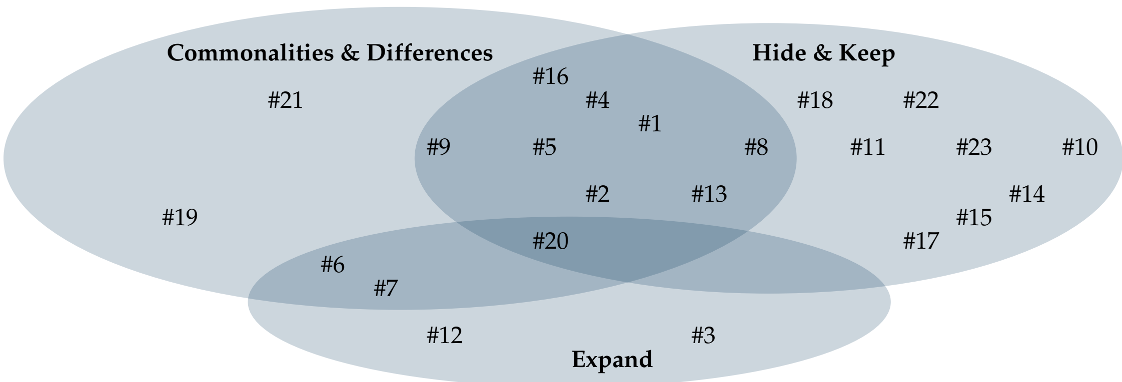


Figure 2.1.: Venn diagram of quotes (Table 1.1) assigned to the clusters they address.

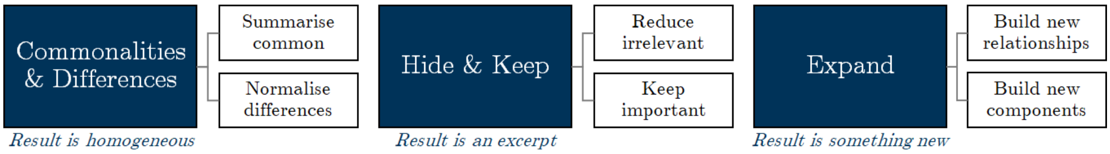


Figure 2.2.: Final Model stating the identified aspects of abstract thinking.

The identified clusters build the three components of my competence model of abstract thinking. In summary, the components are defined as follows (cf. Figure 2.2) and are completed by representative quotes:

– Identifying commonalities in order to summarize them and determining differences to normalize them
Abstraction by parametrization allows us, through the introduction of parameters, to represent a potentially infinite set of different computations with a single program text that is an abstraction of all of them." [58]

– Deciding which information is essential for the intended purpose and which is not
"Extract or remove (something)" [73]

– Revealing relationships between items or processes and building new components and concepts
"abstractions are constructed by assembling available ideas into new structures. [...]" [67]

In the following subsections, I illustrate the components of the proposed competence model and their relevant literature in more detail. Additionally, I describe examples that are related to computer science and require the competence component in focus.

2.1. Commonalities and Differences

As one can see in the citations in Table 1.1, a process that is often associated with abstract thinking is identifying *commonalities and differences*. Weicker et al. [96] describe abstract thinking as the skill to identify commonalities and to normalise differences. Normalising differences is an aspect typical for computer science. Liskov & Guttag [58] describe the process of normalising as 'abstraction by parameterization'. This process is described as treating "things that are different as if they were the same" and can be achieved by using parameters. The resulting abstraction then "represent[s] a potentially infinite set of different computations" [58]. Thus, the authors describe a concrete method how normalisation of differences can be accomplished. This competence component finds further underpinning in [19, 45, 62, 80, 97].

A typical example for finding and summarizing commonalities or normalizing differences is to parametrise various similar entities in order to integrate them into a blueprint. A concrete problem that requires these processes could look like the following. There exist several circles of uniform size and colour, which are spread along the x-axis. Consequently, the x-coordinate of the circle is a parameter and varies in every appearance. All other commonalities, like y-coordinate or colour, remain constant and are explicitly named in the description of the object.

This applies not only to entities, but also to processes. A typical example for *commonalities and differences* in such a setting are maps in the context of programming. When using maps, a given function f is applied to each element e of a list. In a simple example, all elements in the list are increased by one ($f(e) = e + 1$). When looking at this mechanism in detail, this step seems to be more advanced than normalising an x-coordinate. However, the mental process behind it is the same.

The result of this process is a homogeneous mental model that is devoid of differences. It is possible to describe the characteristics of a large number of objects or processes in a simple, comprehensive and precise manner, with only one description (text, picture or code) [58]. The other way around it is possible to replace every specific object by another equivalent object such that all statements remain valid [60].

2.2. Hide and Keep

An even more frequent aspect that comes up in the definitions shown in Table 1.1 is to remove information in order to focus on the essentials [53, 58, 62, 65, 73, 88, 96]. When revealing the essential features [53, 58, 82], it is important that this happens for a specific purpose. One needs also to concentrate on the desired level of abstraction, in order to keep the necessary information and neglect the irrelevant [53, 82].

Defining classes based on given entities, is a frequent task in object-oriented design and programming. Depending on the purpose of the abstraction, the resulting classes might differ. In contrast to *commonalities and differences* a reflected choice is required, whether to keep or neglect information given by the entities.

The two aspects that influence the result of a *Hide & Keep* process are purpose and level. When both are taken into account, the result can be considered as an excerpt. The clearer the purpose and the intended level, the less scope for discussions arises.

2.3. Expand

A process that is less frequently described in definitions of abstract thinking is the process of assembling available ideas into new and more abstract structures, as well as, discovering relationships between entities or steps of a process that can be described by e.g. logical operations and parameters [67]. This kind of new structure appears to be easier to understand, but is increasingly difficult to develop [58].

Based on the previously described example of the circles that are spread along the x-axis, a problem in the sense of an *expand* process can be developed. The example is supplemented by the process of printing yellow circles with a specific diameter ten times in a constant distance starting at a predefined point. Students can parametrise the circle using the process described before. Additionally, they now need to keep track of the number of circles printed so far and they need to find a representation of the coordinates where the circle has to be printed, depending on the number of the circle, e.g. presented in a mathematical statement. All this has to be combined with a repeating statement in order to solve the problem.

Such processes can end up in more advanced designs, like high-level programming languages or new numbers.

The higher the level of abstraction is, the more complex is the definition of high-level programming languages. However, high-level programming languages are more intuitive, as well as, easier to read and to learn than e.g. low-level languages, as they hide a lot of complex single steps.

Another significant example for *expand* processes is the development of new number systems. We put ourself in a state in which we did not know about real numbers and think of the common example of calculating the length of the diagonal of a unit square (side length = 1). From the Pythagorean theorem it follows that the length must be $\sqrt{2}$. However, $\sqrt{2}$ is not a number we know so far in $\mathbb{Q}$. There exist several approaches to define the number space of real numbers, and all of them show that a new concept is necessary to represent the circumstances [8].

The result of an expand process is described as something new. It can be a relationship, component or even a new concept. After this process, it is possible to talk about the resulting abstraction as if it were a new object. However, it is just a new way to talk about the underlying objects [60, p. 161 ff].

Part II.

Operationalisation

Measuring Abstract Thinking Competence

Operationalisation

In order to be one step closer to the diagnostic question of *"Is there a deficit in first-year students' initial competence of abstract thinking?"*, an appropriate assessment tool has to be designed. The process of making competences measurable is called operationalisation [37].

Literature suggests that abstract thinking is a key competence, and thus, a prerequisite for studying computer science or related topics successfully [29, 53, 93]. At the same time, it is assumed that incoming students have deficits in the competence of abstract thinking [93]. In order to support these hypotheses, the diagnosis of the competence has to be carried out at the beginning of students' studies.

Thus, the assessment tool aims at the group of first-year students of the bachelor's programme in computer science and related topics. More concretely, the target date is at the beginning of their studies to exclude influence on the assessment results, e.g. by lectures at university. Due to the point in time, the skills that are necessary for solving the assessment are restricted to basic skills that have been acquired at school, as I want to assess students' competence and not their technical expertise. Thus, no item in the assessment should imply computer-science specific knowledge or skills, but represent a computer-science relevant way of thinking, which also meets the need to embed items in a real context. In addition, items need to have an appropriate level of difficulty. From a theoretical perspective, items must also be in accordance with the competence model. This leads to four criteria for all items:

1. Computer science knowledge must not be a prerequisite.
2. Items should represent the computer science mindset.
3. Items need to have an appropriate level of difficulty for first-year students.
4. Items must be in accordance with the competence model.

Additionally, literature and good didactial practice suggest several aspects that must be considered when developing the assessment tool. The overall assessment tool needs to cover each competence component several times. Different item types should be used in order to avoid a strong influence of other base competences, like reading ability. And each component of the competence model should be assessed by different types of tasks. Furthermore, the items should reflect different purposes and various perspectives[1] such as: artefact, number of artefacts and level [53].

From an organisational point of view, the time limit for the assessment tool is within two lecture hours (1.5 hours). This time should include a short explanation of the test

[1] see Section 1.2

and its purpose as well as concluding the test. Thus, the planned time limit for the assessment itself is 75 minutes. Its adequacy is evaluated in a pilot study (see Section 4.4).

In order to find an appropriate operationalisation for the competence of abstract thinking, one can pursue two approaches: looking for existing assessment tools or developing a new assessment tool, guided by the competence model. The results of both approaches have to be evaluated using the predefined requirements. Using an existing tool would be time efficient and thus is the first step I attempted. Chapter 3 shows the findings of my literature review and the evaluation of each approach. In the end, the previously derived evaluation criteria are picked up again to come to an overall résumé.

As the result of the literature review shows, none of the known approaches is appropriate for my intended setting. Consequently, a new assessment tool called Abstract Thinking Assessment (ATA) is developed in Chapter 4. For this purpose, performance indicators are defined based on the competence model. When designing the assessment tool, all questions are aligned with the competence model and reflect the computer science mindset. Question patterns found during the literature review are adapted and new items are derived from the competence model. The resulting collection of so called items is presented to experts. The ATA is evaluated in a pilot study with eleven representative participants (see Section 4.4). The insights gained are integrated into the final version of the ATA.

In parallel to the ATA, a rating rationale is established based on the competence model. This is used to categorise open format answers with the aim of objectively comparing them (see Chapter 5). For each item of the ATA, the rating rationale is specified to represent the individual item and its characteristics. The compilation of those instantiations forms the coding manual (see Appendix VII). It is enriched by exemplary answers of the pilot study.

3. Literature Review: Measuring Abstract Thinking Competence

After the competence of abstract thinking has been defined based on literature, I have again checked the literature for an appropriate assessment tool. For this purpose, I consider approaches that focus specifically on this competence. Moreover, approaches that focus on programming skills in introductory courses are considered.

3.1. Known Concepts for Operationalisation

In order to identify suitable tasks for the assessment, I analysed a number of different sources, e.g. computer science assessments on university level, school contests, and scientific publications. The emphasis of this search is on the evaluation of assessments with respect to the intended setting and requirements.

3.1.1. Computer Science Assessments at University Level

There exist serveral validated assessments for introduction to computer science, like the Foundational CS1 Assessment (FCS1) [86, 87] and the Second CS1 Assessment (SCS1) [70]. These tests focus solely on technical knowledge of computer science, like logical operators, arrays or recursion. Hence, they are not appropriate to be used at the very beginning of the students' studies nor do they measure the abstract thinking competence explicitly. Consequently, such approaches can be excluded from the review as they do not fulfil the first criterion.

3.1.2. School Contests

Another source to look at are school contests, since they often require little or no technical expertise. In the context of computer science, there exists the 'Bebras International Contest on Informatics and Computer Literacy', which originates from Lithuania and started in 2003. It is an annual contest (online test) for school students at all ages. Items focus on the "understanding of the principles, ideas and concepts that are involved in informatics systems" [24]. The aim of this international challenge is to draw students' interest to computer science, using realistic problems. However, they do not require any professional and computer science related knowledge. At the same time, attempts are made to reduce the often already present fear of coming into contact with computer science. As a result, students should be motivated to get engaged in computer science.

The German release of the Bebras Contest is called 'Informatik-Biber' [18][1]. All tests as well as their solutions are available online[2]. Each task of the assessment is characterised by three kinds of meta-information: targeted school years, level of difficulty, and a description why this task is related to computer science.

This assessment is a valuable source of items, but items most often focus on more than one key competence, as they were not designed for this purpose. The process described by Dagienė and Futschek [24] in the context of the Bebras contest provides good guidance on how to transfer real computer science problems into novice friendly ones.

3.1.3. Scientific Literature

In literature there are various approaches to measure the competence of abstract thinking in particular. In the next paragraphs I will present well-known approaches and discuss their degree of maturity as well as their applicability for my purpose.

Kurtz. A first attempt to measure abstraction processes was made by Barry Kurtz [55] in 1980. He used 15 questions on displaced volume, direct proportion, probability, inverse proportion, propositional logic, correlational reasoning, deductive logic, separation of variables, permutations, and combinations, to evaluate students' competence of abstract reasoning. When analysing the items, the processes required to solve the problems are often similar to the processes described in the competence model.

Students had to answer questions and afterwards they had to explain how they came up with their answer. Students knew that their explanations are way more relevant than to come up with the correct solution. The sample of the study consisted of 23 students in the course introduction to programming. The students' answers were coded by using 0 or 1 to show that a specific formal reasoning is present or missing. Based on the difficulty of the topic of an item and the code, the author classified students into three categories: *late concrete*, *early formal*, and *late formal*. For instance, if a student was not able to exhibit direct proportional reasoning or probabilistic reasoning he or she was classified at the *late concrete* level. At the end, most of the students were classified as *early formal*.

However, Kurtz mainly focusses on reasoning and not on abstract thinking. Moreover, the categories used for classification are not standardised nor validated with help of the data. Additionally, the author gives only little insights into his thoughts on the coding of students' answers. In conclusion, this approach is not reproducible. Additionally, it is not clear whether the tool is suitable for my target group.

[1] Supported by: German Informatics Society, Fraunhofer Information and Communication Technology Group, Max-Planck Institute for Informatics and the federal ministry of education and research.

[2] http://informatik-biber.de/

Or-Bach and Lavy. Another example for operationalising the competence of abstract thinking is the work of Or-Bach and Lavy [68]. The authors define the demonstration of fundamental object-oriented concepts, like classes, inheritance, abstract classes and methods as indicators for abstract thinking. In order to let students show their understanding of fundamental concepts, Or-Bach and Lavy designed one question. The question was answered by 33 students of the third year, who already took a course on object-oriented programming and another on object-oriented analysis and design. Based on the students' answers, the authors extracted a cognitive task analysis taxonomy reflecting the level of abstraction. The taxonomy has the following three stages (ascending order):

1. The abstract class includes attributes.
2. The abstract class includes attributes and implemented methods.
3. The abstract class includes attributes, implemented and abstract methods.

Most of the students achieved level one in the taxonomy. About one-third of the class was able to complete level two, and only four of 33 students accomplished the whole task and are thus classified as level three.

In my opinion, the authors define the stages of abstraction by the completeness of a class hierarchy and this does not reflect cognitive activities regarding abstract thinking. However, a modified version might be used as one item of the test instrument. Another weak point of this work is that the taxonomy does not reflect the decomposition process necessary to solve the task. Or-Bach and Lavy state that an important and difficult activity of the task is "the decision about the relevant" and that some students struggled with this part of the task. Compared to the developed competence model both aspects are important parts of abstract thinking and thus should be tested.

Bennedsen and Caspersen. In the work of Bennedsen and Caspersen [9], the competence of abstract thinking is measured by using the so-called "pendulum test". This test was developed by Inhelder and Piaget [49] to measure the cognitive development stage of a person. It uses an experimental set up of "a simple apparatus consisting of a string, which can be shortened or lengthened, and a set of varying weights". The experimental set-up is depicted in Figure 3.1.

Participants have to identify the factor that is most important for the speed of swing of the pendulum. Beside the length of the string and the weight, "other variables which at first might be considered relevant are the height of the release point and the force of the push given by the subject" [49]. The observation how a child deals with the problem is categorised into the four stages shown in Table 3.1.

Adey and Shayer [2] later repeated the experiment of Inhelder and Piaget with pupils at the age of 16. As only 40% of the pupils achieve one of the two highest levels defined by Inhelder and Piaget, Adey and Shayer revised the known stages and ended up with eight stages of cognitive development presented in Table 3.2. According to Bennedsen and Caspersen, the idea that abstract thinking can be mapped to these eight stages, can be derived from Adey and Shayer's theory of cognitive development [2].

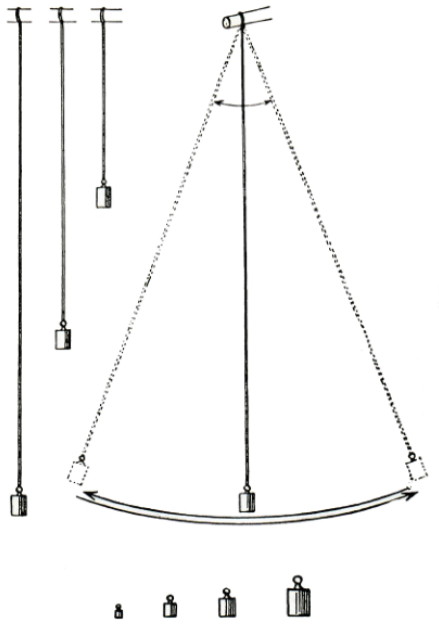


FIG. 3. The pendulum problem utilizes a simple apparatus consisting of a string, which can be shortened or lengthened, and a set of varying weights. The other variables which at first might be considered relevant are the height of the release point and the force of the push given by the subject.

Figure 3.1.: Experimental set-up of the pendulum test [49].

Stage	Description
I	Indifferentation between the subject's own actions and the motion of the pendulum
II	The appearance of serial ordering and correspondences but without separation of variables
III-A	Possible but not spontaneous separation of variables
III-B	The separation of variables and the exclusion of inoperant links

Table 3.1.: Stages of cognitive development observable in the "pendulum test" defined by [49, p. 67 ff].

Stage	Name
1	Pre-operational
2A	Early concrete
2A/2B	Mid concrete
2B	Late concrete
2B*	Concrete generalisation
3A	Early formal
3A/3B	Mature formal
3B	Formal generalisation

Table 3.2.: Stages of cognitive development after the refinement of [2].

A notable source of weakness is that the work does not necessarily measure the competence of abstract thinking, as already indicated by the wording. The cognitive development of a student is possibly not equal to the level of abstract thinking. This point is also discussed in the paper, as the authors did not find a significant correlation between cognitive development and programming ability, which experts would expect. They write: "The pendulum test measures the student's ability to control independent variables in a reasoning task." This ability does not match any component in the competence model of abstract thinking.

Hammer, Zehetmeier, Böttcher, and Thurner. Our research group developed a test focussing on competences, like abstract thinking, which are said to be necessary to study computer science or related topics sucessfully [38, 102]. This test does not require computer science specific knowledge. Instead, the competences are represented in tasks that require typical thinking processes of computer scientists, but no technical knowledge. This test is a first step towards measuring such competences.

Up to now, test results of over 750 students are available and were evaluated. As items of the test instrument can be mapped to several competences at the same time, it is not surprising that Cronbach's Alpha is 0.57. This means that the quality of the test is acceptable, but not good. Thus, I decided to focus this work only on one competence: abstract thinking.

Summary. Taken together, to my knowledge there exists no well-defined and validated approach for assessing the competence of abstract thinking. None of the published methods fulfils the criteria defined for this work. Many approaches require specific subject knowledge, which is not applicable for the diagnosis of the competence at the very beginning of students' studies. In that case, I would only test students' previous knowledge of a specific subject, but not their competence of abstract thinking. Many others have potential for improvement as they have known deficiencies and some are not reproducible due to a lack of information.

3.2. Question Patterns by Hazzan and Kramer

Hazzan and Kramer [43, 44] present a collection of question patterns as a first attempt to assess the competence of abstract thinking in the context of computer science. Each pattern needs to be concretely instantiated to be used in an assessment tool. The authors give an illustrative example of how to do this and explain in which way the question requires and enhances students' competence of abstract thinking.

In an iterative process, they propose collections of question patterns to lecturers and asked for their expert opinion. The last version of the collection [54] is presented in the following:

Pattern 1 Comparing two or more representations and ranking them according to the level of abstraction

Pattern 2 Categorising representations of different systems according to a student's choice of abstraction

Pattern 3 Describing a known system at different levels of abstraction

Pattern 4 Illustrating a representation that is more and one that is less specific than a given one

Pattern 5 Explaining a topic X to another person in two cases: The other person can see X or not. And explaining the considerations that were made to choose the explanation

Pattern 6 Explaining a topic X with one or more constraints. Afterwards, explaining the considerations that were made to choose the explanation

Pattern 7 Finding a metaphor or analogy to a given system

Pattern 8 Reflecting on the last software one developed

Most of the 31 experts, who completed the survey, are experienced in teaching computer science at research universities. Asking students to give two representations, where one is more and the other less abstract than a given one (pattern 4), achieved the highest agreement with respect to adequacy. Experts assessed the lowest suitability for pattern 1 that asks students to arrange representations according to the level of abstraction. The authors see a tendency of lecturers to favour question patterns where students need to construct abstractions for assessing the competence of abstract thinking, like pattern 4.

In summary, some question patterns suggested by Hazzan & Kramer show potential to be instantiated in a way referring to the computer science mindset without requiring any specific knowledge. Thus, they are considered during the development of items for the assessment tool.

4. Design of the Abstract Thinking Assessment (ATA)

As scientific literature does not provide a suitable assessment tool for the intended purpose and setting, this chapter describes in detail the development of an assessment tool called Abstract Thinking Assessment (ATA) based on the previously defined competence model (cf. Part I). Thereby, I close the gap of the approaches described before and tailor the assessment tool to the target group, as well as to the desired setting.

First, the decision about an appropriate assessment method has to be made. Potential methods are interview, questionnaire, observation, and test. All methods have their advantages and disadvantages, which are discussed in Section 4.1. The comparison leads to the decision to design a test.

Second, performance indicators are derived from the competence model, in order to create a test. These indicators, in combination with question patterns and specific situations, form items for the Abstract Thinking Assessment (ATA). In Section 4.2 the development of individual items for specific competence components is described in detail. Most of the items in the ATA are open answer format, as suggested by Rost [74]. This format allows to represent the real life context in a better way. The subsequent sections show how the items are assembled in the final Abstract Thinking Assessment (ATA) and how they cover the competence model.

4.1. Literature Review: Research Methods

In order to develop an assessment tool, it has to be decided which research method should be used. The aim of the assessment tool is to measures students' current level of the competence of abstract thinking. In general, there exist four techniques that are conceivable: observation, questionnaire, interview, and test [12].

During an observation, people are examined by researchers in naturally occurring situations without any interaction between the two. Although observation has unique features compared to all other methods, they are irrelevant in this setting. The first one is avoiding distortion of the behaviour as people might act in accordance with social desirability when asked directly. Another advantage is the presence of information about facial expression or gestures [12]. Nevertheless, both aspects are not of importance when assessing abstract thinking. Moreover, conducting and evaluating observations is cost- and time-consuming. Thus, it can also be excluded due to research economical reasons.

In contrast to observation, questionnaires are conducted in written form and by definition of Bortz and Döring [12] filled in remotely. As a consequence, it is not clear who completed the questionnaire, in which environment it was taken, and how much time the participant spent on it. The latter can be limited to a maximum when completed online. Nevertheless, answers are difficult to compare and possibly not representative.

Interviews address the problems that emerge from questionnaires. Each participant can be identified and the environment as well as the time frame can be controlled. In addition, interviews offer the possibility to react to the participants' answers and behaviour. For example, it is possible to ask in-depth questions or clarify questions when necessary. However, interviews are extremely time consuming and expensive [12]. This is contrary to the intended setting of the assessment: All approximately 150 students of the bachelor's programmes in computer science and scientific computing should be assessed within the first week of the semester. The timing is essential, as students should not be influenced by any lectures at university when measuring their initial level of competence. Interviewing this many students in one week is not feasible because of the research setting[1]. In conclusion, the assessment has to be conducted in written form due to research economical reasons and the given constraints. Thus, interviews are excluded as a research method.

The research method test meets all predefined criteria: First, a test is conducted in written form. Therefore, the implementation is feasible from a research economical perspective [12], and thus, fits the given requirements. Second, all students are assessed under equal conditions. This is necessary to obtain comparable results. Third, their identity can be verified. Additionally, using a test for the diagnosis of competences is also suggested by research, like [39]. Moreover, items in a test can be objectively corrected with a given assessment standard [12, 72]. Hence, the competence is objectively measured for a diagnostic purpose. Defining items, creating a test and developing an assessment standard (coding manual) is described hereafter.

4.2. Item Development

One of the most crucial points in a test design is selecting and developing items to measure the targeted competence. In this work I will approach this problem from two sides: on the one hand, the previous conceptualisation of the competence is a starting point for the development of new items, on the other hand, existing patterns are adapted to fit the requirements.

During the development of items, one has to take care that items always refer to a set of similar real-life situations, where the competence is required. This is a way to limit the risk of arbitrariness in the development process [40]. Additionally, I will follow the suggestion of Kramer [53] that "tests should examine different forms of abstraction, different levels of abstraction and different purposes for those abstractions".

[1]One researcher (me) available to conduct all interviews within a short period of time.

4.2.1. Operationalisation of *Commonalities and Differences*

The competence component of *Commonalities and Differences* is defined as follows: *Identifying commonalities in order to summarise them and determining differences to normalise them.* These facets can be operationalised by items that require:

- Identifying common and different characteristics
- Summarising and normalising characteristics
- Evaluating the level of summary and normalisation

Based on these requirements, three types of items have been developed: RANK AND REASON, MARK, and SUMMARISE AND PARAMETRISE.

Type RANK AND REASON

In order to assess the component, PATTERN 1: RANKING GIVEN REPRESENTATIONS described by Hazzan and Kramer is promising. Even if the participantes in the survey of Hazzan and Kramer [44] stated that ranking questions are not favoured compared to the creation of new abstractions, I decided to start with ranking of given abstractions as a warm-up section (cf. Figure 4.1).

Students need to evaluate commonalities and differences of the given depictions. Additionally, they need to rate the level of summary and normalisation in order to come to a solution. Every ranking is based on the number of details represented by the depiction. The higher the level of abstraction, the smaller the number of details represented by a depiction. Hence, more characteristics are summarised and normalised.

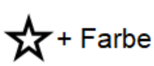
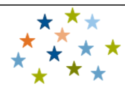
A modification of this type is pattern 3 described in Hazzan and Kramer [44]. Students need to describe a known system at different levels of abstraction. This kind of task is also reflected by an item of the test, where students have to characterise their smartphone on three different levels of abstraction, while focussing on the same aspect.

Additionally, I combined these ranking tasks with several reflection questions, where students need to argue on the overall structure that all rankings have in common (cf. Figure 4.2) or why they decided on the ranking they have chosen. This helps to get further insights in the criteria students use for those rankings, and whether they can explain what they where looking for.

Within the ranking items I differentiate between objects and processes to rank, as suggested in Section 1.2. In computer science, these two types of artefacts are handled differently and this should be reflected in the items. Furthermore, it is interesting to see, whether students deal better with one or the other.

Below are three illustrations (image or text). These can each be arranged in a specific order from **1: concrete** to **3: abstract**.

Example

Illustration			
Sequence	2	1	3

Bring all other groups of three in an order that matches the pattern above. Number . the pictures from **1: concrete** to **3: abstract**

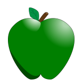
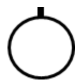
Illustration			
Sequence			

Illustration 1: `https://pixabay.com/images/id-307761/`

Illustration	Write a character x times, insert a line break. Then increase x according to the given rule and start writing again from the beginning.	`a`
		`aaaa`
		`aaaaaaaaa`
		...
			Write a character n^2 times, increase n by one and start over.

Sequence			

Figure 4.1.: Sample items of type RANK, focussing on different kinds artefacts.

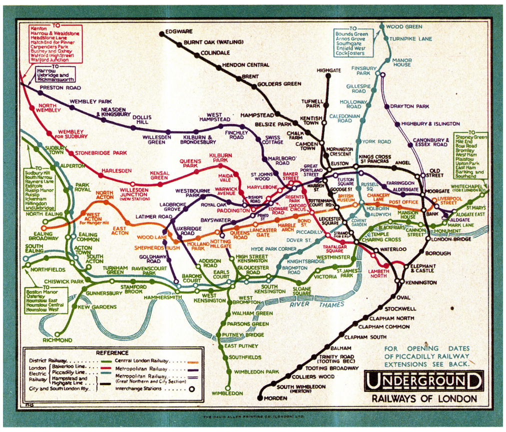
Describe briefly what constitutes the sequence of all four previous examples. *Not sufficient are statements, such as: The illustrations are always ordered from concrete to abstract.*

Figure 4.2.: Sample item of type REASON.

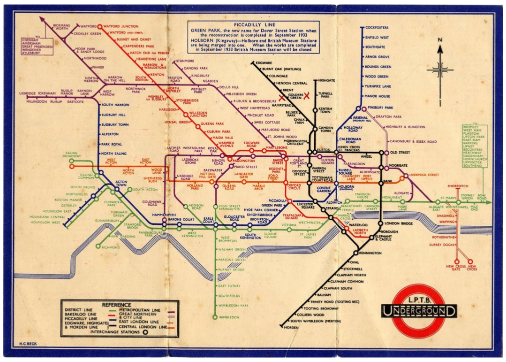
Type Mark

This type of question was inspired by the examples of visual abstraction described in the paper of Kramer [53]. The author depicts several abstractions that are represented visually. An example from every day life is the abstraction of every kind of public transport networks in form of maps. More precisely, Kramer presented the evolution of the London Underground Map between 1928 and 1931 as a concrete example of an abstraction.

In Figure 4.3 (a) the exact way of the London Underground in 1928 is projected onto a geographically accurate map of London. Showing all curves of the train lines as well as the bends of River Thames. Additionally, the relative distance between the stations is transferred. In contrast, the revised version (cf. Figure 4.3 (b)) of Henry Charles Beck (Harry Beck) from 1931 only contains horizontal, vertical and diagonal lines. The distances on the map are no longer proportional to the real geographical distances. These steps make the map more clear and more structured for the purpose of navigating along the underground railway network.



(a) London Underground 1928 by F. H. Stingemore [4].



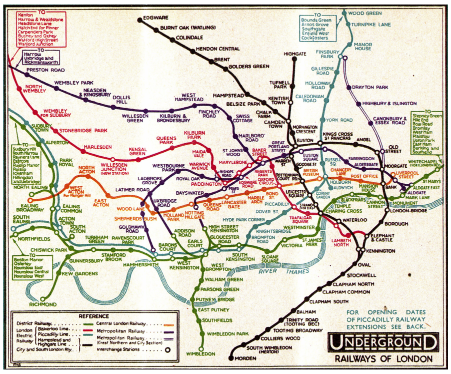
(b) London Underground 1931 by Henry Charles Beck [32].

Figure 4.3.: Evolution of the London Underground Map.

In this question type MARK, students need to evaluate what has changed and what remained unchanged after the abstract thinking process has happened. Section 1.2 argues that the number of artefacts at the beginning of an abstract thinking process should be considered. So far, items used several objects or processes as a starting point. In contrast, the question depicted in Figure 4.4 uses only one single entity. The abstract thinking process underlying this question is adapted from the example given by Kramer [53] and described above. Characteristics like distances or course of the tracks are listed, as well as some distractors such as interchanges. Students now need to check whether a characteristic has been modified or not.

London Underground
The number of stations on both maps should be considered identical!

aus dem Jahr 1928 [4].



aus dem Jahr 1931 [32].

Between 1928 and 1931, the London Underground map has been revised. See the table below for a list of characteristics included in the 1928 map.

Mark which characteristics remained unchanged and which have been changed during the revision.

Characteristic	unchanged	changed
Stations	☐	☐
Intersection points of the tracks (interchanges)	☐	☐
Location of the stations	☐	☐
Distances between the stations	☐	☐
Presentation of the environment	☐	☐
Course of the railway lines	☐	☐

Figure 4.4.: Sample item MARK.

Type Summarise and Parametrise

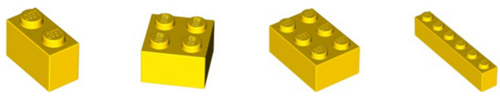
This type of question of the test represents typical problems in computer science that are transferred into novice friendly items. Derived from the competence model, students need to describe commonalities and differences among objects and processes. As Section 1.2 suggests there exists a difference between static and dynamic abstractions. Consequently, it is important to include both aspects into several items.

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. This includes finding an appropriate name for several objects and stating their common characteristics as attributes. It is important to keep in mind that in computer science, we separate characteristics which have an equivalent value for all objects from those who differ for each object.

Without any computer science specific knowledge, students cannot write proper classes in a specific language, but they are able to find an umbrella term or list common characteristics of given objects. Moreover, students can determine whether the manifestation of a characteristic is equivalent for all objects or not. An example is shown in Figure 4.5.

Summarise

1. Find a common and significant name for all objects of a subtask (umbrella term).
2. Name **4 characteristics** that all objects have in common.
3. Mark whether the values/manifestations of the characteristics are the same or different for all objects.



Source: `https://www.die-welt-der-kleinen.de/lego-einzelelemente-einzelteile-ersatzteile-de/LEGO-Grundsteine-de/`

Umbrella term		
	for all objects	
Characteristic	**same**	**different**
	☐	☐
	☐	☐
	☐	☐
	☐	☐

Figure 4.5.: Sample item summarise and parametrise.

In computer science, it is necessary to parametrise common characteristics. Whether students are able to recognise and parametrise characteristics is evaluated with items such as the one shown in Figure 4.7.

This item presents several elixirs and their effect to the students. The composition of each potion can be recorded by using a combination of `X`, `O` and `I`. The task is to describe the characteristic(s) shared by all potions that make chequered. One correct solution is that they have an even number of `X` in their description. In order to come up with this kind of solution, it is necessary to parametrise the number of `X`. In a next step the numbers of `X` for all elixirs making chequered are evaluated and parametrised to come up with the solution of even numbers of `X`.

When similar steps of a dynamic process have to be summarised and parametrised, indices are often the result of a parametrisation step. This often makes it more difficult for students, but is still a process in the sense of the competence component *commonalities and differences*. As Figure 4.6 depicts, this facet of the competence can be assessed without any computer science specific knowledge.

All questions of type SUMMARISE AND PARAMETRISE are deduced from the competence model and take a variation of the kind of artefact into account, as suggested in Section 1.2.

The following procedure is an extract of a computer program. Its goal is to systematically analyse the word `INFORMATIK` for a specific characteristic.

(1) `Compare I with K`	(6) `Compare M with R`
(2) `Compare N with I`	(7) `Compare A with O`
(3) `Compare F with T`	(8) `Compare T with F`
(4) `Compare O with A`	(9) `Compare I with N`
(5) `Compare R with M`	(10) `Compare K with I`

Formulate the procedure above in a way that it can be applicable to any words of length 10.

Figure 4.6.: Sample item SUMMARISE AND PARAMETRISE.

Miraculix has forgotten how to brew the potion that makes the Gauls so strong. He tries to remember and brews various potions. Some show amazing effects.

Miraculix writes down which potion causes which **effect**. For each potion he uses a combination of the letters **X, O and I**. With the help of this notation he can later retrace what effect a potion has.

https://www.comedix.de/lexikon/db/handzumgrus.php

Potion	Effect		
XIXXXOX	–	–	–
XXIIXXX	–	–	–
OXIXXXX	–	–	–
IIOXOII	–	–	makes glowing
IOIIIOI	–	–	makes glowing
OXXXXXO	–	–	makes glowing
OOOIOOO	–	makes weightless	makes glowing
OXOIXXO	–	makes weightless	–
OOOIXXO	makes chequered	makes weightless	–
OXXIXXO	makes chequered	makes weightless	makes glowing
XXXOXXX	makes chequered	–	makes glowing
XXXIXXX	makes chequered	–	makes glowing
OXXOXXO	makes chequered	–	makes glowing
XXOOOII	makes chequered	–	–
XXXIXOO	makes chequered	–	–
OXXXXXX	makes chequered	–	–

What do all the potions making *chequered* have in common?

Figure 4.7.: Sample item SUMMARISE AND PARAMETRISE.

4.2.2. Operationalisation of *Hide and Keep*

The competence component *Hide and Keep* is related to the process of choosing an appropriate level of abstraction and selecting characteristics for a certain purpose. This is what should be represented by all items derived from this component. The items provide a purpose and possibly several objects or processes. Students then have to choose which characteristic is relevant and which is not.

The problem with this type of question is that in many cases, it leads to a huge space of possible solutions, as one can most often find both, arguments for and against a selected subset. A typical example would be deciding which personal information should be stored in an online shop. One can create scenarios where most information suggested is relevant. A possibility to deal with this problem is that students need to write down their arguments. Then, however, the item focusses on the ability to argue instead of the abstract thinking competence. Another edition could state the purpose in much more detail and make more restrictions, but this makes the items almost trivial. Thus, this kind of problem is postponed to be discussed in the lecture.

Nevertheless, through the use of items in which students make a decision whether a characteristic or a set of characteristics is sufficient to differentiate objects of different classes, the solution space can be narrowed in a way that the solution is not trivial. Consequently, I decided to formulate several items of this type (cf. Figure 4.8). In addition, I designed an item where students do the inverse process. They are asked to reproduce the purpose from a given abstraction and the input of the abstract thinking process.

On many websites you see if a password is weak, medium or strong. You want to find out which criteria underlie the process of a specific homepage. To do this, you enter different passwords and note the displayed status. The result of this enquiry is shown in the following table.

Passwort	Status
Servus	weak
Baum36	weak
T?o8	weak
$5&-!?	weak
SteffisPassworT	medium
!nformatik	medium
urlaubindien2013	medium
studium	medium
20!7-!0	medium
B3!$p!3l	strong
§Passwort&	strong
TH3OPW!	strong
§4%68?9!a	strong
6PassWort	strong

What are the components that all various pass phrases consist of?

Which components and other characteristics distinguish `weak` from `medium` pass phrases?

Which components and other characteristics distinguish `strong` from `medium` pass phrases?

Figure 4.8.: Sample item HIDE AND KEEP.

4.2.3. Operationalisation of *Expand*

Expand is the third and last competence component to be assessed. It is defined as *revealing relationships between items or processes and building new components and concepts*. In order to evaluate this part of abstract thinking, concrete entities or processes are presented to the students. The objects are constructed, based on a rule set that is not described. In contrast to *Commonalities and Differences* the introduction of parameters is no longer sufficient. New concepts like counting, substitution or loops are needed. Just to clarify: This does not mean that students need to know loops in a certain programming language. However, they have to come up with some idea of how to repeat steps and maybe keep track of the number of steps done so far.

In order to assess this, students need to reveal and describe the relationships between instances in a collection of similar entities or processes. This also includes building hypotheses about the underlying structure and verifying those using the given examples. Examples for this type of item are shown in Figures 4.9 and 4.10. In order to foster the reflection process, students are sometimes asked to build an instance by applying the proposed rule set.

The following procedure is an extract of a computer program. Its goal is to systematically analyse the word `INFORMATIK` for a specific characteristic.

(1) `Compare I with K`
(2) `Compare N with I`
(3) `Compare F with T`
(4) `Compare O with A`
(5) `Compare R with M`
(6) `Compare M with R`
(7) `Compare A with O`
(8) `Compare T with F`
(9) `Compare I with N`
(10) `Compare K with I`

How would the procedure look like to be applicable to any word of even length. Define this procedure.

Figure 4.9.: Sample item EXPAND.

In contrast to the item depicted in Figure 4.6, it is no longer sufficient to introduce a parameter to solve the task. Students need to come up with the idea of a repeating structure that influences the parameter in every run. If and only if they are able to deal with the variable length of the word.

In the following, four messages were encoded. Below you can see the original message and the code. But the encodings exhibit gaps (empty rows).
Complete the gaps in the code.

Message								Code
X	X	O	O	O	X	X	→	bxcobx
X	O	O	O	O	O	X	→	axeoax
O	O	I	I	I	I	O	→	
X	O	X	I	X	O	X	→	axaoaxaiaxaoax
X	X	O	O	O	X	X	→	bxcobx

Figure 4.10.: Sample item EXPAND. Adapted from [18].

In a previous study presented at EDUCON'17 [102], I conducted several think-aloud sessions [71] and decoding interviews [69] with students and researchers. The aim was to analyse their mental processes and to identify the competences necessary to solve the problem.

One question that was studied is the one depicted in Figure 4.10, but in a closed format. After the interviews, the answers of the participants where summarised for those solving the problem and for those who were not able to. The result of this analysis is again presented in Table 4.1 and Table 4.2. It has to be pointed out that participants who solved the problem describe a flash of inspiration in their thinking process. This is when they come up with a new relationship, component or concept. Participants that came to an incorrect solution did not depict any similar event. Moreover, they did not sufficiently check their hypotheses about the underlying rule set.

Items presented in this chapter are just an excerpt from the final Abstract Thinking Assessment (ATA) to illustrate the design process. The development of all other items followed this design process, where at first, the competence model is analysed to derive specific processes of the competence component. Afterwards, typical problems related to computer science are collected that represent the specified process. Subsequently, the problems are rephrased to fit students' previous knowledge. The resulting problems then form test items. The final ATA (in German) can be found in the Appendix – Part VI.

#	Step
1	Picture and code contain the same characters (x, o, i)
2	All other characters like a and b in the code must carry some information
3	Both X in the picture have the same color and might be a unit
4	Try find correlation between XX __ __ and code `bxcobx` as well as __ __ XX and code `bxcobx`
5	Sequence XX twice in the picture and bx twice in the code
6	From XX follows bx
7	Test hypothesis with last row → correct
8	Both bx as well as XX can be 'removed' for further work
9	Remains OOO and `co` for the first line
10	b → 2, c → 3, a → 1
11	Character in front of x, o, i represents the amount of signs X, O, I in the code until it changes and it matches the number to the character in the alphabet on the position equal to the amount
12	Code consists of tuple (amount of same character, the character itself)
13	Test hypothesis with all examples → works
14	Apply rule to third and missing row
15	Check if result is a possible answer

Table 4.1.: Results of the think-aloud process that leads to the correct result [102].

#	Step
1	Picture has 7 columns whereas the code has 6 and more characters
2	Match of one sign in the picture to a character in the code is not possible
3	Row 1 and 5 are identical
4	Difference between code in row 1 and 2, 2 has no b
5	Conclusion XX equals b
6	Correlation discovered that 2X → b
7	What does the o in the code mean?
8	What does the O in the picture stand for?
9	Correlation XX → b, X → a
10	Solution cannot start with b or a as the picture shows no X
11	Elimination process on the answer possibilities leaves only one choice oociaio (wrong answer)

Table 4.2.: Results of the think-aloud process that leads to a false result [102].

4.3. Composition

The items are arranged in a mixed way due to the component they address. However, items of the same type and referring to a comparable situation are presented as one block with one problem description at the beginning. This description is sometimes enriched by an example that also includes its solution. Thus, students know what kind of answer is expected from them. In other problems, the response field shows the beginning of the sentence anticipated. The size of the response fields is appropriate to the amount of text expected. All this should help to make instructions more clear and thus to minimise bias [74].

An overview of the component each item addresses is depicted in Table 4.3. The order of items in the table is the same as the order in which items appear in the ATA. The final ATA is presented in Part VI of the appendix.

4.4. Evaluation

At the beginning of this Part II, the aim and the target group of the assessment tool have been presented and requirements for the assessment and all items have been deduced. After the development of the items and the composition of the assessment, the implementation of these requirements is reviewed.

In general, different item types should be established in the assessment, which has been fulfilled by using a mixture of textual and graphical descriptions. Additionally, the items use various purposes, like finding the way through the London railway network or building an abstraction in order to automate a process.

As the example items in Section 4.2 show, all of them meet the previously defined requirements. This also applies to every item in the test that is presented in the Appendix Part VI. None of the items requires any computer-science specific knowledge. Instead, items are derived from typical problems in order to represent the characteristic mindset. The problems used often originate from our first-semester introduction to software development. In addition, problems from the 'Informatik Biber' [18] have been examined, which are aiming at secondary education (grades 5 to 12/13) and offer different levels of difficulty. The items focus on the "understanding of the principles, ideas and concepts that are involved in informatic systems" [24]. Thus, all items have an appropriate level of difficulty and represent the computer science mindset. Finally, each item can be assigned to a specific competence component of the proposed model, as shown in Table 4.3.

Item	**Communalities & Differences**	**Hide & Keep**	**Expand**
Rank, analyse and reason			
Rank Apple	X		
Rank Seal	X		
Rank Vending Machine	X		
Rank Typing	X		
Reflect upon Rankings	X		
Generalise Rankings	X		
Rank Ticket	X		
Justify Ticket	X		
Rank Species	X		
Justify Species	X		
Summarise			
Name Vehicle	X		
Characterise Vehicle (1)	X		
Characterise Vehicle (2)	X		
Characterise Vehicle (3)	X		
Characterise Vehicle (4)	X		
Name Brick	X		
Characterise Brick (1)	X		
Evaluate Characteristics Brick (1)	X		
Characterise Brick (2)	X		
Evaluate Characteristics Brick (2)	X		
Characterise Brick (3)	X		
Evaluate Characteristics Brick (3)	X		
Characterise Brick (4)	X		
Evaluate Characteristics Brick (4)	X		
London Railway Network			
Stations	X		
Intersections	X		
Location of stations	X		
Distances	X		
Representation	X		
Course	X		
Justify Tube Map		X	
Fragmentary Codes			
Easy Code			X
Medium Code			X
Biber Code			X
Music Code			X
Faulty Codes			
Mark and Correct Binary to Alpha			X
Mark and Correct Counting Code			X
Rule for Counting Code			X
Mark and Correct Caesar Code			X
Rule for Caesar Code			X
Mark and Correct Digital Code			X
Rule for Digital Code			X

Item	Communalities & Differences	Hide & Keep	Expand
Machines			
Function Sorting			X
Function Set			X
On the Grid			
Grid Moved Diagonal			X
Grid Triangle			X
Grid Chess			X
Grid Tree			X
Pass phrase Security Status			
Pass phrase Components		X	
Differences weak – medium		X	
Differences medium – strong		X	
Rule weak			X
Rule medium			X
Rule strong			X
Potions			
Potion chequered commonalities	X		
Potion weightless commonalities	X		
Potion luminous differences		X	
Automation			
Palindrome constant length	X		
Palindrome various length			X
Smartphone			
Smartphone – 3 abstraction levels	X		
Justify Smartphone Levels	X		

Table 4.3.: Overview of which question is related to which competence component.

In order to evaluate the time limit, the comprehensibility and the difficulty of the Abstract Thinking Assessment (ATA), a pilot study has been conducted. The population was a representative group of 11 people consisting of potential students (last year of Gymnasium), enrolled students and professionals from the STEM field. Participants had a time limit of 75 minutes to finish the ATA. In Table 4.4 an overview of the participants is depicted.

In a performance test with time limit, there is a risk that the last items in the test will often not be answered. Thus, every participant of first run, got a different order of the items in the individual ATA. This helps to reduce the effect that the arrangement of he items might have on the solution frequency.

In summary, the following insights were gained by the pilot study using ATA version 1 and were used to revise the ATA.

Solution rate: The evaluation showed that some of the RANKING items are rather easy for the participants. Nevertheless, these items remain in the ATA to function as 'warm-up' questions.

Item selection: Two items contained an extensive amount of text, which several participants found very difficult. Especially students with difficulties in reading strug-

Version	Current Level (Education)	Highest degree	Specialisation
Version 1	Gymnasium 12th grade	-	-
Version 1	Gymnasium 12th grade	-	-
Version 1	Bachelor	Fachabitur	Economics
Version 1	Bachelor	Abitur	MINT
Version 1	Master	Bachelor	Information systems & management
Version 1	Master	Bachelor	Computer science
Version 1	Non-IT Professional	Abitur	-
Version 2	Gymnasium 12th grade	-	-
Version 2	Non-CS Professional	Abitur	-
Version 2	CS Professional	Bachelor	Computer science
Version 2	CS Professional	Bachelor	Computer science

Table 4.4.: Biographic characteristics of the participants of the pilot study.

gled with these items. Thus, those two items were replaced by one item with less text, but representing a similar situation and problem. This item has been revised by interviews with experts and students.

Arrangement of items: Participants reported that changing item type or competence component frequently, makes it difficult for the students to stay focussed. Additionally, this makes it necessary to understand the task type again and again. Thus, items of the same type, focussing on the same competence component are grouped together. Similar items are presented one after the other. Whenever possible, the initial instruction is just presented once at the beginning of the group of items.

Time limit: High performing participants managed to complete the ATA in time. Thus, neither the number of items nor the time limit need to be adjusted. The time limit will be set to 75 minutes for the following studies. Thus, the organisational requirement *time* can be met.

Wording: When asking students to name similar characteristics the term 'Eigenschaft' (feature) has been used in version one. Several participants noted that the term 'Eigenschaft' is misleading and suggested 'Merkmal' (characteristic) instead. Consequently, the wording was adapted in all items, like the one depicted in Figure 4.5.

All these findings are used to create a second version of the ATA. Subsequently, a second pilot study was conducted to evaluate whether the changes increased readability and to check whether the ATA is still feasible in the predetermined time. Participants still managed to complete version 2 of the ATA within the time limit of 75 minutes and did not suggest further changes. Thus, the second version of the ATA is used in the following studies and forms the final version.

5. Rating Guidelines and Scoring

In order to be able to judge students' answers by some kind of standard, a comprehensive and generic rating rationale is developed (see Section 5.1). It is a blueprint to evaluate items that assess the competence of abstract thinking. From this, a coding manual has been derived to guide the rater through a qualitative rating process (see Appendix Part VII). A detailed description of each item complemented by sample answers helps to obtain unbiased evaluations. The advantage of using a rating rationale is that answers are objectively categorised. Thus, answers are comparable [12]. Afterwards, the rating process is presented in Section 5.2. At the end, codes in the rating rationale are assigned to scores for the data analysis. During this scoring process, several codes can map to one score (see Section 5.3).

5.1. Rating Rationale

I identified several dimensions for looking at students' answers, like correctness, level of abstraction or quality of reasoning. Moreover, completeness seems to be important, as this represents whether students thought of all aspects that are relevant. Descriptive characteristics like kind of presentation (prose text, graphic or formula) and kind of characteristic have been identified. These are documented for several items, to gain further insights that influence the teaching concept. As the quality of students reasoning is a field of research on its own, it is not part of this work. All other dimension seem promising for the evaluation of abstract thinking competence in the context of computer science.

Accuracy and completeness form the basis for a solution. Thus, both are taken together. It is not sufficient, to give accurate answers, if these are not complete. On the other hand, if answers represent all relevant aspects, but for example depict a wrong relation, the answers cannot be evaluated correctly. Thus, they are comprised into one scale called *Correctness*.

For the scale *Correctness*, the categories *correct* and *false* are defined. There must be a hard border between these two categories. Therefore I rate only answers as *correct*, which are entirely accurate and complete. Partly correct or wrong answers are rated as *false*.

The *Level of Abstraction* shows to which degree a solution represents a set of different objects. This, in essence, answers the question about students' competence of abstract thinking. The *Level of Abstraction* is evaluated independently of the *Correctness*.

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 [19, 44, 58, 82, 95]. Thus, it seems obvious that I have to find an adequate measure for the level of abstraction students achieved. Based on literature, two opposed categories *concrete* and *generic* can definitely be defined [45, 62, 67].

Teaching experience shows that students tend to focus on given objects while building abstractions, which restricts the range of possible applications of the solution. However, this is more than concrete, but not the most generic solution. Thus, I introduced a third category *specific* between the poles *concrete* and *generic*. *Concrete* answers are characterised by a practical and stepwise explanation. Moreover, 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, the answers are *specific*. In the case that the described rule set can be applied beyond the given examples, the answers are rated as *generic*.

All scales are completed by the rating category *empty* for an empty answer or answers that are crossed out. In addition, the category *unconnected* is used for answers that do not show any relation to the problem statement.

A detailed description of the comprehensive and generic rating rationale for the two scales *Correctness* and *Level of Abstraction* can be found in Table 5.1. In case that several codes apply, the lowest is chosen. The application of the rating rationale to evaluate students' answers is exemplarily shown in Section 5.2.

Based on the coding of the *Level of Abstraction* a second, dichotomous scale can be derived. This scale specifies whether the solution represents an abstraction of the given examples or not, and thus, whether the solution indicates a sufficiently high level of abstraction or not. Consequently, the scale is called *Abstract Thinking Competence*.

Experts in software engineering need to build abstractions on a high level of abstraction, which are also accurate and complete. As abstract thinking competence and correctness are assessed independently in the coding step, this relation is not reflected by one of the scales, yet. Thus, it is interesting to compute a scale representing the relation between *Correctness* and *Abstract Thinking Competence* representing *Professionality*.

In summary, the following four scales have been defined:

Correctness Accuracy and completeness	***Level of Abstraction*** Degree of abstraction
Abstract Thinking Competence Indication for a sufficiently high level of abstraction	***Professionality*** Accuracy and completeness as well as an indication for a sufficiently high level of abstraction

Correctness		
Empty	The response field is ... – empty. – crossed out completely.	999
Unconnected	The answer is ... – unconnected or out of context. – a buzzword. – a paraphrase of the problem. – unreadable.	99
False	The answer is ... – deficient from a professional perspective. – partly deficient from a professional perspective. – unclear in its formulation.	0
Correct	The answer is ... – accurate and complete from a professional perspective.	1
Level of Abstraction		
Empty	The response field is ... – empty. – crossed out completely.	999
Unconnected	The answer is ... – unconnected or out of context. – a buzzword. – a paraphrase of the problem. – unreadable.	99
Concrete	The answer ... – describes the given examples. – uses everyday or well-known terms. – describes actions step by step. – uses several cases that are specific for a single example.	0
Specific	The answer ... – depicts a rule or a rule set, which exclusively describes the given examples.	1
Generic	The answer ... – depicts a rule or a rule set, which describes the given examples and goes beyond that.	2

Table 5.1.: Rating rationale for items designed to measure the competence abstract thinking.

5.2. Rating Process

A majority of 40 out of 61 questions of the Abstract Thinking Assessment (ATA) are open format questions. Answers are given in a written form on paper. All filled assessments are rated by one rater using the coding manual (see Appendix VII). A second rater classifies answers of 10% of the filled assessments independently. The degree of agreement of both ratings is a basis for the evaluation of the objectivity of the assessment and its rating. The original test sheets including answers, notes and sketches are accessible to both raters. None of the answers are evaluated automatically. In the following, the rating process is illustrated in detail for a choice of items.

Rank, analyse and reason: Justify Ticket

This item represents the competence component *Commonalities and Differences*. Students need to rank several objects as illustrated in Figure 4.1. In a next step, they are asked to give a justification for the sequence they have chosen. This item is open format and represents a typical process of abstract thinking. Consequently, in addition to *correctness*, the *level of abstraction* is rated using the coding manual. The results of the rating process for several answers are depicted in Table 5.2.

Sample Answer	**Correctness**	**Level**
Peter had to count the money first, but in concrete terms he has the same coins as Mona.	False	Unconnected
In 1 it is exactly described which coins are used, in 2 only the total amount. In 3, the process is described only generally and 4 is a symbol.	False	Concrete
Number 1 contains the most information, whereas number 4 offers much scope for interpretation.	False	Specific
I go from the general to the detailed depiction.	False	Generic
1. Complicated process 2. Same process simplified 3. Process generalised & reduced 4. Abstract description reduced to on image	Correct	Concrete
The largest information content is in (1), then (2) $rightarrow$ In (2) you do not know exactly how the change is split. (3) is just a description of the activity and (4) a representation with no details.	Correct	Specific
The process of ticket purchase is described with fewer and fewer details. The level of detail is the sorting criterion. The less details, the more abstract.	Correct	Generic

Table 5.2.: Rating of different sample answers to item Justify Ticket, according to the two scales. Translated from German into English while trying to maintain sloppiness in wording.

Pass phrase Security Status: Differences medium – strong

A sample item shown in Figure 4.8 represents items of the *Hide and Keep* competence component. In the second and third item, students are asked to describe a characteristic or a set of characteristics that is sufficient to distinguish different classes of objects. The application of the coding manual to the third task, asking for criteria to differentiate pass phrases of medium and strong status, is shown in Table 5.3.

Sample Answer	**Correctness**	**Level**
Numbers used as letters, special characters mixed with these 'words'	False	Concrete
At least 1 number or 1 special character for strong. 8 digits	False	Concrete
strong: consists of at least 3 different components, weak: consists of not more than 2 different components	False	Specific
3/4 components used for strong, 2/4 for medium	False	Specific
At least 3/4 of the components are used for strong. Medium pass phrases, however, use at most 2 of the 4 components.	Correct	Specific
Number of used character types	Correct	Generic

Table 5.3.: Rating of different sample answers to item Differences medium – strong, according to the two scales. Translated from German into English while trying to maintain sloppiness in wording.

On the Grid: Grid Chess

A typical item derived from the competence component *Expand* is shown in Figure 5.1. Students need to reveal the relationships between the 'x' on the grid, and then describe the rule set of their creation. Sample answers and their ratings are presented in Table 5.4.

Find and formulate a rule that describes when an 'x' is placed on the following grid. Each cell can be clearly identified by a row and a column number. The rule should be as general as possible.

		Column					Rule
		1	2	3	4	5	
	1	x		x		x	
R	2		x		x		
o	3	x		x		x	
w	4		x		x		
	5	x		x		x	

Figure 5.1.: Sample item `Expand` as it appears in the test.

Sample Answer	**Correctness**	**Level**
An x is printed with one free box each. If the row number is even, the free box comes first.	False	Specific
An x in each cell whose column number is equal to the row number and in each cell of a row whose column number is 2 smaller or larger than its row number.	False	Specific
x is set if row number is odd if numbers are divisible only by themselves. If row number is even, x is set for numbers that are divisible by 2.	False	Generic
If the number is even, column numbers consist only of even numbers and vice versa.	False	Generic
An 'x' is set if row % 2 = 0.	False	Generic
For all even row numbers, an 'x' is set at column 2 and 4. For all odd row numbers, an 'x' is set at 1, 3, 5.	Correct	Concrete
An x is set if column number + row number is an even number.	Correct	Generic
An x is set if row number odd and column number odd too or if row number even and column number even too.	Correct	Generic
An 'x' is set if row number % 2 = column number % 2.	Correct	Generic

Table 5.4.: Rating of different sample answers to item Grid Chess, according to the two scales. Translated from German into English while trying to maintain sloppiness in wording.

5.3. Scoring process

For the analysis, the answers are scored after the rating. The mapping of codes and scores for all scales is depicted in Table 5.5. As the Abstract Thinking Assessment (ATA) is a performance test, empty and unconnected answers on scale *Correctness* show the same low performance as wrong answers. This is reflected by the scoring. Empty, unconnected and false answers receive the same score 0. Only correct answers are valued with a score of 1.

For scale *Level of Abstraction*, empty and unconnected answers are marked with score 0 as it is not possible to make a statement about the abstraction level present. All other codes are increased by 1 as each score should represent a single level. Thus, a scale from 0 to 3 with a step size of 1 emerges.

Based on the *Level of Abstraction* a dichotomous scale called *Abstract Thinking Competence* with values 0 and 1 has been derived. Answers showing the competence of abstract thinking are scored with 1, whereas score 0 represents that no abstract thinking competence is present in the solution.

The mapping rules for the derivation of *Abstract Thinking Competence* is as follows: Code 1 (specific) on *Level of Abstraction* is assigned to answers depicting an abstraction of the given examples, but they still contain specifics that limit the abstraction to the given examples. Solutions with code 2 (generic) do not contain any restrictions of the abstraction with respect to the examples. Hence, both codes stand for an abstraction. Consequently, answers with code 1 and 2 assigned on *Level of Abstraction* are scored 1 on scale *Abstract Thinking Competence*. Answers with code 0 (concrete) on *Level of Abstraction* solely describe the examples one by one or use numerous cases to discriminate each example individually. The number of cases is in this case often equal to the number of examples. This kind of solution does not represent an abstraction, the same applies to empty or unconnected answers. Thus, they are scored 0 on scale *Abstract Thinking Competence*.

For the new scale *Professionality* only answers that are correct and that show a sufficiently high level of abstraction are scored 1. All others do not depict a professional solution and are consequently scored 0.

Label	**Code**	**Score**
Correctness		
Empty	999	0
Unconnected	99	0
False	0	0
Correct	1	1
Level of Abstraction		
Empty	999	0
Unconnected	99	0
Concrete	0	1
Specific	1	2
Generic	2	3

Label	**Code**	**Score**
Level of Abstraction		*Abstract Thinking Competence*
Empty	999	0
Unconnected	99	0
Concrete	0	0
Specific	1	1
Generic	2	1

Score	**Score**	**Score**
Correctness	*Abstract Thinking Competence*	*Professionality*
0	0	0
0	1	0
1	0	0
1	1	1

Table 5.5.: Scoring scheme for the rated answers.

Part III.

Studies and Analysis

Studies and Analysis

After the definition of the competence model for abstract thinking, the Abstract Thinking Assessment (ATA) has been introduced. Data collected using the ATA reflect correctness in the sense of accuracy and completeness as well as the level of abstraction of students' answers. Analysing the data collected using the ATA makes it possible to test commonly known hypotheses, for example on the relation between the competence of abstract thinking and the acquisition of professional skills in computer science, or the lack in abstract thinking competence among first-year students.

The ATA was applied in two studies: the *Main study* and the *Post study*. The population that forms the basis for these studies are 151 first-year students starting in winter semester 2017/18 in the bachelor's programmes in computer science and scientific computing at the Munich University of Applied Sciences. Thus, the *Main study* provides insights into first-year students' initial level of abstract thinking competence when entering university. In order to evaluate students' progress, a *Post study* has been conducted at the beginning of the second semester.

At my institution, first-year students are split into sub-cohorts taught in parallel. Consequently, the *Main study* took place in three sessions to fit the individual schedules of the sub-cohorts. These sessions were carried out under comparable conditions: 75 minutes editing time, same lecture hall, clock present in all sessions, time check after 30 minutes and 15 minutes before submission. In order to motivate students and to reward participation, all students received a joker for an arbitrary lab-assignment. This means, that they can skip one lab-assignment without consequences for their admission to the final exam. Overall, $n = 134$ students participated in the study. 129 students were 'real' first-year students. The other five students are in their third semester, retaking courses of the first semester that they did not pass before.

During the *Post study*, students participated in one session. The conditions were the same as the ones for the *Main study*. This time, students received an individual evaluation of their results in the ATA as a bonus. Additionally, five power banks have been raffled among the participants. All in all, 45 students from all three sub-cohorts attended the *Post study*.

This part of the thesis focusses on the analysis of the data collected in the two studies. Based on the data of the *Main study*, common quality criteria are evaluated for the ATA in Chapter 6. Subsequently, students' results are analysed in Chapter 7 to get an overview of first-year students level of proficiency. At the end, the data collected are used to verify well-known hypotheses from literature and one further hypothesis that arose during the coding process. The findings are presented in Chapter 8.

6. Evaluation of the Quality of the Abstract Thinking Assessment (ATA)

The quality of an assessment is defined by its items and their composition. A good quality can be achieved by *design* and can be verified with help of *empirical data* [12, 17]. The main quality criteria for assessments are:

- **Validity:** Correspondence to the underlying model.
- **Objectivity:** Independence of the investigator.
- **Reliability:** Accuracy of the measurement.
- **Item difficulty:** Variety of levels of proficiency.

In the following four sections, methods to evaluate the quality of the ATA are presented and applied. The data underlying all analyses originates from the *Main Study*.

6.1. Validity

The quality criterion validity describes the degree to which an instrument measures what it claims to. There exist three types of validity:

- Content validity
- Criterion validity
- Construct validity

Content validity is achieved by a deliberate and thorough *design* of the assessment. The fact that the items of the ATA are derived from a well-researched competence model contributes to its content validity.

After *data* have been collected, criterion validity can be examined by comparing the results with another criterion that should correlate with the outcome of the assessment. Subsection 8.3 shows that students' competence of abstract thinking measured using the ATA shows an impact on the acquisition of programming skills. As this corresponds to known hypotheses, criterion validity is assumed.

Construct validity is evaluated by the comparison of results of the new assessment with the results of an established one measuring the same. As the ATA is the first attempt to measure the competence of abstract thinking, a comparison is not possible at this time. Thus, complete validity cannot be proven.

6.2. Objectivity

The objectivity of an assessment describes the independence of the investigator. A high objectivity can already be achieved in the *design* phase by providing standardised instructions for the following three phases [17]:

– Application
– Evaluation
– Interpretation

An objective application is achieved by comparable conditions like time limit, lecture hall and assistance during the assessment. In all three sessions of the *Main study* they were exactly the same for all students. Additionally, these conditions have been noted in this work to be available for further applications of the ATA (see introduction of Part III).

In order to achieve an objective evaluation and interpretation, guidance for the investigator must be provided. In case of the ATA, a rating rationale has been developed, which serves as a blueprint for the evaluation of the competence of abstract thinking. From this, a coding manual has been derived to guide the rater through the evaluation of each item. Moreover, the application of the coding manual is exemplarily shown. In order to limit the scope for interpretation, the scoring is additionally presented (see Chapter 5). Furthermore, the analysis of the results presented in this part of the thesis can be considered as a guide to interpret the results of the ATA.

The objectivity of the evaluation can also be verified *empirically*. A well-known measurement for the agreement of independent raters is `Cohen's Kappa`, representing the so-called interrater reliability. Cohen's Kappa represents the agreement of two raters beyond what would be expected by chance [56]. Perfect agreement of both raters is represented by $\kappa = 1.0$. Fleiss & Cohen [31] suggest a value between 0.60 and 0.75 for a good agreement. Values above 0.75 represent an almost perfect match.

To validate the objectivity of the ATA, a second person rated 10% of the 134 assessments, which were randomly drawn, independently of the first rater. In order to identify misunderstandings or problems using the coding manual, a rater training took place at the beginning. The agreement of both raters κ has then been calculated for each item on the basis of 14 assessments. Subsequently, the average Cohen's Kappa κ has been computed for the two scales *Correctness* and *Level of Abstraction*. According to known interpretations, the interrater reliability, which is shown in Table 6.1, is almost perfect for the ATA and its coding manual.

Scale	**Cohen's Kappa κ**
Correctness	0.87
Level of Abstraction	0.92

Table 6.1.: Average interrater reliability (Cohen's Kappa κ) for the two scales.

6.3. Reliability

Generally speaking, reliability describes the degree of accuracy of the measurement. The reliability of an assessment is examined using *empirical data*. There exist three methods to investigate the reliability and to calculate the reliability coefficient. Depending on the setting of the study one can compute [17]:

- Retest reliability
- Parallel test reliability
- Internal consistency

With the help of retest reliability, the stability of the results over time is investigated. The assessment is filled in by the same group of participants at two different points in time. Although students have been assessed using the ATA at different times, this measurement is not appropriate, as an increase in the competence is expected between *Main* and *Post study*.

In case of parallel test reliability, two different assessments measuring the same construct are required. It is expected that both assessments lead to similar results. So far, there exists no other assessment for the competence of abstract thinking. Thus, the application of this method is not possible yet.

The only reliability measurement that requires just one study and one assessment is internal consistency. Consequently, this is the measurement of choice in my case. Internal consistency is reported using the reliability coefficient *Cronbach's Alpha* (α). The coefficient is calculated on a scale level and is a modification of parallel test reliability. For the calculation, each item is interpreted as an assessment on its own and correlated to all other items in the assessment. Thus, *Cronbach's Alpha* represents the revised average correlation between the items [12, 17]. There exist several suggestions for the interpretation of Cronbach's Alpha. Bland and Altman [10] as well as Tavakol and Dennick [85] propose a value ranging from 0.70 to 0.95 to be good, whereas studies like [77] consider α-values starting from 0.60 as adequate in case of a small number of items.

The results of the reliability analysis for all four scales are depicted in Table 6.2. The value of Cronbach's alpha using all 61 items representing *Correctness* is 0.799. For the *Level of Abstraction*, Cronbach's alpha reaches 0.807 using the 24 items of this scale. Both scales *Abstract Thinking Competence* and *Professionality* derived from the initial coding show good reliability, too. In summary, all four scales show good measurement accuracy.

Scale	n	Scale length	Cronbach's alpha
Correctness	134	61	0.799
Level of Abstraction	134	24	0.819
Abstract Thinking Competence	134	24	0.781
Professionality	134	24	0.749

Table 6.2.: Description of all scales of the Abstract Thinking Assessment (ATA).

6.4. Item Difficulty

Items have different solution rates that are quantifiable as psychometric item difficulty, which is often just called item difficulty. Tough items are solved correctly only by a few participants of an assessment, whereas easy items, are solved by almost every participant. Hence, the item difficulty has a significant influence on the distribution of the results. Moreover, a good variation of item difficuly allows to differenciate participants according to their proficiency [12, 17]. Thus, the range of item difficulty is evaluated for the ATA.

The item difficulty is calculated separately for each item and is indicated by an index corresponding to the proportion of those who correctly solve the item. For dichotomous scales the item difficulty is equivalent to the percentage of correct answers. Thus, the name of the measurement does not necessarily correspond to the everyday understanding of difficulty [12, 17].

As suggested by Bortz & Döring [12], items with an value between 0 and 0.2 are classified as tough, whereas an item difficulty index of 0.8 - 1.0 indicates easy items. An index between 0.2 and 0.8 (excluding the interval limits) is denoted as moderate. In order to be able to differentiate various persons according to their abilities, a good assessment should have a broad distribution of item difficulties. To denote the difficulty of an entire assessment, the average item difficulty per scale can be calculated. Table 6.3 presents the average item difficulty for the three dichotomous scales of the ATA.

Scale	**Average item difficulty**
Correctness	0.62
Abstraction Thinking Competence	0.57
Professionality	0.39

Table 6.3.: Average item difficulty of the three dichotomous scales.

Overall, the ATA shows a moderate average of item difficulty. Detailed analysis of the items, which are presented in the following subsections, show a satisfactory range of item difficulty. Item difficulty ranges from several easy items, over moderate, to a few tough items. Thus, the ATA allows to differentiate students on all levels of proficiency.

Each subsection starts with a summary of the scale in the form of a graphical overview. For dichotomous scales a pie chart reveals the average item difficulty. The overview is supplemented by a bar chart presenting the amount of items per category of item difficulty (easy, moderate and tough). And additionally, the range of item difficulties is indicated by the minimum and the maximum value. In case of the non-dichotomous scale *Level of Abstraction* the distribution of scores is shown using a bar chart. A detailed list of all individual items per scale is presented at the end of each subsection. The item difficulty is reported in case of dichotomous scales. And for the non-dichotomous scale *Level of Abstraction* the distribution of answers across the scores is outlined.

6.4.1. Scale *Correctness*

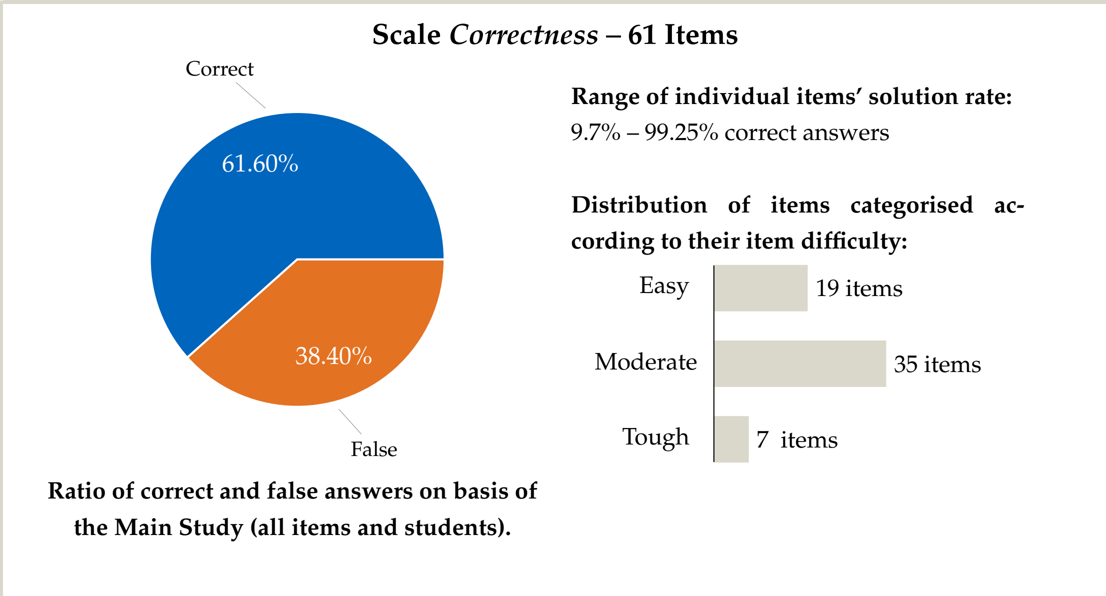


Figure 6.1.: Overview of scale *Correctness* ($n = 134$).

The correctness of students' abstractions is assessed using 61 items. Figure 6.1 presents an overview of the important results of the analysis of this scale. The scale *Correctness* differentiates the whole performance range. Overall, 61.60% of the students' answers have been assessed as correct. Item difficulty varies from easy (solved by 99.25% of the students) to tough (solved by 9.7% of the students). The categorisation of items according to their item difficulty shows that 18 items can be categorised as easy, 36 as moderate and 7 as tough. Each item belongs to one of 11 subsets, such as SUMMARISE or ON THE GRID. An in-depth analysis shows that at least one student achieved the maximum number of points within a subset.

The results in Table 6.4 reveal that items of group SUMMARISE representing the concept of classes are rather easy for the students. The solution rate of RANK items varies from 48.51% to 95.52%.

It is also noticeable that the item REFLECT UPON RANKINGS is very tough. Moreover, the reliability of the ATA increases to a higher factor ($\alpha = 0.803$) when the item is deleted from the assessment. This indicates that the item requires something in addition or different. My assumption is that students struggle with reflecting on their thinking process.

Items, such as PALINDROME VARIOUS LENGTH, that require building an abstraction and parametrising a process are most difficult for students. Items like RULE MEDIUM where students need to be very precise and clear are solved correctly by only a few students as well.

Item – Scale *Correctness*	**Item difficulty**
Rank, analyse and reason	
Rank Apple	0.96
Rank Seal	0.78
Rank Vending Machine	0.87
Rank Typing	0.46
Reflect upon Rankings	0.10
Generalise Rankings	0.42
Rank Ticket	0.82
Justify Ticket	0.46
Rank Species	0.65
Justify Species	0.25
Summarise	
Name Vehicle	0.97
Characterise Vehicle (1)	0.97
Characterise Vehicle (2)	0.98
Characterise Vehicle (3)	0.97
Characterise Vehicle (4)	0.96
Name Brick	0.96
Characterise Brick (1)	0.93
Evaluate Characteristics Brick (1)	0.89
Characterise Brick (2)	0.90
Evaluate Characteristics Brick (2)	0.90
Characterise Brick (3)	0.89
Evaluate Characteristics Brick (3)	0.69
Characterise Brick (4)	0.86
Evaluate Characteristics Brick (4)	0.81
London Railway Network	
Stations	0.70
Intersections	0.81
Location of stations	0.69
Distances	0.62
Representation	0.12
Course	0.78
Justify Tube Map	0.10
Fragmentary Codes	
Easy Code	0.99
Medium Code	0.64
Biber Code	0.62
Music Code	0.72

Item – Scale *Correctness*	**Item difficulty**
Faulty Codes	
Mark and Correct Binary to Alpha	0.81
Mark and Correct Counting Code	0.66
Rule for Counting Code	0.49
Mark and Correct Caesar Code	0.72
Rule for Caesar Code	0.57
Mark and Correct Digital Code	0.22
Rule for Digital Code	0.24
Machines	
Function Sorting	0.80
Function Set	0.54
On the Grid	
Grid Moved Diagonal	0.76
Grid Triangle	0.75
Grid Chess	0.69
Grid Tree	0.15
Pass phrase Security Status	
Pass phrase Components	0.64
Differences weak – medium	0.43
Differences medium – strong	0.19
Rule weak	0.55
Rule medium	0.17
Rule strong	0.25
Potions	
Potion chequered commonalities	0.66
Potion weightless commonalities	0.70
Potion luminous differences	0.40
Automation	
Palindrome constant length	0.41
Palindrome various length	0.13
Smartphone	
Smartphone – 3 abstraction levels	0.31
Justify Smartphone Levels	0.22

Table 6.4.: Item difficulty of items rated on scale *Correctness*.

6.4.2. Scale *Level of Abstraction*

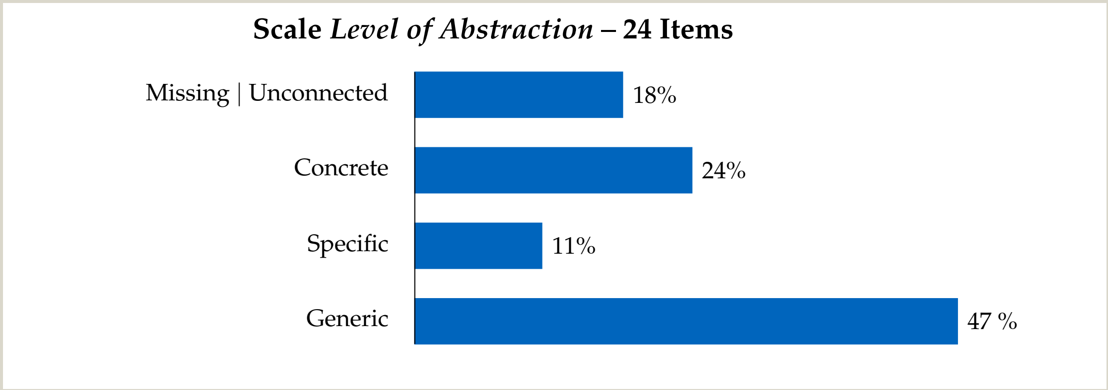


Figure 6.2.: Overview of scale *Level of Abstraction* ($n = 134$).

The scale *Level of Abstraction* comprises 24 items. Figure 6.2 depicts the distribution of the four scores[1] for all students participating in the *Main study*. Contrary to the expectations, almost 50% of students' answers are assessed to be *generic*. Whereas, level *specific* is only achieved by a small number of answers. Table 6.5 reports the detailed frequencies of scores for each item on the basis of all 134 students.

There exist items of categories, such as FUNCTION or GRID (1-3), which 75% of the students solve on a *generic* level. In contrast, items like PALINDROME VARIOUS LENGTH or DIFFERENCES MEDIUM – STRONG PASS PHRASES are solved by less than 25% of the students on the highest level. Answers to the items labelled in the ATA with *Profistufe* (Expert level)[2] are most often *missing* an answer. There exist items, where the majority of students answered on a concrete level. Examples are: PALINDROME CONSTANT LENGTH, DIFFERENCES MEDIUM – STRONG as well as JUSTIFY SPECIES and JUSTIFY TICKET.

[1]The attentive reader might ask, why the frequencies of scores are reported instead of the item difficulty. Experts involved in the design of the ATA could not agree that *Level of Abstraction* is an interval scale. Consequently, values assessed using this scale can just be considered as ordinal values. For this kind of scale, the median would be the mathematically correct measurement. The basis for the calculation of the item difficulty index is the average, but as explained, the average is not an appropriate measurement in this case. Nevertheless, the average is the preferred measurement in classical test theory [17] to report on the item difficulty of ordinal scales, as the median simply splits the sample into two groups of equal size and provides no information about how the two groups internally look like. However, as the distributions of scores of items on the scale *Level of Abstraction* are often bimodal, the mean is not considered to be an appropriate measurement. As all these problems do not occur on dichotomous scales, the scale *Abstract Thinking Competence* derived from the *Level of Abstraction* is used for all further in-depth investigations concerning the competence of abstract thinking.

[2]Indicates that the item is tough by design and students might skip it at first.

When looking at the frequencies of scores per item shown in Table 6.5, almost every conceivable distribution can be found. Several items are often solved on level *generic*, others on level *concrete*, and still others show a balanced distribution over all levels. The only thing that stands out is that level *specific* is highest only for one item (RANK REFLECTION). However, the difference between level *concrete* and *specific* is less than 2%, which cannot be considered as substantial.

Item – Scale *Level of Abstraction*	**Missing/ Unconnected**	**Concrete**	**Specific**	**Generic**
Rank, analyse and reason				
Reflect upon Rankings	6.72%	32.84%	34.33%	26.12%
Generalise Rankings	19.40%	23.88%	23.88 %	32.84%
Justify Ticket	5,97%	50.75%	21.64%	21.64%
Justify Species	10.45%	68.66%	13.43%	7.46%
Faulty Codes				
Rule for Counting Code	20.90%	14.93%	14.18%	50.00%
Rule for Caesar Code	5.97%	14.93%	24.63%	54.48%
Rule for Digital Code	61.19%	16.42%	6.72%	15.67%
Machines				
Function Sorting	11.94%	2.99%	0.75%	84.33%
Function Set	3.73%	8.21%	3.73%	84.33%
On the Grid				
Grid Moved Diagonal	2.24%	2.24%	0.75%	94.78%
Grid Triangle	2.99%	19.40%	0.75%	76.87%
Grid Chess	7.46%	9.70%	4.48%	78.36%
Grid Tree	58.21%	21.64%	17.91%	2.24%
Pass phrase Security Status				
Pass phrase Components	2.99%	14.18%	2.24%	80.60%
Differences weak – medium	7.46%	42.54%	23.13%	26.87%
Differences medium – strong	10.45%	61.94%	17.16%	10.45%
Rule weak	11.94%	26.87%	3.73%	57.46%
Rule medium	14.93%	24.63%	4.48%	55.97%
Rule strong	17.16%	27.61%	14.93%	40.30%
Potions				
Potion chequered commonalities	22.39%	4.48%	1.49%	71.64%
Potion weightless commonalities	17.16%	3.73%	17.91%	61.19%
Potion luminous differences	35.07%	4.48%	2.24%	58.21%
Automation				
Palindrome constant length	32.09%	55.97%	0.75 %	11.19%
Palindrome various length	55.22%	26.87%	2.24%	15.67%

Table 6.5.: Percentage frequencies of scores for items rated on scale *Level of Abstraction*.

6.4.3. Scale *Abstract Thinking Competence*

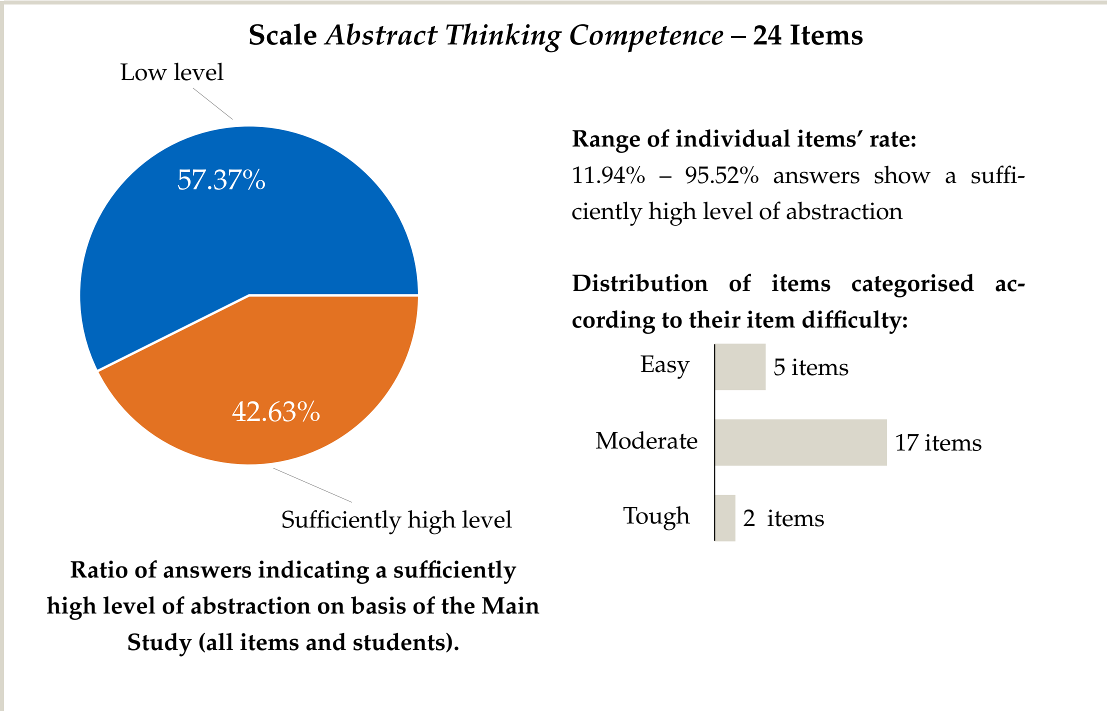


Figure 6.3.: Overview of scale *Abstract Thinking Competence* ($n = 134$).

The scale *Abstract Thinking Competence* is a dichotomous scale derived from on the scale *Level of Abstraction* and includes 24 items. Figure 6.3 shows that 57.37% of the students' answers indicate a sufficiently high level of abstraction. The amount of solutions indicating a sufficiently high level of abstraction ranges from few (11.94%) to many (95.52%). Overall, the items show a balanced distribution, which allows differentiating students on all levels of proficiency. A detailed evaluation of the percentage of answers indicating a sufficiently high level of abstraction is depicted in Table 6.6.

Items of group AUTOMATION and item GRID TREE are solved by less than 20% of the students in a way that shows a sufficiently high level of abstraction. It is not surprising that only a small number of students is able to give a parametrised solution for GRID TREE, as the parameters row-number, column-number and height are interdependent. In contrast to this, it is astonishing that items of the group PALINDROME, which guides students through the abstraction process, show such poor results. Most students first used a non-parametrised approach, numbering the positions of the characters and describing all ten comparisons. If they try to keep this pattern for words of various length, this leads to the problem that the list of comparisons is of indefinite length. Students might not be able to perform the expand process, and thus, are not able to introduce a concept for variable length.

In contrast, many students show a sufficiently high level of abstraction in items Grid Moved Diagonal, Grid Chess and in the group Machines. A reason for the good performance concerning the On the Grid items might be the similarity of the desired solution to mathematical equations, as students are familiar with this concept. The underlying processes of items of group Machines are well-known processes, such as sorting. Consequently, students are very often able to solve these items by showing a sufficiently high level of abstraction.

Item – Scale *Abstract Thinking Competence*	**Item difficulty**
Rank, analyse and reason	
Reflect upon Rankings	0.60
Generalise Rankings	0.57
Justify Ticket	0.43
Justify Species	0.21
Faulty Codes	
Rule for Counting Code	0.64
Rule for Caesar Code	0.79
Rule for Digital Code	0.22
Machines	
Function Sorting	0.85
Function Set	0.88
On the Grid	
Grid Moved Diagonal	0.96
Grid Triangle	0.78
Grid Chess	0.83
Grid Tree	0.20
Pass phrase Security Status	
Pass phrase Components	0.84
Differences weak – medium	0.50
Differences medium – strong	0.28
Rule weak	0.61
Rule medium	0.60
Rule strong	0.55
Potions	
Potion chequered commonalities	0.73
Potion weightless commonalities	0.79
Potion luminous differences	0.60
Automation	
Palindrome constant length	0.12
Palindrome various length	0.18

Table 6.6.: Item difficulty of items rated on scale *Abstract Thinking Competence*.

6.4.4. Scale *Professionality*

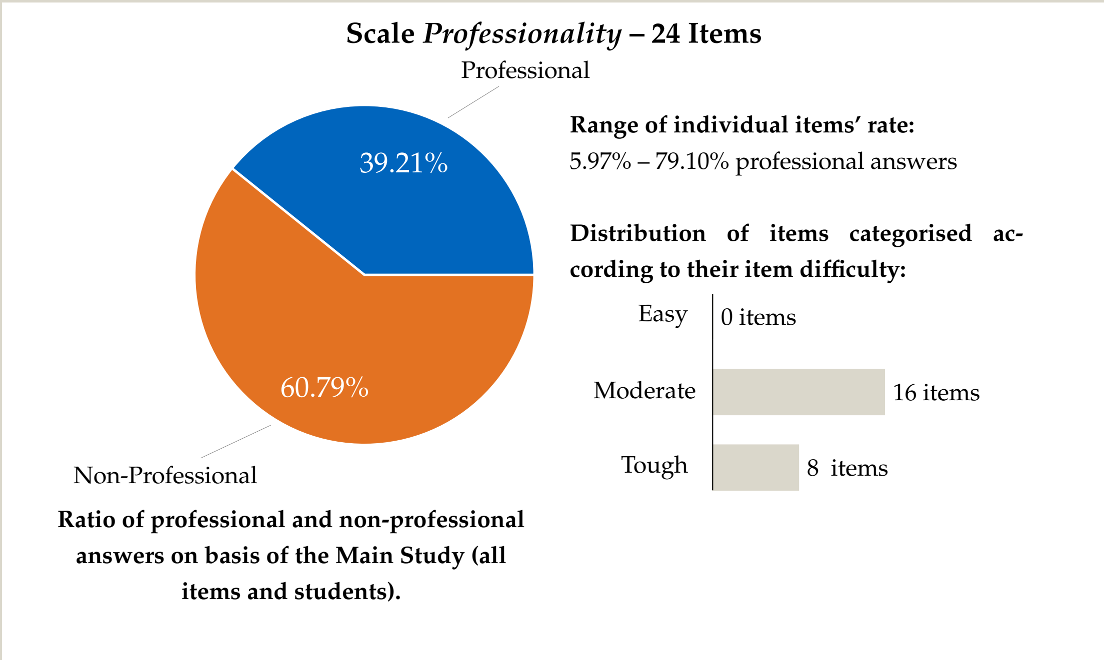


Figure 6.4.: Overview of scale *Professionality* ($n = 134$).

The scale *Professionality* has been derived from scales *Correctness* and *Abstract Thinking Competence*, and consists of 24 items. Figure 6.4 depicts that only 39.21% of the students' answers were assessed as professional. Item difficulty ranges from moderate (solved by 79.10% of the students) to difficult (solved by 5.97% of the students). The analysis shows that 16 items can be categorised as moderate and 8 as tough. Thus, the ATA allows to differentiate between medium and high performing first-year students regarding their professionality. In general, the scale is more difficult for the first-year students. On the other dichotomous scales the average solution rate was more than 50%. On this scale, the solution rate dropped noticeably.

The tough items of this scale require both, a high level of abstraction and accuracy to a high extent to solve them professionally. To clearly differentiate medium pass phrases from weak and strong ones (item RULE MEDIUM), seems to be challenging for the students. When describing a process with a predefined number of steps, students do not parametrise a dynamic process normally (item PALINDROME CONSTANT LENGTH). Item PALINDROME VARIOUS LENGTH indicates that they might not have sufficient skills to perform this step. Answers to item DIFFERENCES MEDIUM – STRONG often just describe the structure of both pass phase groups and lack in the quantification of the differences, which indicates missing abstract thinking competence.

Item – Scale *Professionality*	Item difficulty
Rank, analyse and reason	
Reflect upon Rankings	0.07
Generalise Rankings	0.34
Justify Ticket	0.37
Justify Species	0.13
Faulty Codes	
Rule for Counting Code	0.45
Rule for Caesar Code	0.53
Rule for Digital Code	0.16
Machines	
Function Sorting	0.79
Function Set	0.49
On the Grid	
Grid Moved Diagonal	0.75
Grid Triangle	0.64
Grid Chess	0.65
Grid Tree	0.06
Pass phrase Security Status	
Pass phrase Components	0.63
Differences weak – medium	0.37
Differences medium – strong	0.15
Rule weak	0.54
Rule medium	0.17
Rule strong	0.25
Potions	
Potion chequered commonalities	0.64
Potion weightless commonalities	0.69
Potion luminous differences	0.40
Automation	
Palindrome constant length	0.06
Palindrome various length	0.07

Table 6.7.: Item difficulty of items rated on scale *Professionality*.

Summary of findings in Chapter 6

The ATA fulfils all quality criteria for a good assessment. It is valid, reliable and objective. Moreover, the analysis of item difficulty shows that the ATA allows for distinguishing students on the whole range of performance. The analyses additionally reveal that students are able to recognise abstractions, but building their own abstractions seem to be challenging for them. Furthermore, it revealed that on scale *Professionality* the majority of answers is categorised as non-professional, is the first indicator for a deficit among the first-semester students' competence of abstract thinking.

7. Analysis of the Student Population

Students starting in the bachelor's programme of computer science at the Munich University of Applied Sciences, are an extremely heterogeneous group regarding cultural, educational and professional background. Thus, it is tough for lecturers, to pick them up where they are and help them to develop whatever they need to study successfully.

Prior to an in-depth analysis of students' competence, it is interesting to gain insights into students' ideas of abstraction and abstract thinking. A negative association with for example fear or difficulty might cause unconscious learning obstacles. This could especially appear, when abstract thinking is made more explicit in the lecture.

As shown in Chapter I, a well-researched definition of abstract thinking has just been introduced in this work. However, everyone has an intuitive understanding of the terms abstract, abstraction and abstract thinking, thus also the students. They wrote down their own intuitive definitions of *abstract* during the justification of rankings. Their definitions are often in accordance with the colloquial use described in Section 1.2, or are very vague. Students define abstract for example as:

- *vague*
- *less well imaginable*
- *generalisation*
- *difficult*
- *difficult to find an example*

As abstract thinking is a crucial competence in computer science it is important for lecturers to know their incoming students' profiles regarding the competence. With help of the ATA, I can now provide lecturers with an idea of their students' competence of abstract thinking. Thus, they know right from the beginning of a semester what they can build on, and what part of the competence has yet to be developed in the majority of students. Knowing this, lecturers can develop new teaching concepts that take these findings into account. Moreover, I can examine the development of students' competence over time.

Consequently, Section 7.1 depicts an overview of the student population of winter semester 2017/18 based on data collected at the beginning of their studies. The curriculum of the bachelor's programme at the Faculty of Computer Science and Mathematics schedules object-orientation in the first semester. Thus, an analysis has been conducted to explore students' intuitive understanding of objects when entering university (see Section 7.2). At the end, the development of students' competence over a period of one semester is investigated (see Section 7.3).

7.1. Overview of First-year Students' Results

In order to adjust the teaching right from the beginning, it is necessary, to know the incoming student population as good as possible. The goal of this analysis is to identify groups, typical student profiles and to describe the degree of heterogeneity of first-year students, with respect to their competence of abstract thinking.

The following subsections are organised according to the four scales and investigate the results of the students participating in the *Main study*. Each subsection shows a histogram of the results of the student population to provide insights into the heterogeneity of the students. Additionally, this might lead to first indications regarding groups of students with similar performance. In order to define groups, the well-known categorisation of *good*, *bad* and *murky middle* [21] is preferred.

7.1.1. Scale *Correctness*

Figure 7.1 shows the distribution of students with respect to the points achieved on scale *Correctness*. The total score of students' correct answers ranges from 10 to 50 out of a maximum of 61 points. On average, the student population achieved 38 points, which corresponds to 62.29%. Thus, a large part of the students achieve more than 50% of the maximum points on this scale.

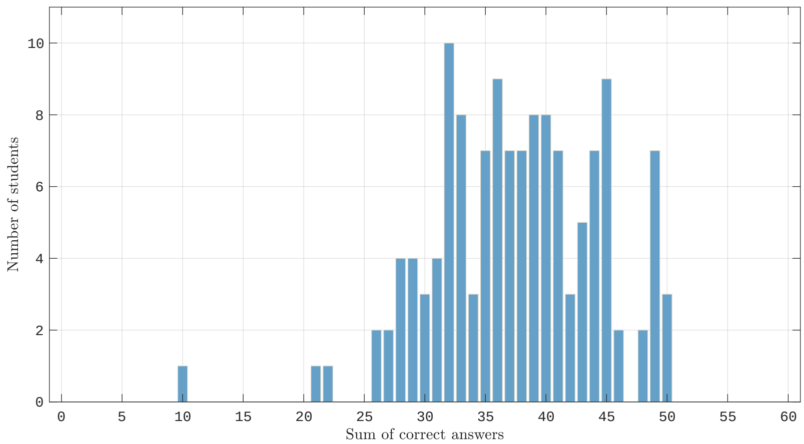


Figure 7.1.: Distribution of students' total scores on scale *Correctness* based on the data of the *Main study* ($n = 134$).

When looking at the number of points students achieved in Figure 7.1, it is positive to note that the majority of students achieved more than 40% of the maximum number of points. The figure also shows that students still have scope for improvement. Moreover, no significant gap can be found in the histogram. This is an indicator that the students' results are spread over the whole spectrum of correctness and that they show a high degree of heterogeneity regarding their performance on scale *Correctness*.

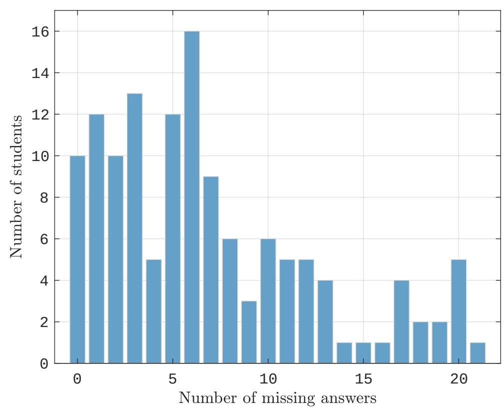
Consequently groups, like *good*, *bad* and *murky middle* cannot be clearly identified in the data. For an attempt to categorise students, the gaps between 23 and 25 points could be used to split the bad from the murky middle. Good students achieved more than 47 points. As a result, 3 students are insufficiently prepared, 12 students are categorised as good and all other 119 student count to murky middle. The information gained by this approach is very little, as almost 90% of the students are classified to potentially struggle during the study process.

As a consequence, the high heterogeneity makes adaptations of teaching quite difficult. Material cannot be simply adjusted to three groups of similar students. The groups cannot be confirmed and their profiles are not well defined, yet.

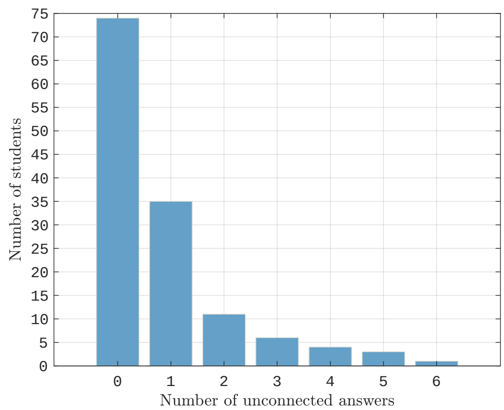
On average, students did not answer 7 out of 61 questions. In total, the distribution depicted in Figure 7.2 (a) indicates that the time frame was set appropriately, as the majority of students left no more than six items blank, which is less than 10%. The analysis of students' individual distribution of scores throughout the ATA supports this assumption. It does not show a relation between skipped items and the items' position in the assessment.

A student with a remarkable high number of 48 *missing* answers can be found in the data. About the reason for this high number only assumptions can be made. It might be caused due to a lack of motivation, a lack of abstract thinking competence, a lack of reading ability or something else.

The number of *unconnected* answers in the ATA ranges from 0 to 6 per student (see Figure 7.2 (b)). The mean of unconnected answers is 0.84 per student. The distribution of unconnected answers has its peak at 0 and decreases continuously. From this I conclude that items were expressed comprehensibly.



(a) Distribution of missing answers.
(Student with 48 missing answers not depicted here.)



(b) Distribution of unconnected answers.

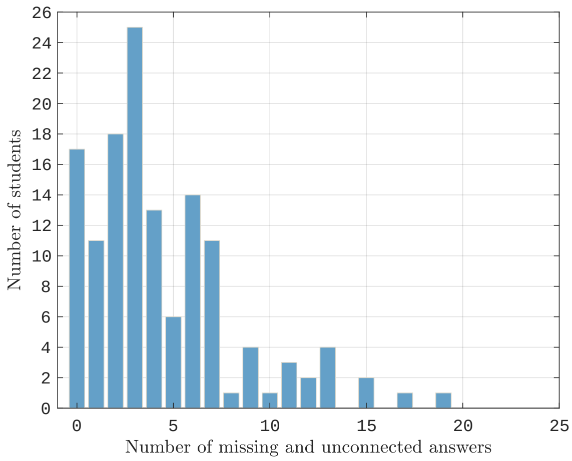
Figure 7.2.: Distribution of students' number of *missing* (left) and *unconnected* (right) answers on scale *Correctness* based on the data of the *Main study* ($n = 134$).

7.1.2. Scale *Level of Abstraction*

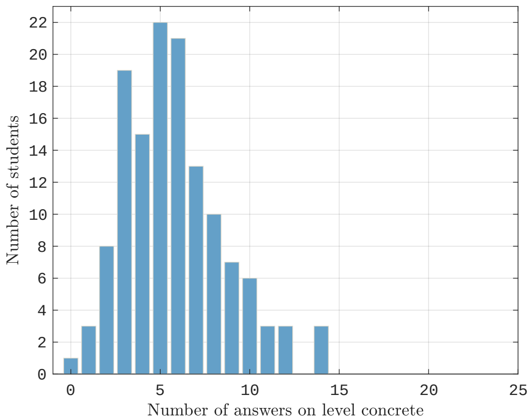
The four histograms depicted in Figure 7.2 present the distribution of the students' total scores with respect to the four scores that have been assigned on scale *Level of Abstraction*. The sub-figures again show a high heterogeneity of the student population and do not clearly show groups.

Figure 7.2 (a) shows that there are only some students with an increased number of missing or unconnected answers. Sub-figure (b) reveals that only three students answered more than 50% of the items on a *concrete* level. During lectures, we experienced that students focus on given objects while building abstractions. Thus, their solution is more than *concrete*, but not *generic* yet. This is not confirmed by the data (see Figure 7.2 (c)). Students' answers are rarely assessed as specific. The histogram (d) showing level *generic* illustrates that each student out of 134 gave at least three answers, which were classified as generic. This suggests that all students have at least a basic level of abstract thinking competence. In addition, the sub-figure shows a significant drop at 14 answers. This does not correlate to the categorisation of items according to their item difficulty. Possible reasons could be a lack in concentration or problems with the characteristics of the items. To answer this question, further investigations have to be conducted.

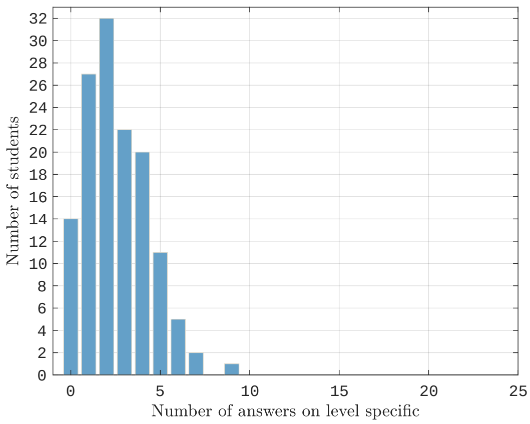
In order to get further insights into students' profiles eight profiles are exemplary depicted in Table 7.1. They represent the students with most (right) and least (left) answers on a specific level. These profiles already indicate the large variety of distributions, which can be confirmed when looking at all student profiles. Several students show a profile with many generic answers, and others exhibit an almost balanced number of answers across all levels. Nevertheless, as the levels per student cohere: the higher the number of answers on one level, the lower are the numbers on the other levels.



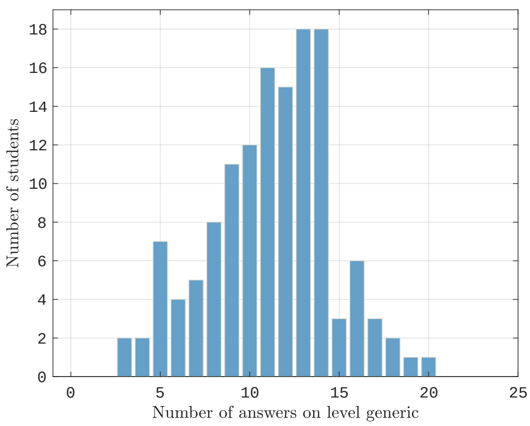
(a) Distribution of missing and unconnected answers.



(b) Distribution of answers on level concrete.



(c) Distribution of answers on level specific.

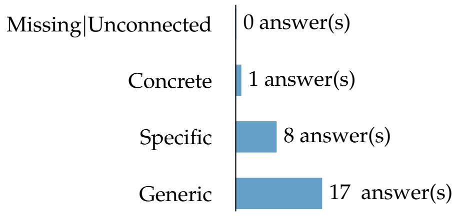


(d) Distribution of answers on level generic.

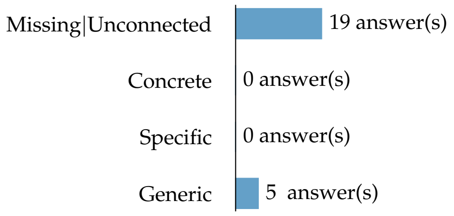
Figure 7.2.: Distribution of students' total scores on scale *Level of Abstraction* based on the data of the *Main study* ($n = 134$).

Missing | Unconnected

Distribution of answers per level for student with least answers missing or unconnected:
(One student profile has been randomly chosen.)

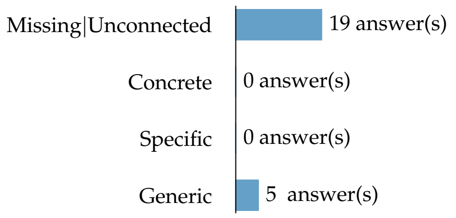


Distribution of answers per level for student with most answers missing or unconnected:

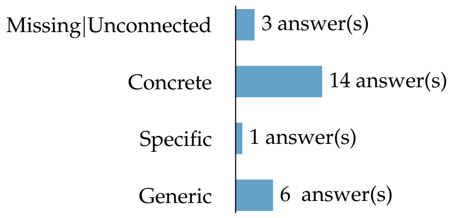


Level Concrete

Distribution of answers per level for student with least answers on level concrete:

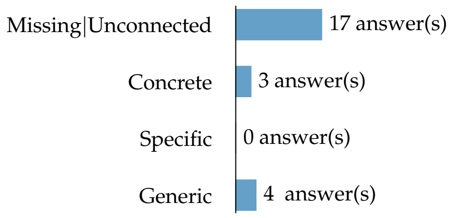


Distribution of answers per level for student with most answers on level concrete:
(One student profile has been randomly chosen.)

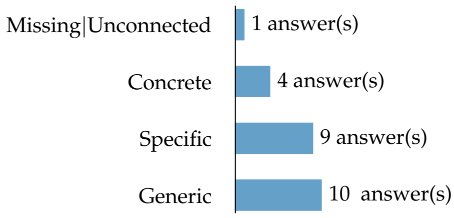


Level Specific

Distribution of answers per level for student with least answers on level specific:
(One student profile has been randomly chosen.)

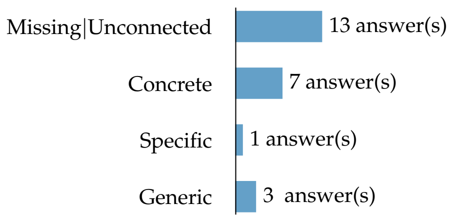


Distribution of answers per level for student with most answers on level specific:



Level Generic

Distribution of answers per level for student with least answers on level generic:
(One student profile has been randomly chosen.)



Distribution of answers per level for student with most answers on level generic:

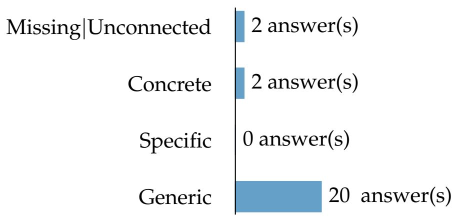


Table 7.1.: Overview of student profiles at the extrema of answers per level regarding the scale *Level of Abstraction*.

7.1.3. Scale *Abstract Thinking Competence*

Figure 7.3 presents the distribution of the number of points students achieved on scale *Abstract Thinking Competence*. The total score of individual students' answers ranges from 4 up to 23 out of a maximum 24 points. On average, students achieved 14 points on this scale, which corresponds to 57.37% of the number of points possible.

The distribution of students points in Figure 7.3 again reveals a gap to zero points and again shows a high degree of heterogeneity. A significant gap in the points students achieved cannot be found. However, there exists a substantial decrease in the number of students from 12 to 13 points and a increase from 16 to 17 points. These can be used to attempt a classification of the students. As a result, 40 students are categorised as insufficiently prepared, 63 as murky middle and 31 as well prepared. However, this suggestion for categorising students would need further research.

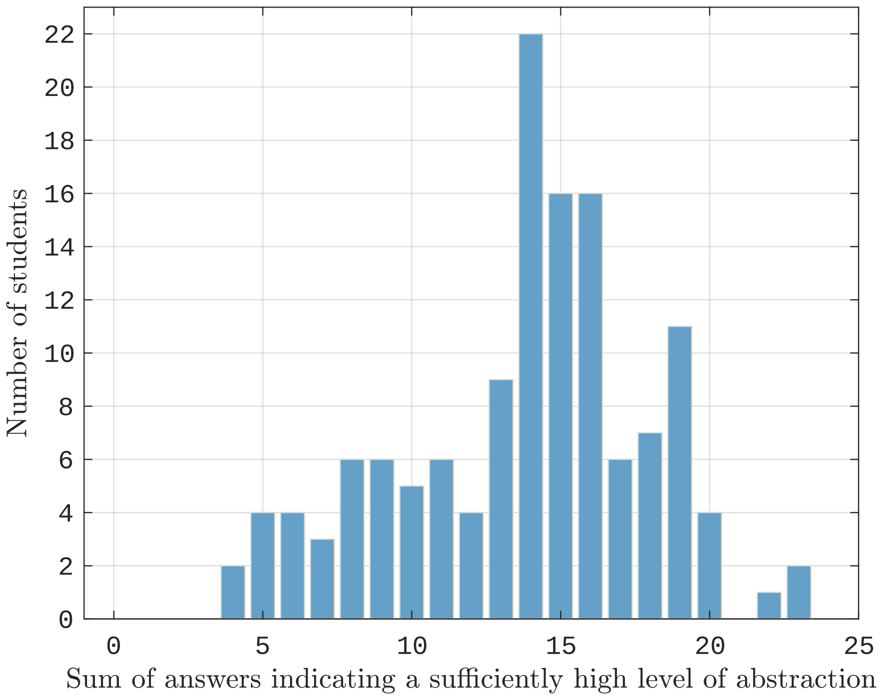


Figure 7.3.: Distribution of students' total scores on scale *Abstract Thinking Competence* based on the data of the *Main study* ($n = 134$).

7.1.4. Scale *Professionality*

The distribution of students with respect to the sum of points they achieved on scale *Professionality* is depicted in Figure 7.4. Students' number of answers assessed as professional ranges from 2 up to 18 out of a maximum of 24 points. On average, students achieved 9 points, which is 37.50% of the number of points possible. Hence, the average points students achieve dropped noticeably compared to the other scales and the majority of students' answers is categorised as non-professional.

The histogram in Figure 7.4 shows no big gaps, which would provide indications of groups. The image of a heterogeneous student population is confirmed once again. A drop can be noted from 14 to 15 points. This can not be simply explained by the categorisation of the items according to their item difficulty. Thus, further research should be conducted to explain the drop.

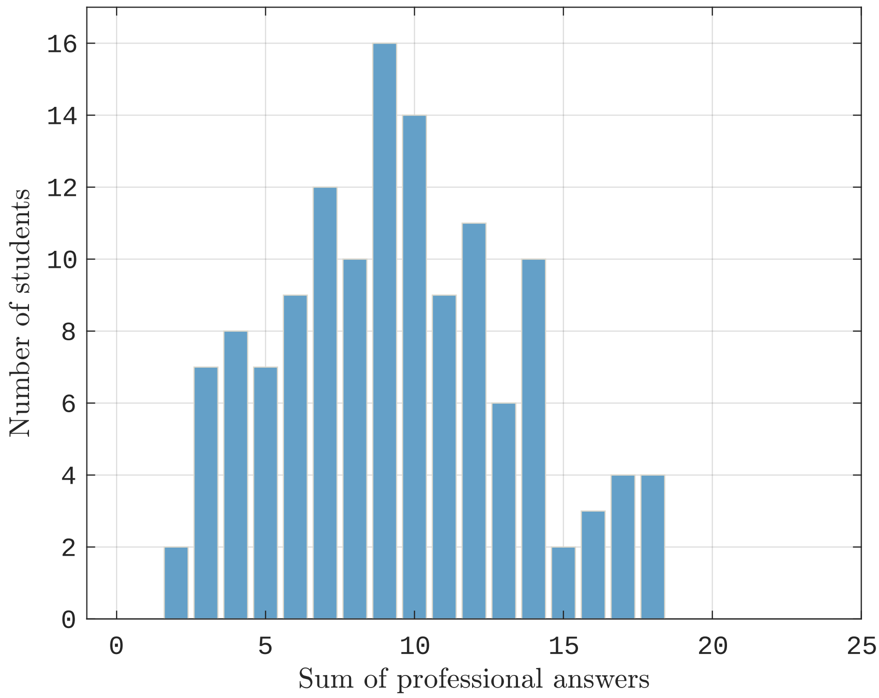


Figure 7.4.: Distribution of students' total scores on scale *Professionality* based on the data of the *Main study* ($n = 134$).

7.2. Novices' Intuitive Understanding of Objects

The curriculum of the bachelor's programme in computer science at Munich University of Applied Sciences and many others, schedules the introduction of the object-oriented paradigm in the first semester. A typical task in the context of object-orientation is the development of classes. Programmers need to find an appropriate name for the class representing several objects. Each class contains attributes and methods describing common characteristics. In order to adapt teaching to students previous knowledge, it is useful to know students intuitive understanding of objects.

The ATA contains two questions (see Figure 4.5 for an example) 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. The questions focus on summarising depictions under one umbrella term or on naming common details. Either way, students need to identify common characteristics of a set of entities. Characteristics students' named have been categorised into five classes during the rating process. A clipping from the coding manual that lists the categories and names examples is depicted in Table 7.2.

Category	Examples
Missing | Unconnected	Empty, "I don't know", Car
Component	Tires, Windows, Seats
Attribute	Colour, Weight, HP
Behaviour	Drive, Consume fuel
Others	Number of Tires, Carry people, Require a driving licence

Table 7.2.: Coding manual for the categorisation of answers concerning the item SUMMARISE – VEHICLE.

Most of these categories can be mapped to a specific programming concept; e.g., components are represented by associations, and behaviour by methods. With this analysis, I want to evaluate which concept students spontaneously use to describe objects.

Figure 7.5 depicts that students most often choose components or attributes to characterise objects. Both refer to the static aspects of the objects. Only a few students see the dynamic aspect of objects, which would be implemented as methods in object-oriented programming. Humbert [47, 48] describes in his work that students often interpret objects as data storage, and thus, overemphasize the data aspect of objects. This preconception might also be present in the students participating in my *Main study*. As students intuitively use attributes and components to describe characteristics of similar objects and rarely name behavioural characteristics, this should be considered in the introduction of object-oriented programming.

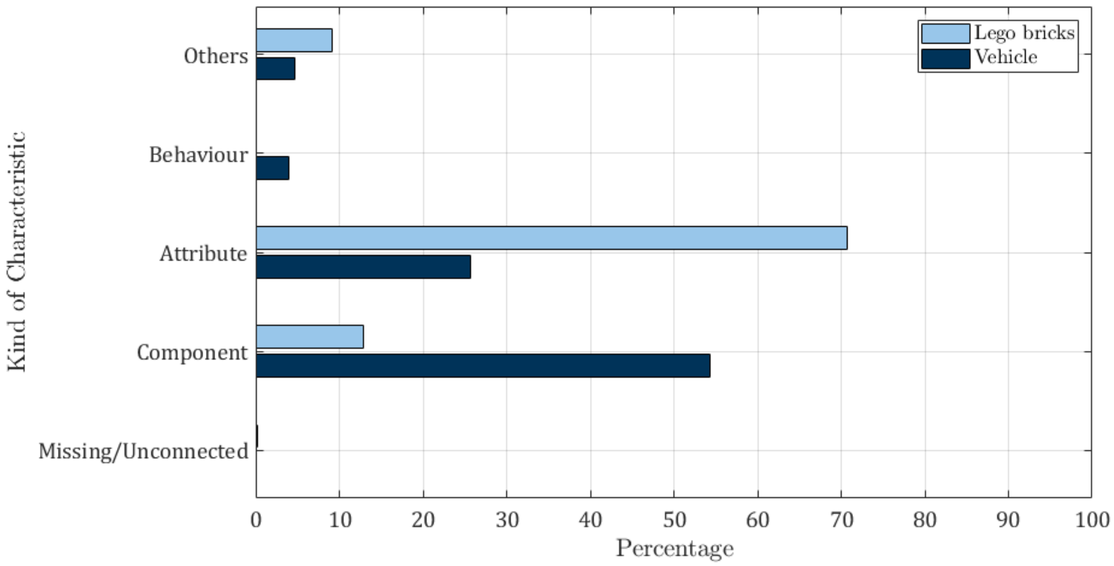


Figure 7.5.: Distribution of the kind of characteristics students used to describe collections of lego bricks and vehicles.

Commonly, attributes and methods are introduced early and quickly. Due to the findings, lecturers should put more emphasis on teaching methods, spend more time on this concept and address it more explicitly.

Moreover, vehicles are a common example to introduce the behaviour of classes, like acceleration. The analysis shows that behaviour is not the students' first choice to describe the characteristics of objects, especially of vehicles. Thus, the selection of introductory examples should be reconsidered and chosen carefully.

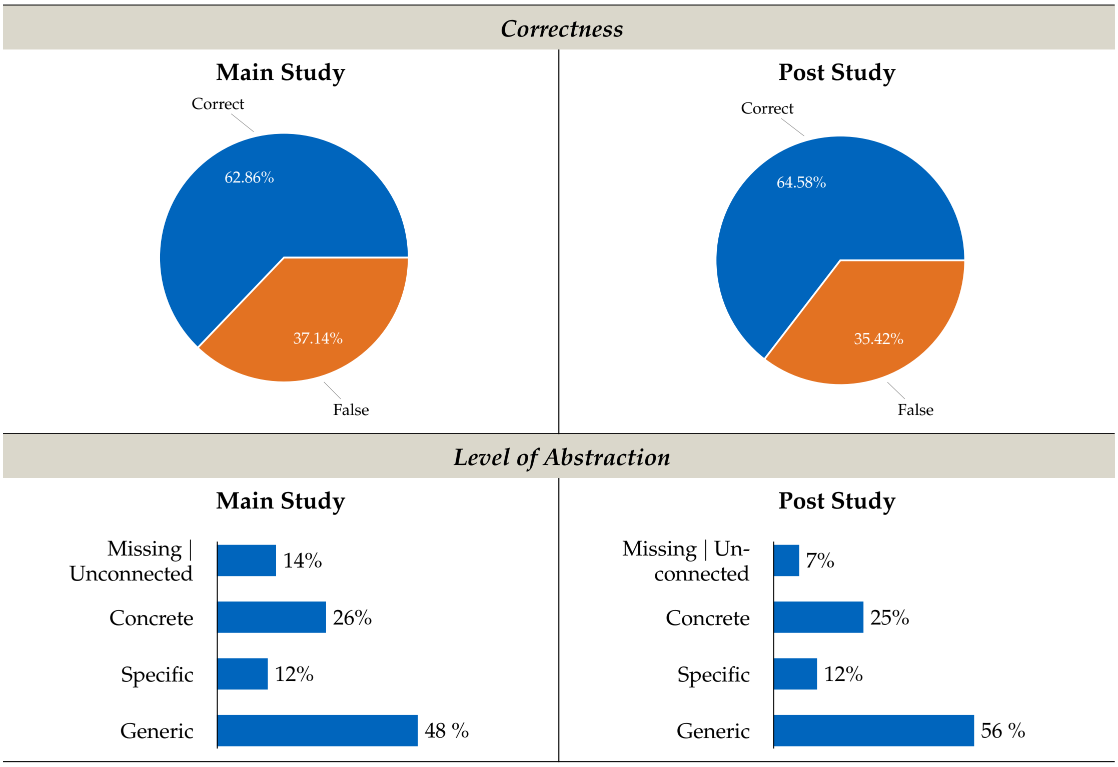
Further adaptations can be made regarding the sequence of topics. Components in the form of associations could be taught much earlier in the curriculum as students have an intuitive understanding of the concept. Associations usually remain a hidden concept in the beginning. Only advanced courses later in the curriculum deal with the concept explicitly, when introducing aggregation and composition. However, using classes as part of another seems to be a concept that students understand intuitively.

7.3. Development of Students' Competence

During the first semester, students train their competence of abstract thinking in several modules, such as *Software Development I*. In order to evaluate the development of students' abstract thinking competence over one semester, the ATA has been conducted again six months after the *Main study* at the beginning of the students' second semester in a *Post study*. This time, the bonus for participating were power banks being raffled and each student got an individual evaluation of his or her ATA results. In total, 45 students took part in the *Post study*, which is only one-third of the participants of the *Main study*.

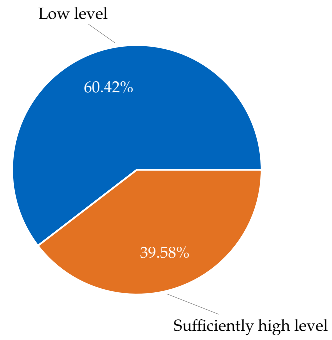
To measure the development of the competence, it is necessary to have matching pairs of the assessment filled in by the same student. The matching ends up with 38 students that took part in both studies. This group shows slightly better results in the *Main study* compared to the results of all participants in the *Main study*. In conclusion, the generality of the statements that can be derived are limited due to the small number of students and their representativeness.

Table 7.3 illustrates the comparison of *Main* and *Post study* results of the 38 students. There is an increase on the generic level of abstraction, and thus, in the abstract thinking competence. The scale *Correctness* also depicts a slight improvement. As *Professionality* is a combination of *Correctness* and *Abstract Thinking Competence*, the evaluation shows only a small growth of proficiency.

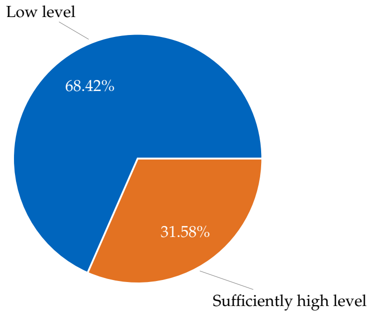


Abstract Thinking Competence

Main Study

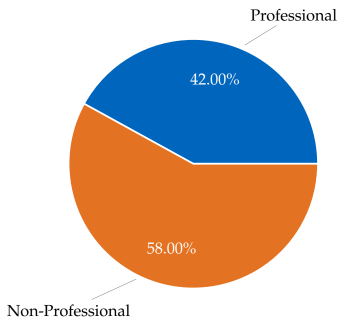


Post Study



Professionality

Main Study



Post Study

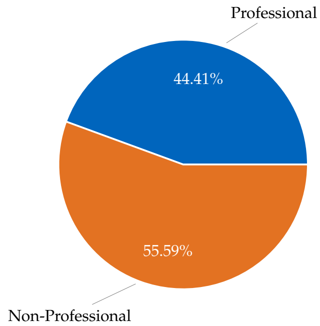


Table 7.3.: Comparison of the results of the main and post study for all four scales, limited to the students participating in both studies ($n = 38$).

To evaluate the significance of the development, the Wilcoxon-Test for paired samples has been used. This test is necessary to evaluate whether the increase in competence is random or not. The Wilcoxon-Test is a variation of the t-Test for dependent samples. It requires related samples and in contrast to the t-Test, it can be applied to ordinal scales. Thus, I decided to use the Wilcoxon-Test instead of the t-Test. In order to quantify magnitude of the development of students' competence, the effect size is calculated.

The results presented in Table 7.4 show that only the decrease in missing answers, the increase answers on level generic, as well as the increase in answers indicating a sufficiently high level of abstraction is statistically significant (p-value $\leqslant 0.05$). Students' development on all other scales is not statistically significant. The effect size r representing the magnitude of the development has been calculated for each scale. For the scales with a significant increase the development of students' competence shows a large effect, meaning that it is substantial.

Scale (individual score when required)	**Evaluation of Students' Development**	
	Significance	**Effect size**
Correctness	$p = 0.264$ not significant	$r = 0.18$ small effect
Level of Abstraction – Missing	$p = 0.001$ significant	$r = 0.56$ large effect
Level of Abstraction – Concrete	$p = 0.580$ not significant	$r = 0.009$ small effect
Level of Abstraction – Specific	$p = 0.952$ not significant	$r = 0.001$ small effect
Level of Abstraction – Generic	$p < 0.001$ significant	$r = 0.596$ large effect
Abstract Thinking Competence	$p = 0.003$ significant	$r = 0.479$ large effect
Professionality	$p = 0.547$ not significant	$r = 0.098$ small effect

Table 7.4.: Overview of the significance and the effect size of students' development ($n = 38$).

Figures 7.6 to 7.11 illustrate the development of 38 students over one semester. The line graphs illustrate the progress of individual students, whereas the area graphs depict a comparison of the student population according to their points achieved in *Main* and *Post study*. This analysis is reduced to the scales showing a significant development.

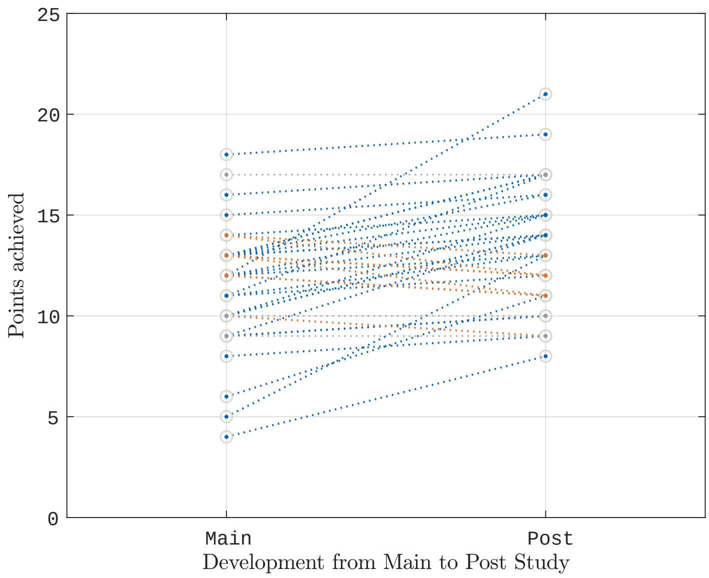


Figure 7.6.: Analysis of the development of individual students with respect to the number of answers on a generic level on scale *Level of Abstraction* ($n = 38$). (blue: increase, orange: decrease, gray: constant)

In case of the level generic on scale *Level of Abstraction*, eight students deteriorated on level generic, three achieved the same number of points and all other 27 achieved more points in the *Post study* (see Figure 7.6). The overview of the student population (Figure 7.7) shows that students made good progress within their first semester: the minimum number of points increased as well as the maximum number of points.

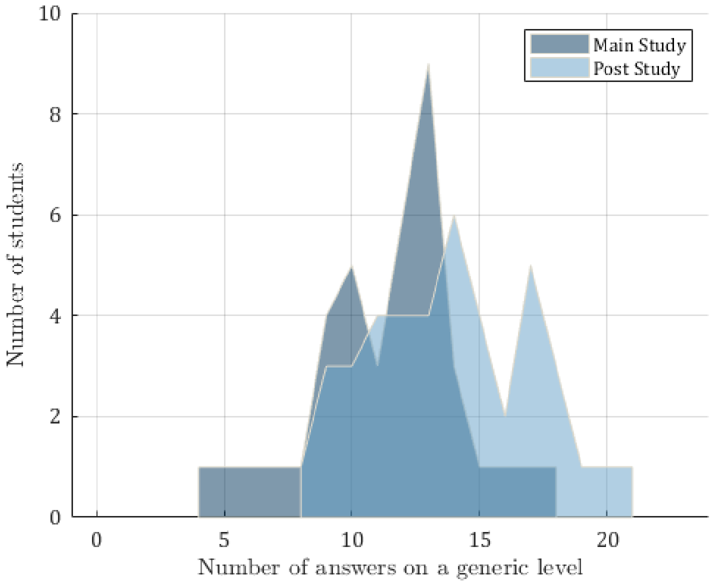


Figure 7.7.: Analysis of the development of the student population with respect to the number of answers on a generic level on scale *Level of Abstraction* ($n = 38$).

A similar trend can be observed in Figures 7.8 and 7.9 depicting students' development on scale *Abstract Thinking Competence*. One student was able to achieve the maximum number of points in the *Post study*. In Figure 7.8 this student can be identified. He or she developed from 15 answers indicating a sufficiently high level of abstraction to 24. When looking at the initial level of competence (see Figure 7.9), the student belonged to the intermediate performance group. Overall, 24 students have shown a growth on scale *Abstract Thinking Competence*, 10 achieved less points on this scale and four students kept their level of proficiency.

In the *Main study*, the distribution of points of the 38 students shows a normal distribution (see Figure 7.9). After one semester, the shape of the distribution changes from one main peak to three peaks. Thus, it seems that students start to split into groups according to scale *Abstract Thinking Competence*. This might indicate that there are students that develop their competence as lecturers would expect it and others, that need more time or further interventions to catch up on their competence.

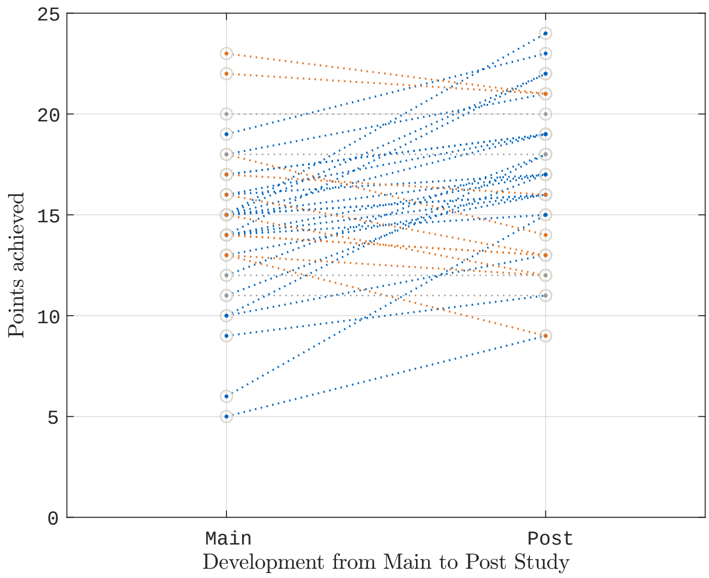


Figure 7.8.: Analysis of the development of individual students with respect to scale *Abstract Thinking Competence* ($n = 38$). (blue: increase, orange: decrease, gray: constant)

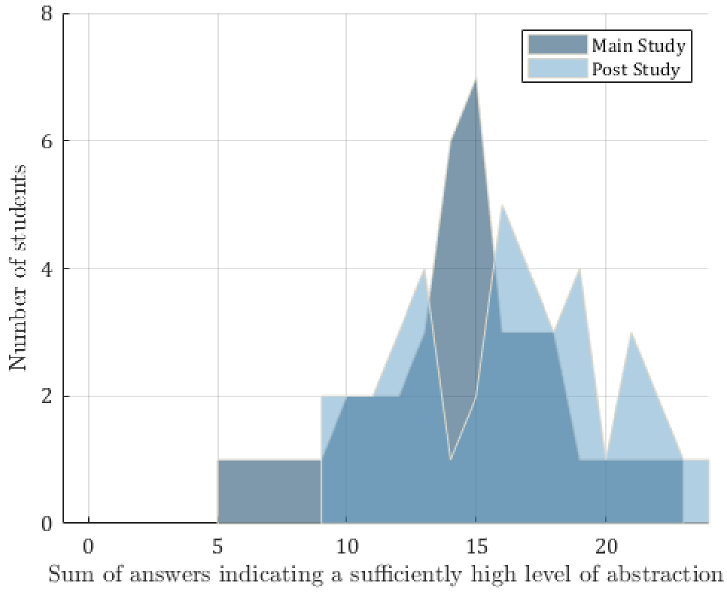


Figure 7.9.: Analysis of the development of the student population with respect to scale *Abstract Thinking Competence* ($n = 38$).

For 23 students the number of missing or unconnected answers dropped (see Figure 7.10). The biggest positive change can be observed in students with an initially high number of missing answers. However, these students also had the most potential to increase their competence. In case of seven students, the number of missing or unconnected answers increase. This could have several reasons: the students did not concentrate or did not take the assessment seriously. Another eight students in the *Post stud* ended up with the same number of missing or unconnected answers than in the *Main study*.

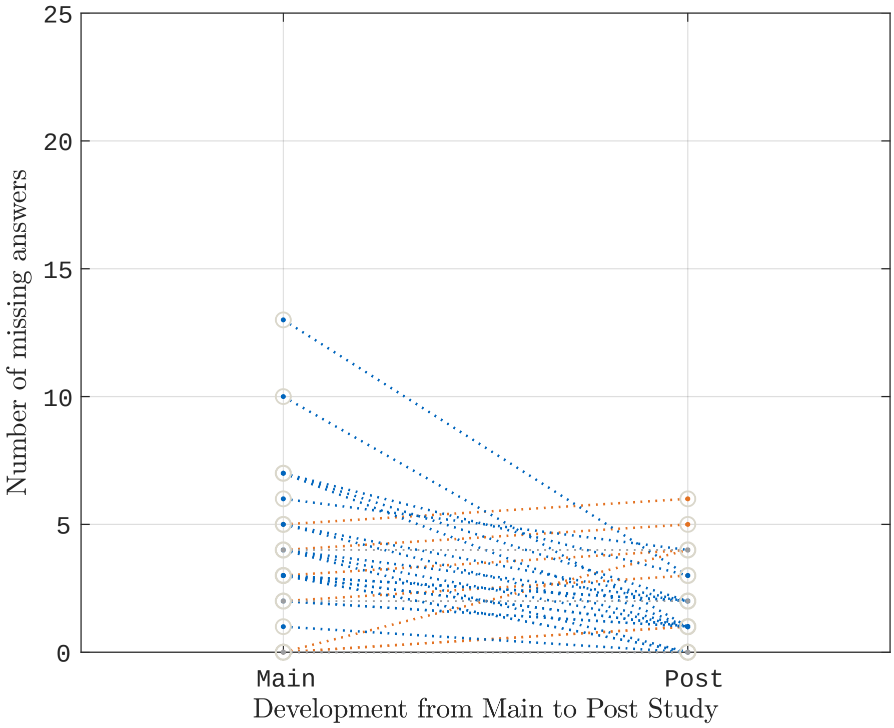


Figure 7.10.: Analysis of the development of individual students with respect to missing and unconnected answers ($n = 38$). (blue: increase, orange: decrease, gray: constant)

Figure 7.11 depicting the student population also shows a positive development regarding the number of missing or unconnected answers. The maximum number of missing answers per students decreased from 13 to 6 answers. The majority of students ranges from 0 to 4 missing or unconnected answers in the *Post study*.

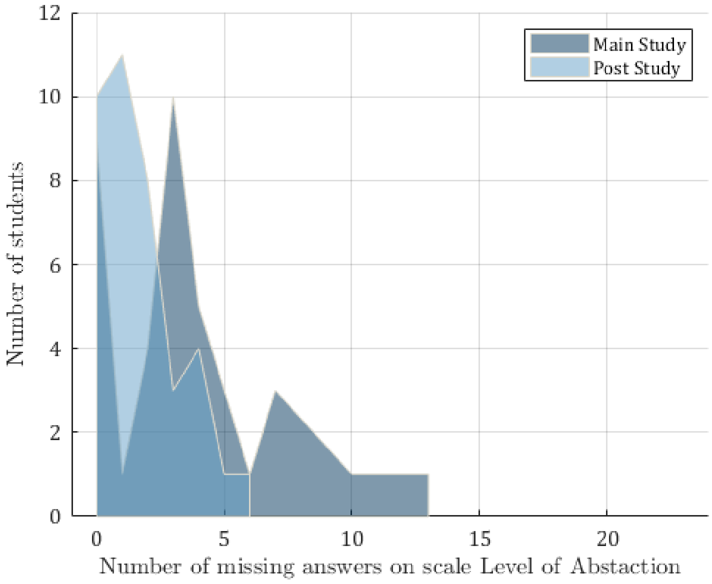


Figure 7.11.: Analysis of the development of the student population with respect to missing and unconnected answers ($n = 38$).

In summary, a small but significant improvement in the competence of abstract thinking can be observed after one semester. The number of answers that indicate a sufficiently high level of abstraction increased from 60.42% to 68.42%. Additionally, the number of missing or unconnected answers decreased significantly by 7%.

Summary of findings in Chapter 7

The initial analysis of student data did not reveal any groups among the students regarding their performance. The commonly used categories good, bad, and murky middle go down in the high heterogeneity of the student population. Classes in the object-oriented paradigm reflect the static and dynamic characteristics of a set of entities. Nevertheless, the analysis showed that students intuitively use attributes and components when describing the common aspects of given entities. Only a few students named characteristics describing the dynamic aspect of the entities, which are implemented as methods in an object-oriented programming language. And after one semester, students developed from a low level of abstraction to a sufficiently high level of abstraction significantly, but only to a small extent. In contrast, the correctness of their abstractions did not improve significantly. This results in a non-significant development of students' professionality. The insights gained by the student analysis will enable lecturers to specifically tailor lectures and materials to the different needs of first-year students in computer science or related topics.

8. Verification of Hypotheses

Researchers like Kramer [53] assume that good computer science students benefit from their "ability to perform abstract thinking and to exhibit abstraction skills." Moreover, computer science educators agree that the competence of abstract thinking is essential for learning computer science and programming in particular [9]. Nevertheless, Thurner et al. [93] report a deficit in abstract thinking competence among their first-year students in computer science or related topics. Experiences I made during the coding process suggest that there exists a relation between the level of abstraction and the correctness of a solution. Using the data collected in the *Main study* these hypotheses are examined in this chapter.

8.1. Relation between *Correctness* and *Abstract Thinking Competence*

Experiences in the coding process suggest that there is a relation between correctness and level of abstraction, although both have been assessed independently. In order to describe commonalities, differences and relationships of a large number of artefacts, a high degree of accuracy on a concrete level is necessary, which students often did not show. The higher the level of abstraction, the more general is a description and the higher the chance that the description fits all artefacts. Thus, I expect a relation between the correctness of a solution and its level of abstraction.

Figure 8.1 shows a bubble plot of all 134 students participating in the *Main study*. On the x-axis the points achieved on scale *Correctness* (reduced to 24 items where the level of abstraction can be measured) are depicted. The number of points achieved on scale *Abstract Thinking Competence* is shown on the y-axis.

The plot indicates a relation between *Abstract Thinking Competence* and *Correctness*. Pearson's correlation coefficient of 0.74 indicates a strong correlation. The p-value, which is $< 2.2e - 16$ defines the relation as statistically significant. Thus, students who achieve a higher number of points for their correctness are more likely to describe the abstractions they built on a sufficiently high level of abstraction.

After the correlation has been proven, a detailed analysis of the distribution of correct and false on the different levels of abstraction is performed. Figure 8.2 depicts the results of all students and all items in a stacked bar plot. The four levels of the scale *Level of Abstraction* are depicted on the x-axis. On the y-axis, the percentage of answers is depicted, which is split according to the number of correct and false answers one this level.

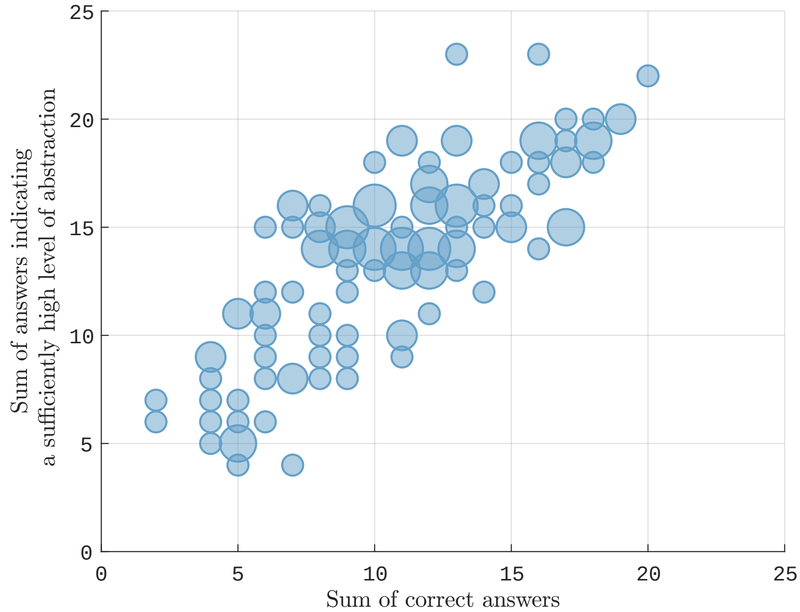


Figure 8.1.: Relation of *Correctness* and *Abstract Thinking Competence*. The number of students with the same values is indicated by the diameter ($n = 134$).

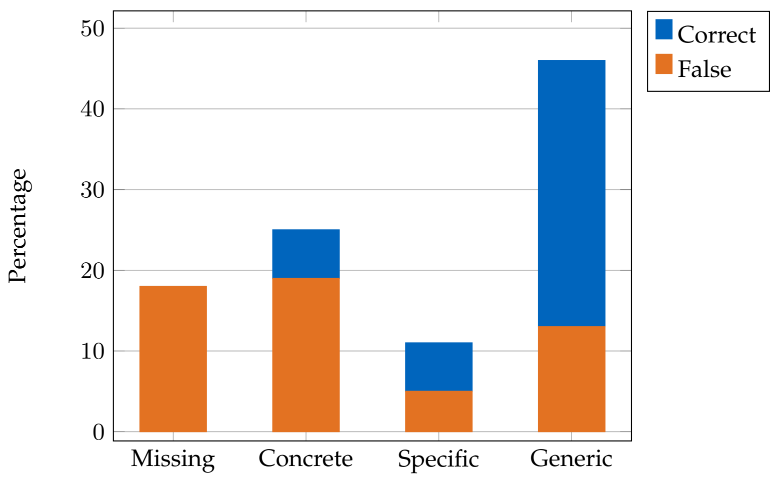


Figure 8.2.: Distribution of correct and false answers on each level of abstraction ($n = 134$).

It is not surprising that empty or unconnected answers are always false, as this interdependency was specified in the scoring scheme. The relation between level of abstraction and correctness becomes clear again in this plot. On level concrete only a few answers are correct. *Specific* answers are divided almost fifty-fifty with respect to correctness, whereas level *generic* shows a large number of correct answers (see Figure 8.2).

From observations during the coding phase, an explanation for the distribution could be derived. Describing a collection of objects or processes concretely requires a high level of accuracy, as there may be hundreds or thousands of special cases. In terms of the ATA as a performance test, there would not be enough time for such an approach. On the other hand, students sometimes fail to build a correct abstraction on a high level of abstraction, as they often use wrong parametrisation or miss to increase a counter. Other students use a wrong level of abstraction and thus include objects or processes they should explicitly exclude.

These approaches and their interdependencies are summarised and generalised in Table 8.1. The accompanying analysis of the distribution of correct and false answers indicating a sufficiently high level of abstraction is presented in Figure 8.3.

	Abstract Thinking Competence	
	Low level	Sufficiently high level
Correct	Complete and accurate description of all individual entities	Precise summary using appropriate parameters to represent a set of entities
False	Incomplete or inaccurate attempt to describe all individual entities	Imprecise abstraction not representing the whole set of entities

Table 8.1.: Characteristics of the results of an abstract thinking process.

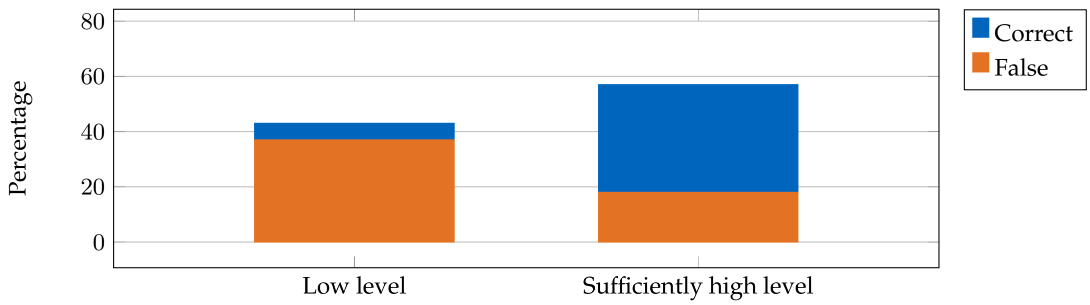


Figure 8.3.: Distribution of correct and false answers related to the scale *Abstract Thinking Competence* ($n = 134$).

8.2. Students' Deficits in the Competence of Abstract Thinking

Thurner et al. [93] 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.

The previous descriptive presentation of the results revealed items students often struggle with and indicate typical cognitive processes students do not manage. In this section students' results are analysed to confirm or to reject the hypothesis of a deficit among first-year students regarding the competence of abstract thinking.

Before starting to explore whether students have a deficit in the competence of abstract thinking or not, a basis for grading and a threshold need to be defined. There exist absolute and relative grading. When using absolute grading, the grading is defined by a point system. If the grades are determined based on relative grading, the students are usually ranked in order of their performance and a predefined percentage of students receive a certain grade [35]. As stated by Gronlund & Waugh [35] the choice between absolute grading and relative grading should be done in alignment to the institutions grading policies. In my case I will use absolute grading, as this is most common at the Munich University of Applied Sciences.

The next step is to define a threshold students need to pass in order to fulfil lecturers expectations. So far, there exists no common agreement on the number of points students need to achieve in order to pass an assessment, only suggestions. Gronlund & Waugh [35] make suggestions in the form of three grading schemes: first the assessment is failed when less than 60% of the points are achieved, second the assessment is failed below 65% of the points and third the assessment is failed below 75% of the points [35]. Bohn et al. [11] compare worldwide grading systems. When taking the lowest threshold in the international comparison, students should pass an assessment, when achieving equal or more than 50% of the points. This corresponds to the common practices at the Munich University of Applied Sciences or Technical University of Munich. Lecturers usually choose a 50% threshold in their exams. Thus, this threshold is used to evaluate students' results.

Figure 8.4 depicts an exploratory analysis of students' answers assessed on scale *Abstract Thinking Competence* using a heat map. The x-axis represents the percentage of answers that need to show a sufficiently high level of abstraction (threshold). For each student it is evaluated, whether he or she passed the threshold. Every entry in the heat map shows the percentage of students who exceeded the threshold. The figure reveals that 73% of the students achieve equal or more than 50% of the points and thus exceed the threshold. This indicates a rather small deficit among the cohort. However, this analysis solely focusses on a sufficient level of abstraction and not on the correctness of the abstraction build. By including this facet, the scale *Professional* has been derived. When analysing the thresholds of professional answers from 0% to 100% fulfilment, the picture is a different one.

100%	100%	99%	90%	81%	73%	47%	23%	5%	2%	0%
≥0	≥10	≥20	≥30	≥40	≥50	≥60	≥70	≥80	≥90	=100

Figure 8.4.: Exploratory analysis of students' answers showing a sufficiently high level of abstraction. Every entry in heat map shows the percentage of students fulfilling the expectations indicated on the x-axis ($n = 134$).

Figure 8.5 depicts an exploratory analysis for scale *Professionality*. When again using the 50% threshold, only 30% of the students are able to fulfil this expectation and build a correct abstraction. This analysis reveals a major deficit in students' initial competence. It is shown that a majority of students does not meet the expected prerequisites. 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. In order to examine the hypothesis of a positive effect of abstract thinking competence on the success in programming, the impact of a sufficiently high level of abstraction and the correctness of an abstraction on the grade in the module *Software Development I* is investigated in the following section.

100%	99%	87%	66%	47%	30%	10%	6%	0%	0%	0%
≥0	≥10	≥20	≥30	≥40	≥50	≥60	≥70	≥80	≥90	=100

Figure 8.5.: Exploratory analysis of students' professional answers. Every entry in heat map shows the percentage of students fulfilling the expectations indicated on the x-axis ($n = 134$).

The data reveal that students in computer science are not well prepared regarding their competence of abstract thinking. Building a correct abstraction on a sufficiently high level of abstraction seems to be challenging for them. However, correct abstractions form the basis for any concept in software development.

Several lecturers have been involved in the design of the ATA. The threshold has been agreed with them, too. Thus, the analysis reflects lecturers' expectations at the beginning of the first semester. If there exists such a significant gap between students' initial competence and lecturers' expectations, teaching does not pick students up where they are. Consequently, students might have problems to follow the lecture.

8.3. Impact of Abstract Thinking Competence on Grades

Key competences, such as abstract thinking, are said to be the basic prerequisite for acquiring programming skills. There exist several publications on the investigation of predictors for the success in programming, like [55, 75, 94]. To my knowledge, there exists only one publication analysing the impact of abstract thinking and success in programming.

Bennedsen & Caspersen [9] investigated the correlation between abstract thinking competence and programming ability. The programming ability is indicated by the final grade in the CS1 course. In order to measure the competence of abstract thinking, the authors used the "pendulum test" initially developed by Inhelder & Piaget [49]. This operationalisation is (even mentioned by its authors) not ideal. The pendulum test measures the cognitive stage of participants, while they demonstrate their ability to control independent variables in a reasoning task.

The authors examined the correlation between cognitive development and programming ability by conducting an empirical study. They only found a weak correlation between both, which is an unexpected result, as most experts state that the competence of abstract thinking is the key in programming. It could be the case that the competences necessary to complete this task are different or not exclusively the competence of abstract thinking. Thus, the authors requested further research.

As the ATA is an improved operationalisation of the abstract thinking competence, I investigate the relation between the results of the *Main study* of the ATA and the programming ability shown in the final exam of the first-semester module *Software Development I*. By performing this analysis, the initial research question *Does the initial level of the abstract thinking competence have an impact on the final exam in software development?* is answered. It has to be added that the analysis is based on the data of only the 33 students I taught myself in winter semester 2017/18. All other students had other lecturers, so their grades were not available to me, due to data protection directives.

The majority of questions in the end-of-term exam on *Software Development I* requires writing Java code to solve given problems. Moreover, students are asked to compare two solutions and give advise regarding the selection of a concept or a solution with respect to given context. Thus, the exam specifically addresses the higher levels of Bloom's taxonomy (analyse, evaluate, create). In addition, some few questions solely require the lower three levels remember, understand and apply.

For those students who participated in the end-of-term exam, the following three diagrams (Figures 8.6, 8.7, and 8.8) show the relation of the grade they received and the points they achieved in the *Main study* according to the three dichotomous scales *Correctness*, *Abstract Thinking Competence* and *Professionality*. Visualising the students' data provides interesting insights.

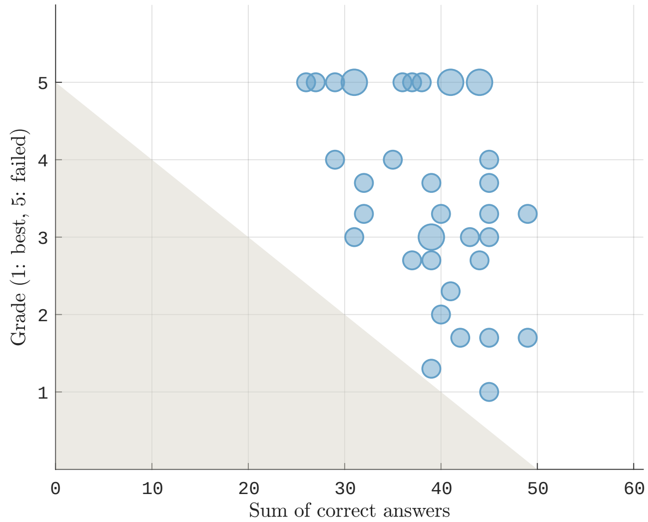


Figure 8.6.: Relation between number of correct answers and the grade achieved in the end-of-term exam ($n = 33$).

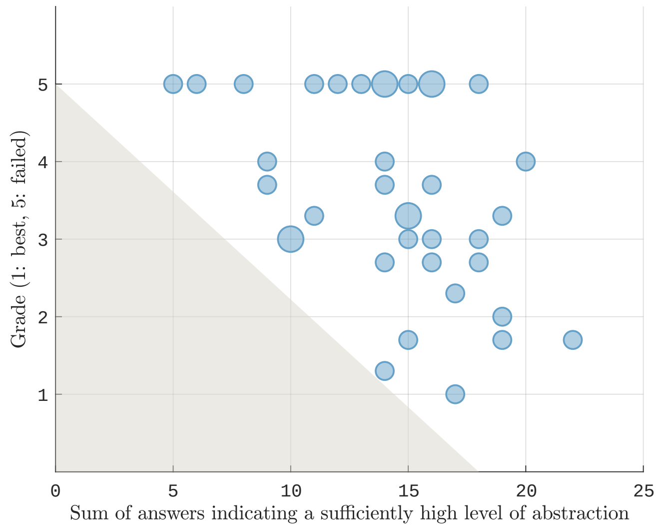


Figure 8.7.: Relation between number of answers indicating a sufficiently high level of abstraction and the grade achieved in the end-of-term exam ($n = 33$).

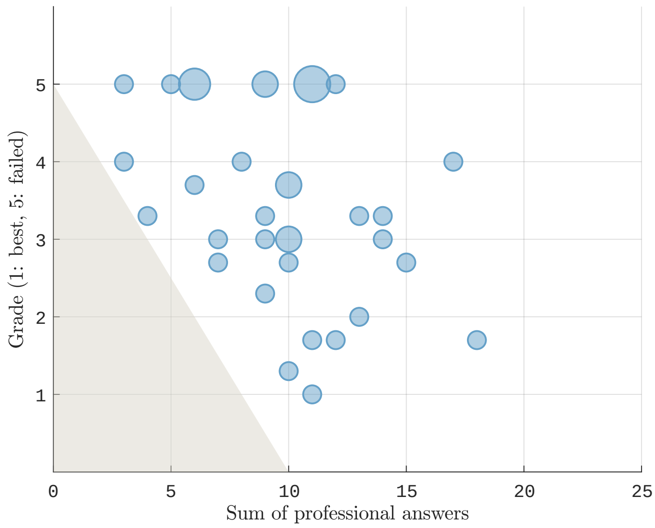


Figure 8.8.: Relation between number of professional answers and the grade achieved in the end-of-term exam ($n = 33$).

There are no students who scored low in the ATA (independent of the scale), but still achieved a good grade in the exam, as highlighted by the gray triangle in the lower left area of the plots. The correlation becomes most clear in Figure 8.7 showing the relation between the grade and the scale *Abstract Thinking Competence*.

Pearson's correlation coefficient *cor*, measuring the strength of a linear relationship between two variables, leads to the results depicted in Table 8.2. As expected and indicated by the plots, the correlation between students' grades and the scales *Correctness* as well as *Abstract Thinking Competence* is moderate, but statistically significant. Scale *Professionality* correlates only with weak evidence with students' grades in the end-of-term exam.

Scale used to correlate grades with	Correlation	Significance
Correctness	$cor = -0.46$ moderate relationship	$p = 0.004$ significant
Abstract Thinking Competence	$cor = -0.46$ moderate relationship	$p = 0.004$ significant
Professionality	$cor = -0.37$ weak relationship	$p = 0.024$ not significant

Table 8.2.: Overview of strength, direction and significance of the correlations ($n = 33$).

These results clearly indicate that a sufficiently high level of abstraction is necessary for acquiring software development skills, but not sufficient. That the competence alone is not enough is not surprising due to the numerous key competences, which are said to be perquisites for acquiring software development knowledge and skills. Thus, the well-known hypothesis that the key competence of abstract thinking influences the acquisition of programming skills is confirmed.

Summary of findings in Chapter 8

There exists a relationship between *Correctness* and *Abstract Thinking Competence*. Statistical analysis shows that the relationship is strong and statistically significant. Thus, a sufficiently high level of abstraction helps to build correct abstractions. Students demonstrate only a small lack when building abstractions on a sufficiently high level. However, investigations revealed a deficit in students' *Professionality*, so in the combination of *Correctness* and *Abstract thinking Competence*. Further analyses have shown that a sufficiently high level of abstraction is necessary to acquire computer science specific knowledge and skills, but is not sufficient.

Part IV.

Training

Teaching Abstract Thinking Competence

Training

The analysis of the results of the ATA has shown that the competence of abstract thinking is an essential prerequisite for the acquisition of professional software development skills. Moreover, a deficit in abstract thinking competence of first-year students is indicated by the data. Thus, teaching professional software development skills should additionally address the competence of abstract thinking explicitly. Hence, the last part of this work is to propose a concept to promote the students' competence of abstract thinking integrated into a class on software development.

Software development consists of two tasks: modelling and implementation. While modelling, the competence of abstract thinking is highly relevant. During the process, common features are identified and differences are determined. In addition, relationships are created between objects or processes so that they can then be implemented in software. In particular, modelling tasks requires higher levels (analyse, evaluate, create) of Bloom's learning taxonomy [5] to a large extent. In contrast, implementing an existing model is largely limited to competences at the lower three levels (remember, understand, apply). Publications, such as Nguyen and Wong [66] or Or-Bach and Lavy [68] demand to put more emphasis on modelling in teaching. Additionally, Bucci et al. [15] as well as Hazzan and Kramer [43] state to do so right from the beginning.

From all these findings, I conclude that teaching software development needs to put more emphasis on modelling and the contained abstract thinking competence right from the beginning. Moreover, it could be helpful to clearly separate modelling and implementation.

Literature suggests numerous concepts for teaching the competence of abstract thinking. Several of them are integrated into the context of software development. Concepts and approaches from literature are discussed in Chapter 9.

One of the first lectures in which students come into contact with software development and abstract thinking competence is the module *Software Development I* in the first semester. Thus, Chapter 10 presents the revision of the teaching concept of this module with regard to the competence of abstract thinking. On the one hand side, the redesign deals with the preparation of the teaching units, the material used and how they have been put into practice (see Section 10.1). This process considers findings from the competence model (see Chapter 2), the analysis (see Part III) and the literature review (see Section 10.2). On the other hand, the teaching style has been changed from a traditional approach to pair-teaching (see Chapter 10.2). In the end, the teaching concept is evaluated in Section 4.4.

9. Literature Review: Lecture Design and Teaching Approaches

Current teaching practice in introductory courses on software development primarily focusses on concrete programming issues [78]. A reason for this could be that many lecturers assume students will grasp the essence of concepts such as abstraction through the lecture implicitly [43].

This, in turn, might be affected by the fact that competences like abstract thinking competence are never applied out of context. When teaching them, another topic is addressed at the same time [43]. In the context of abstract thinking competence, Hazzan and Kramer suggest to integrate the competence into several courses and to mention it on different occasions, rather than to create a dedicated course on abstract thinking [43]. For the realisation of such courses, there exist recommendations on the overall design of lectures, as well as suggestions for specific approaches to promote students' competence of abstract thinking.

A general decision in lecture design is the choice of examples. Lecturers should put more emphasis on this. Examples are a critical point, as good examples can facilitate students' understanding of concepts by sharpening the meaning and its application [68].

Regarding the overall structure, there exist recommendations to combine design and implementation tasks in one lecture, as there exists a strong relationship between design and implementation [68]. Since the abstract thinking processes mainly occur during modelling, this approach is promising and in my opinion necessary to promote students' abstract thinking competence.

Moreover, there exist two different concepts in literature that have a strong focus on modelling: One concept introduces mathematical modelling right from the beginning. According to the authors, mathematical models allow precise specifications independent of implementation details [15, 23]. Another concept solely focusses on modelling and postpones implementation details and topics like formal loop semantics, data collections or complexity analysis to some later point in the curriculum. The approach focusses on abstractly decomposing problems and expressing ideas while using design patterns throughout the lecture. Thus, the authors call this realisation an "objects-first-with-design-patterns" approach. They see patterns as a very tangible instance of good abstractions and as a "meta-level between concrete objects and difficult abstract concepts". Additionally, design patterns provide a standard vocabulary to discuss abstractions [66].

More specific approaches suggest that students should learn to identify categories and their characteristics, which are relevant to a problem, right from the beginning [84]. This approach is aligned with the competence model of abstract thinking and becomes highly relevant during the object-oriented design process.

Furthermore, existing abstractions should be presented to the students, before they have to design their own abstractions [68]. A more concrete realisation of this approach is to use component libraries in the course, as those are abstractions by definition [63, 84].

Another approach proposes to work out the differences between novices' and experts' solution processes and to explicitly address them in the lecture. While doing so, lecturers can focus on the solution process itself or on the resulting abstraction [68]. Both approaches can be used to motivate the necessity of abstractions in the context of computer science. Moreover, they help to make the relevant thinking processes and decisions transparent for the students.

Some modelling decisions are related to the purpose of the abstraction. Consequently, there exist recommendations to consider the purpose of an abstraction directly and explicitly in class. Furthermore, lecturers and students should discuss whether an abstraction fits a purpose when evaluating solutions of activities like modelling or problem solving [44]. Meyer [63] suggests that students are encouraged to adapt components, such as classes, to new purposes.

At the end of a module that promotes the competence of abstract thinking, students should, for example, be able to identify abstractions as a central concept [43]. And as professionals, students should be able to switch between abstraction levels easily [41].

In summary, there exist several ideas on how to promote students' competence of abstract thinking in literature. However, most approaches have never been put into practice. Thus, practical experiences, students' feedback or empirical data is often missing. Hence, teaching abstract thinking remains an issue in educational research.

Table 9.1 shows a summary of the teaching concepts and approaches proposed, including a rating, whether they are assessed positively or negatively concerning the requirements of the module *Software Development I* (*Module aligned*), as well as if they have been put into practice (*Realisation*) and evaluated (*Evaluation*). For example, *objects-first-with-design-patterns* is an interesting concept, but does not match the module description of the intended course, whereas *presentation of existing abstraction* sounds promising and fits the module description.

All approaches that are in line with the module description of *Software Development I* are considered for the teaching concept developed in Chapter 10, despite the fact that these approaches have not been put into practice so far.

Approach	Module aligned	Realisation	Evaluation	Reference
#1: OO wih patterns objects-first-with-design-patterns	-	+	-	[66]
#2: Mathematical models usage of mathematical modelling	-	+	+	[15, 23]
#3: Examples choice of examples	+	-	-	[68]
#4: Design and implementation combination of design and implementation	+	-	-	[68]
#5: Novice and expert comparison of novice and expert solution processes and their results	+	-	-	[68]
#6: Existing abstractions presentation of existing abstractions and using them, like libraries	+	-	-	[63, 68, 84]
#7: Identifying enities identification of categories and their characteristics	+	-	-	[84]
#8: Purpose focus on the purpose of an abstraction	+	-	-	[44, 63]

Table 9.1.: Summary of recommendations and teaching approaches and their evaluation.

10. Concept for the Module Software Development I

The teaching concept presented in this part has been developed for a first-semester class in the bachelor's programme in computer science. More specifically, the concept is integrated into the 8 ECTS module Software Development I[1]. The course consists of four hours of lecture and two hours of lab session attendance time per week. During the lab sessions, students have to solve programming problems. Additionally, lecturers of this module expect students to work around 8 hours per week at home. This module is similar to the well-known course CS1 [91] and addresses central concepts of object-orientation as well as topics such as control structures and arrays. In the end, students are able to implement simple algorithmic solutions using an object-oriented programming language. Moreover, students can analyse a problem statement and identify the steps necessary to solve the problem. Furthermore, they are able to choose an appropriate concept and implement this in a syntactically and semantically correct way.

First-semester students at the Munich University of Applied Sciences are usually split into three study groups. And each of this groups can have a different lecturer in the different modules. This was also the case in winter semester 2017/18 were the concept has been put into practice. Consequently, the scope of possible changes is limited. The topics taught in the module are specified by the module description and agreements between lecturers of one specific module.

In recent years, an *objects-first-test-second* [92] approach has been introduced in this module. Due to the positive experience gained when using this approach, and to achieve comparability of different years with regard to the lecture evaluation, the established approach is retained in the new teaching concept.

Hazzan and Kramer [43] suggested to teach the competence of abstract thinking integrated into a lecture, as the competence is not exclusively connected to a specific topic and is never applied out of context. Thus, topics in the module *Software Development I* that require the competence have been worked out. Whenever abstract thinking competence has been identified, it has been explicitly picked out as a central topic and presented as learning content. Consequently, the new concept additional addresses the following learning outcomes:

– Students differentiate between abstractions and entities.

[1] Further details can be found in `https://w3-mediapool.hm.edu/mediapool/media/fk07/fk07_lokal/studienangebot_3/studiengaenge_neu/docs/ifb/module_pdf_IF_2018_WS.pdf`.

- Students build abstractions and identify their characteristics based on a set of specific entities.
- Students choose characteristics with respect to the purpose of the abstraction.
- Students differentiate between modelling and implementation.
- Students recognise, where and when an abstraction is required.
- Students decide upon one of several given abstractions with respect to the given problem and purpose of the abstraction.

In order to make abstract thinking transparent in teaching, implicit expert thinking processes have been analysed while designing the teaching concept. For this purpose techniques such as think-aloud [71] and decoding [69] have been used. Due to the positive experiences, the processes revealed have not only influenced the design of several teaching units, they were also prepared to be presented to the students explicitly.

During the design of the concept, several aspects that make learning how to code difficult, have been identified. A detailed and thorough description of how they have been deduced is beyond the scope of this work. However, the difficulties identified are listed, as they also guided the design of the concept.

- The content in software development is exclusively abstract.
- Novice programmers need to adapt to an unfamiliar way of thinking right from the beginning.
- Concepts in software development are highly complex and highly interconnected.
- Software development requires a high degree of accuracy, but at the same time offers many degrees of freedom.
- Novice programmers face 'Magic' very often, for example in the signature of the `Java main`-method. Students need to use it to get their code running, but initially do not understand all concepts involved, like access modifiers.

All these information influenced the concept taught in winter semester 2017/18. I designed and taught the lecture on *Software Development I* together with Prof. Dr. Axel Böttcher for one study group of three. The lecture took place Tuesdays and Thursdays. An overview of the topics covered in winter semester 2017/18 and their sequence is depicted in Table 10.1. The design process of four teaching units focussing on the competence of abstract thinking is described in Section 10.1. The units discussed in detail are highlighted in Table 10.1. Please note that this is just an extract. There exist further topics, such as control structures, that require the competence of abstract thinking to a high extent and that we revised in the course of this work.

Week	Lecture Tuesdays	Lecture Thursdays	Lab
1	– Organisational issues – **Modelling classes: Attributes** – Classes in Java	– Instances using `new` – `main`-Method – Modelling classes: Methods	Introduction to the IDE

2	– Constructor
	– `this`
		– Wrap-Up
		  – **Π of Abstraction**
		  – Where is the mental performance?
		  – Misconceptions identified
		– Associations
			– Introduction to Scratchpad
			– **Ogre-Lesson**

3	Unit-Testing	Expressions and Instructions	Ogre-Head as component
4	*Holiday*	If-else statement	Testing Ogre-Challenge
5	Wrap-Up: Expressions and instructions, strings	Reflection: What makes software development so difficult?	Implementation Ogre-Challenge
6	– Loops – Testing loops	– Algorithms – Helper-Methods	Rectangles – Intersecting or not?
7	– Implementation of concrete algorithms – `static`	Information Hiding	Castle-Builder: drawing brick by brick
8	Implementation of concrete algorithms	Linked-Lists (Theory, not Java API Class)	**Creating a maze using the hunt and kill algorithm (Modelling)**
9	Exam preparation	Interfaces	Creating a maze using the hunt and kill algorithm (Implementation)
10	Inheritance	– State, equals(), copy – abstract classes	Represent trains by implementing a linked list from scratch
11	Experts decision between inheritance, interface and abstract class	Christmas Lecture	Calculating the change of a ticket machine
12	*Holidays*	*Holidays*	Calculating the change of a ticket machine
13	*Holidays*	*Holidays*	Calculating the change of a ticket machine
14	Exceptions	– Varargs – Immutables	Creatures
15	Exam preparation	Exam preparation	Creatures

Table 10.1.: Overview of the topics taught in the course and their sequence. The teaching units focussing on the competence of abstract thinking that are presented in Section 10.1 are highlighted.

10.1. Teaching Units

The literature review regarding lecture design and teaching approaches revealed several ideas that seem promising to promote abstract thinking competence in the module *Software Development I*. The content of the module is given by the module description. Moreover, the overall structure is defined by the well-established concept of *objects-first-test-second*. Thus, only the way the programming concepts are taught is changed. The development of four teaching units is exemplarily presented in this section.

Each unit considers theory presented in Chapter 9 and findings of this work. Additionally, they are aligned with the intended learning outcomes. The fundamental considerations that influenced the design of each unit are presented in the following subsections. I refer to approaches described in Chapter 9 by using *approach #*. After the theoretical basis has been described, I depict how each teaching unit has been put into practice. This description contains main decisions, actions in the classroom or instructions in the lab. Concrete actions and instructions are highlighted by a vertical blue line at the left margin. Thus, the hurried lecturer can just read through the highlighted parts to get all necessary information to use the teaching unit.

10.1.1. Modelling Classes: Attributes

Introducing the concept of classes in the context of object-oriented design and implementation offers numerous opportunities for the application of the findings from my research and recommendations from literature. Prior to the in-depth design of the lecture, my colleague and I spent a lot of time on choosing an appropriate example (*approach #1: Examples*). For this selection, we defined the following requirements: The example ...

- can be used at several points in the lecture.
- works without control structures.
- has visualisable entities.
- uses only simple mathematical basics.
- fits into students' daily life.
- is not contrary to current research.

Fractions are a commonly used example in textbooks and lecture material. However, our experience and previous research [104] has shown that students lack in the mathematical skills necessary to deal with this example. As a consequence, this example was excluded from further considerations.

Persons are another example frequently used for introducing the concept of classes. However, using real persons to introduce classes is not the best choice, as this encourages the development of the misconception described by Sorva [83]. Persons always have names and concrete objects are stored in variables, which also have names, this can lead to great confusion. The attribute called `name` representing the name of a person and the name of the reference variable a specific instance is assigned to, is difficult to distinguish for novice programmers.

Other inspiring examples are heroes, cameras and books, which can be found in *Schrödinger programmiert Java: das etwas andere Fachbuch* by Ackermann and Leowald [1]. Cameras seemed to be a good example for the lecture, but all methods described require control structures. This also applies to heroes and books. As we adopt the concept of *objects-first-test-second*, students do not know any concepts of procedural programming, when we introduce the concept of classes. Thus, methods must not require control structures such as if-statements or loops. In a reasonable time frame, we were not able to find adequate methods for a class representing cameras that do not require control structures. Thus, heroes, cameras and books do not fulfil all requirements.

Consequently, we needed to come up with another example fulfilling all our requirements. After long discussions and studying textbooks, the idea of flats appeared. First-year students often just looked for a flat, and thus, know the characteristics very well. As the example originates from students' daily life they can identify with the example. Moreover, flats can be easily visualised in different ways to make the entities tangible. And the biggest advantage is that methods like the calculation of the rental price based on the apartment size and the price per square metre just require simple mathematical operations and do not require control structures. Additionally, we did not find any objections from literature concerning flats as an example. Furthermore, experiences have shown that the idea of flats supports several further topics, like testing, associations and even lambdas.

An overview of examples, their sources and objections are shown in Table 10.2. We are aware that there exist many more examples and several of them also fulfil the requirements. However, we stopped our investigations when we found an appropriate example.

Example	Source	Objection(s)
Fractions	Lecture material	Requires more than mathematical basics, objection from literature exist in Zehetmeier et al. [104]
Persons	Lecture material	Objection from literature exist in Sorva [83]
Heroes	Ackermann and Leowald [1]	Requires control structures
Cameras	Ackermann and Leowald [1]	Requires control structures
Books	Ackermann and Leowald [1]	Requires control structures
Flats	Discussion and considerations	

Table 10.2.: Overview of the examples investigated, their sources and their objections.

> We chose flats as a sound example for introducing the concept of classes.

The basis for each abstraction process are entities, often also referred to as 'objects of the real world'. In order to establish this basis, several entities have to be collected.

> Students had to describe their favourite flat. We collected the entities and their details on the blackboard.

From the analysis of the items VEHICLES and LEGO BRICKS[2] of the ATA one can see that students most frequently use components and attributes to characterise objects. This observation is again confirmed by the collection of characteristics during the lecture.

> Students named components like bathroom or garage, as well as attributes such as accessibility to public transport or size of $90m^2$. Furthermore, students mentioned vague and non-quantifiable characteristics, like 'cheap' or 'good' location.

Having created the basis, the first abstraction process can be performed together with the students. In this case, the focus is on processes of the competence models' subcomponent *Commonalities and Differences*. This procedure is supported by *approach #2: Identifying enities*.

> Students had to identify and summarise common characteristics of the flats and to choose an appropriate name for the resulting attributes.

Several additional aspects relevant to the abstract thinking process have been revealed during the derivation of the competence model (see Section 1.2). I agree with *approach #3: Purpose* that the purpose is highly relevant and thus, should be addressed in teaching. Especially those processes of the subcomponent *Hide & Keep* are influenced by the purpose. So far, this aspect was not explicitly addressed in the module *Software Development I*. Consequently, this facet has been integrated into the design of the module right from the beginning.

> Two situations are presented to the students: calculation of a flat's rental price is something students are aware of. But the calculation of the property tax is unusual for them, although it is of practical relevance. They had to choose the important attributes for both purposes. We discussed the selection process, to make the decision process transparent.

The resulting picture on the blackboard illustrates two abstractions (*approach #4: Examples*) that differ because of their purpose (*approach #5: Purpose*). This way of teaching was repeated several times during different lectures. In the lab exercises, students were always reminded to think of the purpose of the current problem.

[2]Example item is shown in Figure 4.5 and the results of the analysis are presented in Figure 7.5.

10.1.2. The Π of Abstraction

By now, students had a lecture on modelling characteristics and behaviour, read about primitive data types and are able to implement simple `Java`-classes. Thus, *approach #6: Design and implementation* has been implemented. Moreover, we present an existing abstraction in the form of a `Java`-class to the students (*approach #7: Examples*).

> During the lecture, we introduced all necessary concepts regarding `Java`-classes, attributes and methods and developed a proper `Java`-class representing a flat, based on the abstraction *flat* defined so far.

In order to make the development process described previously more comprehensible and tangible for the students, we illustrated a conceptional model of the process. The result is the 'Π *of Abstraction*' depicted in Figure 10.1. It is called Π as the three arrows form the shape of a Π. The depiction clearly shows that the competence of abstract thinking is especially important while modelling.

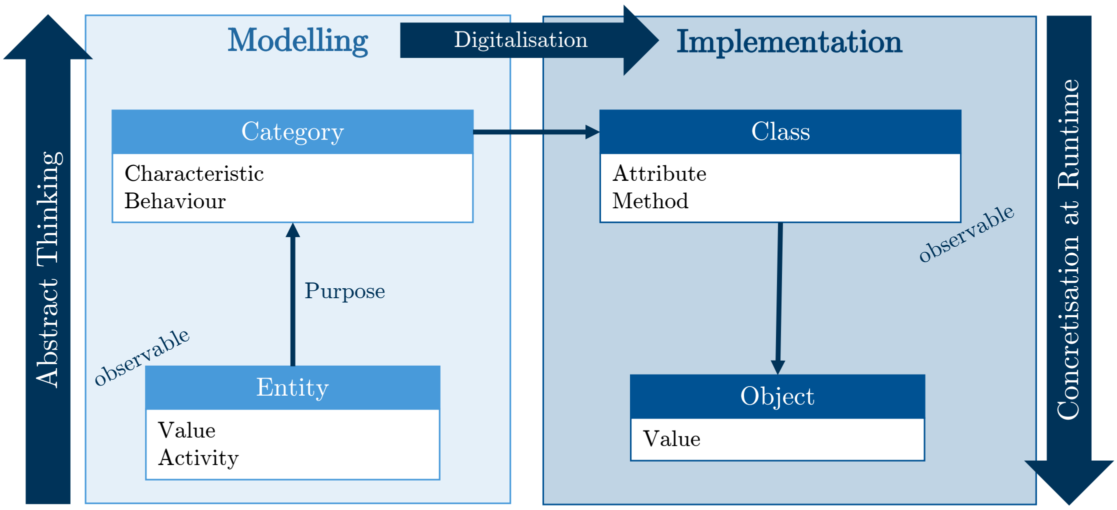


Figure 10.1.: Π of Abstraction for object-oriented design. The three arrows forming the shape of a Π gave it its name.

The left part of Figure 10.1 shows that observable entities of the real world are summarised and normalized, in order to build an abstraction, we call category, representing all underlying entities. During this process, abstract thinking competence is highly important. The purpose of the abstraction plays a significant role, too. This process can be carried out without any software specific knowledge.

In the next step, the category is translated into source code of a specific programming language (arrow digitalisation). And at runtime, objects representing concrete entities are generated.

The Π *of Abstraction* (see Figure 10.1) reveals an interesting relation. Entities are observable while modelling, but their equivalent in software, namely objects, are invisible as they are created at runtime. The same effect applies to categories and classes. Whereas classes can be observed in the code, categories built based on the entities are invisible mental models.

Explicitly, presenting this process to the students and splitting modelling and implementation, was designed to promote students abstract thinking competence. This should also help to make the single steps more transparent to the students. Moreover, the Π *of Abstraction* can serve as a tool students can use to reflect and communicate at which point in the development process they are facing problems.

> In the Wrap-Up of the thematic block on `Java`-classes, we presented the Π *of Abstraction* to the students in order to summarise the content learned so far.

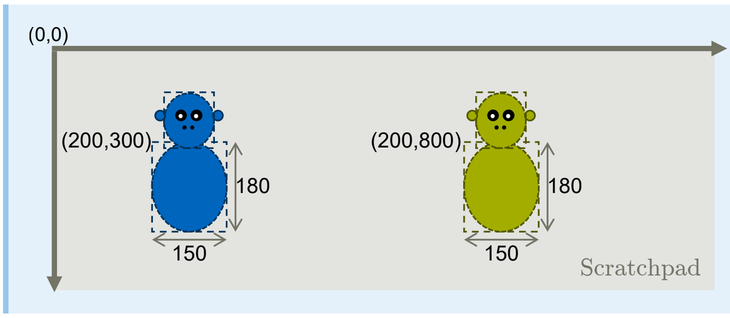
A next step could be to introduce a common language to describe categories in a systematic way. Visual representations could help students to make abstractions more tangible. This could be done by using Unified Modelling Language (UML) or any other (semi)-standardised language that is suitable to describe abstractions. This approach would be similar to *approach #8: OO wih patterns*, but less drastic.

10.1.3. Ogre-Lesson

In several teaching units, students were guided along the process described in the Π *of Abstraction* (see Figure 10.1). The first lab-assignment realising this approach was the Ogre-Lesson implemented using Moodle-Lesson. The detailed description of the development of this teaching unit serves as an example of the implementation of this approach.

At first, we decided to use graphical representations of objects instead of generating an output on the command line using `System.out.println()`. The reason for this is that students then tend to mix up the concept of return values and information printed. Thus, we provide two classes `Colour` and `Ellipse` to the students. With the help of these classes, it is possible to draw ellipses on a canvas. In order to familiarise students with their usage as well as the methods provided, we gave a short introduction to the so-called Scratchpad before starting the Ogre-Lesson.

> The goal of this teaching unit is that students write a proper `Java`-class representing Ogres (for an example see figure below) in varying colours at different x- and y-coordinates using the classes `Colour` and `Ellipse`.



Following the Π *of Abstraction*, the first step is to collect or construct similar entities.

Students draw a single ogre within the `main`-Method using hard-coded coordinates and sizes to describe the ogres' components.

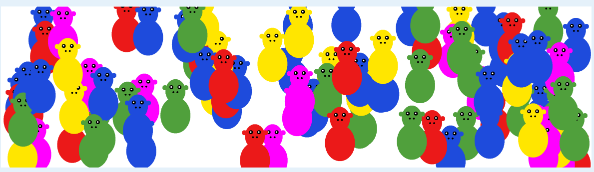
Component	x-coordinate	y-coordinate	width	height	color
body	200	300	150	180	green
head	225	210	100	120	green
left iris	245	245	20	20	black
left pupil	250	245	8	8	white
right iris	285	240	20	20	black
right pupil	290	245	8	8	white
left ear	210	240	25	25	green
right ear	315	240	25	25	green
left nostril	260	270	10	10	black
right nostril	280	270	10	10	black

One entity is not enough to point out the necessity of building an abstraction. Thus, students construct at least one more entity by changing the characteristics.

Students draw a second ogre in a different colour and at another position within the `main`-method.

In order to motivate the abstraction process even more students are asked whether they want to repeat this process of copy, paste and adjust more than 50 times.

Would you like to draw this meadow full of ogres using the process you have performed so far?



At this point, the advantage of Moodle-Lessons comes to the fore. This activity allows asking questions and controlling students' way through the lecture individually. Lecturers can, for example, predefine whether the student is permitted to continue, has to read (additional) material or has to repeat the task. At this point in this unit, students should realise that building an abstraction representing more than one entity is a profitable solution. If not, they are guided to a task to rethink their decision.

Generating entities within the `main`-method by copying and adjusting the relevant details is a procedure we often observe in novice solutions. Using abstract thinking and building an abstraction that summarises similar entities, parametrises their differences and allows to create new entities is how experts in software development solve the task. Thus, students are guided from a single entity created on purpose, over several entities created by copying and adjusting representing a common novice solution, to the experts' solution actually an abstraction. This procedure is a realisation of *approach #9: Novice and expert*.

> **YES**: Re-think your answer
> - How much time do you need to create 50 ogres or even more?
> - Would you really like to do this until next week?
>
> **NO**: Good decision! In computer science, we use abstractions to represent several similar entities. Go ahead.

The next step is to perform the abstraction step and to build a category representing all entities. Additionally, important characteristics all entities share are collected. When considering the purpose of the abstraction, the relevant characteristics can be selected.

> Collect characteristics all ogres share and select those necessary to draw them. Build a category based on your collection. Determine if the characteristics have an equivalent value for all ogres or if the values change from one ogre to the other.

After the category is completely specified, students can perform the digitalisation step and translate their model into a proper `Java`-class. This process is again supported by several guiding questions. For every part of the model (category, characteristic and behaviour) students were asked to choose the appropriate programming concept. If they gave a wrong answer, the relevant information from the lecture was shown. After studying those, students were able to repeat the task.

10.1.4. Maze Task

Up to now, the teaching units mainly dealt with the static aspects of the entities and their translation into code. Now, the dynamic aspect is in focus. The teaching unit was designed as a lab assignment and took place in the sixth week of the first semester.

Students have a good basis in object-oriented design and gained first theoretical and practical experiences in using concepts of procedural programming. In the next step, we wanted students to model and implement a non-trivial algorithm. As our overall goal was to put more emphasis on the modelling phase right from the first semester, we designed this unit where students build a model of all necessary parts first (*approach #10: Design and implementation*).

As building the data model and specifying the behavioural aspects of the algorithm is the first complex modelling task for the students, we wanted to make the experts' thinking processes as transparent as possible. For this purpose we used guiding questions [103] at decision points of experts' thinking and provided intermediate results, to get students back into the experts thinking process whenever necessary. This should help students to develop their idea of the concept.

Students had to document and turn in their answers to the guiding questions. Only then, they receive our ideas to compare their results with. The uploaded documents provided insights into the students' thinking processes and revealed their mental pictures. Furthermore, we were able to derive results regarding the students' level of abstraction. A detailed evaluation of four tasks of this unit using the rating scheme proposed in this work can be found in [101].

There are many challenges in the design of such a unit. A major one is to find a reasonably complex algorithm to make the advantages of *modelling before implementing* visible. Moreover, think-aloud sessions need to be conducted to define the experts' solution process presented to the students. The interviews conducted to design this teaching unit revealed several correct but different approaches to solve the problem. Thus, it is important not to over-restrict the process presented to the students, as there exists more than one successful solution process.

After extensive research and discussing several algorithms, we decided to base the unit on the implementation of a maze-generating software using the hunt-and-kill- algorithm described in [16]. The maze-generator "moves" on a grid of cells. Each cell has a right and a bottom wall (cf. Figure 10.2). A maze is generated by erasing walls according to given rules.

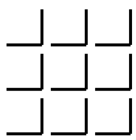


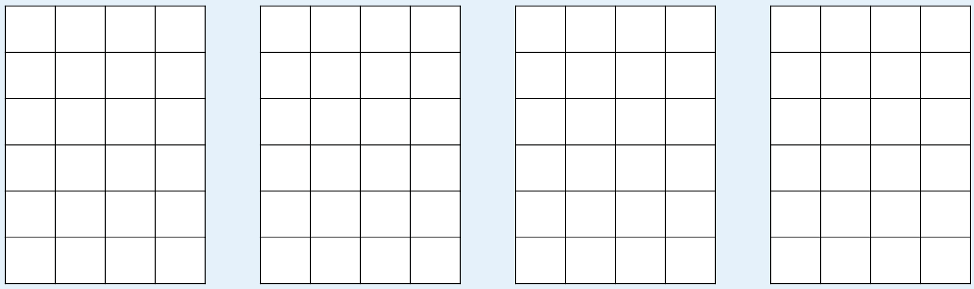
Figure 10.2.: Field of cells.

Jamis Buck describes the algorithm as follows:

1. Choose a starting location.
2. Perform a random walk, carving passages to unvisited neighbours until the current cell has no unvisited neighbours.
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. 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 [16] describes this algorithm based on some subtle design details.

The first task should help students to get familiar with the process of the algorithm by applying it using pen and paper.

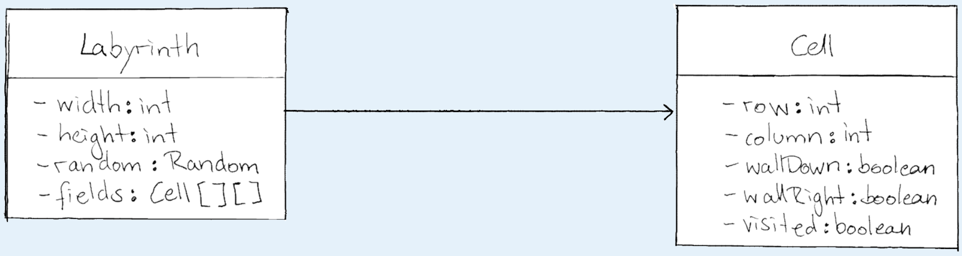
> Erase walls according to the algorithm, in order to build four mazes. Use the given drawings.
>
> 

While utilising this experience, students can name components and characteristics, which build the foundation for the abstraction.

> – Which characteristics does every maze have?
> – Which are the components every maze consists of?
> And which characteristics do they have?
> – What can you say about the arrangement of the components?

Based on these considerations and the description of the algorithm provided, students then deduce a data model.

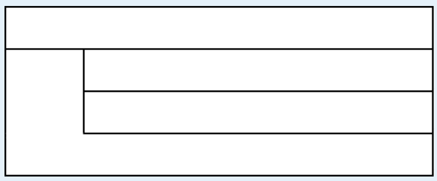
With the help of the preliminary considerations, you should now have identified all entities, their relevant characteristics and their associations. On this basis, you can now draw the corresponding UML class diagram of all classes involved, as well as their attributes and associations.



Data model presented to the students as an intermediate result.

After presenting the data model to the students, they worked on the algorithm. Step by step they were guided from an high-level perspective describing the interaction of the major steps towards the algorithmic details.

Identify the essential steps of the *Hunt* and *Kill* algorithm and depict their sequence by using the following structogram:



The algorithm contains steps that occur repeatedly. Moreover, those steps contain most of the algorithm's logic. Thus, we guided students, to work these steps and their internal functionality out.

Which actions occur repeatedly?

After another intermediate result, which represents our experts' solution, students are asked to visualise several states of the data model.

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.

> Which special cases can you identify for other starting cells S? Please draw them.
> *(The expected solution is shown in gray)*

S		

	S	

These visualizations should help students to deal with the following two open format questions, regarding the algorithmic details.

> How can you identify adjacent cells on the grid?

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

	B	
	A	

	A	B

The model derived in this unit should guide students through the subsequent implementation unit.

This was only an extract from the concept we realised in the winter semester 2017/18. Just to mention a few more examples: We additionally introduced associations in the second week of the semester, which is much earlier than usual. This decision was based on the insights gained by the analysis of the data collected using ATA. Moreover, during the Ogre-Challenge, students used a library again, to learn how to work with existing abstractions. Thus, students were also provided with a good example of an abstraction (both *approach #11: Existing abstractions*).

10.2. Teaching Style

So far, teaching material, classroom actions and lab instructions have been in focus of the concept. In addition to this, my colleague and I also changed the teaching style from a traditional one to pair teaching.

The presentation and discussion of topics by two people is common practice in several fields, like educational TV-shows, debates, or in interviews. Moreover, working in pairs has become a traditional approach in the field of software engineering and is called pair programming. As these practices have proven highly efficient, we wanted to transfer these approaches into our teaching practice.

During the cooperative teaching of our class on *Software Development I* we experimented with various roles and their pairing. Pair teaching can occur in several constellations, as the lecturers can take different roles. From our perspective, each formation has its own intended use, strategies and thus advantages and disadvantages. It is important that each role is personalized by a lecturer [6]. We also found a couple of anti-patterns that should be avoided. From our point of view, the fundamental success factors that must be given are that both persons involved see each other as peers and both must be committed to the success of the module. Additionally, they should have similar didactical knowledge and teaching experience.

As computer science is built on a solid theoretical basis like many other disciplines, it is necessary to show novice programmers the differences between the scientific concepts and the concrete implementation (*approach #12: Design and implementation*), as they need to master both later on.

Theorist vs. practitioner according to [14]. This is the role distribution for the classical situation where students learn the theory and apply it later. In this case, one lecturer takes the role of the practitioner and the other acts as a theorist. The practitioner works on a real problem that is adjusted to the students' level of knowledge. The theorist teaches the theory and points to the theory students learned during the lecture/module.

When considering computational thinking [99] and cognitive apprenticeship [22], we came up with two more pairs, which are focusing on education in computer science but might be transferable to other disciplines.

Real world vs. software orientation. In their later profession, students need to deal with non-computer scientists as domain experts in order to collect requirements and transfer them into software. In this constellation, the process of translation from real-world abstractions to programming concepts is accompanied by one person focusing on the real-world domain and one focusing on the software-based representation of that domain. Equivalent to the case above there exists the role of a practitioner which is in our case a software engineer. The pair is completed by a person that concentrates on the concrete real-world entities the software is intended to represent later.

Computational thinking vs. implementation. Another important aspect of teaching computer science is to focus on processes represented in computational thinking and to consider it independent of a concrete implementation. In this case, both roles need to have a background in software development. One focuses on identifying general concepts like categorization, repetition, or discrimination of different cases in order to solve a real-world problem. During this process, he or she demonstrates the application of key competences like logical or abstract thinking and highlights evaluation criteria. The other transfers the abstract concepts into appropriate constructs of a specific programming language. For both roles, it is important that they make their thinking tangible and highlight decision-points as expressed in the framework of cognitive apprenticeship.

While teaching, it is not always necessary to take contrary or complementary positions. Pair thinking also happens if both peers act on the same level. This could help to identify misconceptions, interpretation of students' questions as they are not also clear. Another advantage is that the students have access to two perspectives and explanations. After the lecture, both can reflect on the lecture in order to improve it for the upcoming semester, to identify content that needs to be repeated or to plan the next steps.

Anti-Patterns. Especially during the teaching phase, the pair needs to beware of typical anti-patterns. Often one does not intend to make this happen, but they might occur. A typical example is that one takes the role of a professor or lecturer and the other one acts as a student or even worse an airhead. This intellectual hierarchy lets the second person appear non-professional. Another anti-pattern might arise from the job hierarchy. The supervisor or professor should not act with her or his employee e.g. Ph.D. candidate as they are not on the same level. However, from personal experience, we immediately recognized this pattern and planned the lectures in a way that this would not show up.

10.3. Evaluation

The concept has been evaluated at two points of the semester: a reflection lecture took place six weeks after the semester started and the university-wide course evaluation was conducted at the end of the semester.

During the reflection unit, we guided students to think about the course, its content and their difficulties. Students were asked to think of obstacles that currently reduce their speed of learning and reflect on characteristics why software development is difficult for novices. About one third of the students named difficulties concerning the typical thinking processes in computer science, accuracy and complexity. Students reported that they are aware of thinking processes, like abstract thinking and the translation of real-world problems into the software context. Sample statement from our students are: *"I have not internalized that way of thinking yet"* or *"I still have trouble translating text into code"*.

Thus, focussing on modelling and implementation has at least established the awareness that typical ways of thinking exist in computer science and that the translation of concepts into source code is a process separate from finding the right concept. Although they were able to communicate the necessary processes they need to perform as professionals, they also stated that they are no professionals yet.

At the end of each semester there is a university-wide lecture evaluation, which is a solid data basis we can use to evaluate our teaching. The questionnaire consists mainly of items whose answer format are Likert scales with values ranging from 1 (fully disagree) to 5 (fully agree). We have data from a previous course taught in winter semester 2014/15. These can be compared with data collected with the help of the same questionnaire in winter semester 2017/18, where we applied the teaching concept.

The evaluation results we obtained for this make us feel optimistic. A statistically significant improvement can be seen in the questions listed in Table 10.3. We think that these are related to those goals which we expected to influence, such as making experts' thinking processes tangible, choosing better examples and promoting students' competence of abstract thinking.

Question	**Average 2014**	**Average 2017**	**p-Value**
The content structure of the course is comprehensible.	3.1	4.2	< 0.001
The learning outcomes of the course are clear.	3.4	4.1	< 0.001
The two parts of the course complement each other well.	3.2	4.0	< 0.001
The applicability of the course content is clear.	3.5	4.2	0.002
The practical exercises are well adapted to the subject matter.	3.2	3.9	0.002
The content of the course is complemented and illustrated by comprehensible examples.	3.6	4.2	0.004
The level of knowledge of the students is taken into account by the lecturer.	3.3	3.8	0.048
Complicated issues were explained in a comprehensible manner.	3.5	3.9	0.05
My previous knowledge is sufficient to follow the course.	3.0	3.6	0.051

Table 10.3.: Results of an unpaired t-test, showing the significance level for improvement between lectures in the years 2014/15 ($n = 40$) and 2017/18 ($n = 29$).

Even though the ATA has been conducted in all study groups at the beginning and the end of the first semester, I decided not to use the data to evaluate the effectiveness of the lecture. One reason is that splitting the students data ($n = 40$) according to their three study groups leads to very small sample sizes (between 5 and 20 students). Moreover, I do not have biographical and performance data of the two study groups. Thus, I am not able to investigate the representativeness of these groups. Furthermore, I cannot make a statement about the comparability of the three groups for example regarding the distribution of grades. Consequently, the validity of all findings derived from the data could not be guaranteed.

From our lecturers' perspective, the pair teaching approach is great. During the lecture, we experienced pair thinking and relaying for example when dealing with students' questions. Often students cannot formulate their questions precisely and have underlying misconceptions. We were much more effective in identifying the bottom line and correcting them immediately.

When reviewing the semester, the question was raised, whether one can take the different roles on his or her own when disciplined enough. In our opinion it could be possible, but it is necessary to be aware of all the different roles and to take them one by one. Furthermore, we think it is necessary to concern and embed the roles already in the design. Nevertheless, some constellations of roles just occurred during teaching and we did not plan them ahead. Thus, the approach added value to our teaching. Furthermore, effects like pair pressure, relaying and thinking can just occur in pairs. Hence, we recommend to try this approach and experience the great improvement if possible.

Part V.

Conclusion and Future Work

Conclusion and Future Work

The competence of abstract thinking is one of the key competences in computer science. However, there was a notable research gap. The current research was missing a consistent and complete definition of the competence, as well as an assessment tool. Consequently, hypotheses and teaching concepts could not be validated. This thesis narrows this gap. The approach described consists of four major steps: defining a competence, deriving an assessment, conducting studies and adapting the teaching based on the findings. This can not be applied only to the competence of abstract thinking. Thus, this dissertation serves as a sample for the investigation of further key competences in computer science, such as logical and analytical thinking.

A typical sequence of steps to investigate a competence is as follows: If there does not exist a competence model, one can be derived from a thorough literature review and should be completed by discussions with experts. During this step, cognitive processes that are involved in solving representative problems are collected. Afterwards, similar processes are summarised into competence components, which together form the competence model. Subsequently, the competence model can be used to develop an assessment. It is important to consider the intended setting, such as the time frame and the target group, to tailor the assessment accordingly. In addition, the research questions should be taken into account as these also influence the design of the assessment. The assessment can then be used to conduct studies in order to answer the research questions. Moreover, the data collected allows evaluating the quality of a new assessment, what I really recommend. All these steps provide important information that influences the development of future teaching practice.

This approach has particular significance since the analyses in this dissertation have shown that the competence of abstract thinking is necessary, but not sufficient to acquire computer science specific skills. Thus, more key competences in computer science should be defined, assessed and analysed to complete the picture and to provide insights that are relevant for teaching.

Conclusion

How can the competence of abstract thinking be defined in the context of computer science based on current literature? is the first research question stated in this dissertation. In order to answer the question, a literature review studying dictionaries, encyclopaedias and scientific publications has been conducted. The selection of scientific publications includes the domains psychology, educational research, mathematics, engineering and computer science. All statements collected were analysed and clustered. In the end, I came up with three clusters of cognitive processes, which build the three subcomponents of my final competence model namely *Communalities and Differences*, *Hide and Keep*; and *Expand*.

According to my knowledge, this is the first attempt to build a competence model for the competence of abstract thinking. Consequently, this thesis is a contribution to narrow the research gap regarding the competence. It establishes the basis for measuring and teaching the competence.

The development of an assessment for the competence of abstract thinking is the second goal of this dissertation which is derived from the research question *How can the competence of abstract thinking be measured in the context of computer science?* Based on the competence model it has been possible to develop an assessment called *Abstract Thinking Assessment (ATA)* to measure the abstract thinking competence of first-year students in computer science.

In order to judge students' answers objectively, a comprehensive and generic rating rationale has been developed. This serves as a blueprint to interpret answers to open-ended questions focussing on abstract thinking competence. For the ATA a detailed coding manual has been described to evaluate all items using a shared standard. Thus, the ATA creates the opportunity to assess students' level of competence in a standardised way. In addition, it is now possible to evaluate teaching concepts and methods in terms of effectiveness.

However, before the ATA allowed to make statements about students' competence, the quality had to be evaluated. The quality analysis shows that the ATA is objective, reliable and valid. Moreover, it allows differentiating students on all levels of proficiency. Hence, the ATA is another contribution of this dissertation to close the research gap regarding the competence of abstract thinking.

The ATA has successfully been used to collect data of 134 students, in order to answer two more research questions of this dissertation. The first is: *Is there a deficit in first-year students' initial competence of abstract thinking?* Analyses of the item difficulty have shown that items requiring to recognise abstractions seem to be rather easy for the students, whereas they struggle when building their own abstractions. Further analyses have shown that 73% of the students fulfil the lecturers' expectations on scale *Abstract Thinking Competence* by showing a sufficiently high level of abstraction. However, in my opinion, a high level of abstraction belongs to the competence of abstract thinking just like the accuracy and completeness of an abstraction. When combining both aspects the picture of students' competence is different. It is almost shocking that only 30% of the students achieve the expected number of answers that are correct and on a sufficiently high level of abstraction at the same time. In my opinion, this result shows that first-year students have a serious deficit in the competence of abstract thinking at the beginning of their studies.

Additionally, I investigated the impact of the initial abstract thinking competence on the acquisition of programming skills. The analysis shows a significant relation. The competence of abstract thinking is necessary for being successful in the end-of-term exam of the module *Software Development I*, but not sufficient. Thus, the research question *Is there a relation between the competence of abstract thinking and the acquisition of professional software development skills?* can be answered with a yes.

Since the deficits are severe, but the competence is needed to acquire professional software development skills, I see only two approaches to deal with this situation: Potential students can be tested and possibly rejected before starting their studies or the development of the competence is integrated into teaching at universities. As the first option is no longer allowed in Germany and as our research team believes that some students have the potential to succeed if they are supported enough.

This leads to the last research question of this dissertation: *How can students' competence of abstract thinking be promoted in a class on software development?* The result is a teaching concept, which consists of two parts: first is a description of specific teaching units and their development, second is the description of the teaching style pair teaching.

The teaching units follow recommendations for teaching the competence of abstract thinking and consider findings from the analyses conducted as part of this thesis. A major approach to design the teaching units were think-aloud sessions. These helped to make the implicit thought processes of experts visible. Consequently, the processes can be made transparent for the students. I recommend treating none of the processes as trivial or obvious, as none of them is for novices. Students rarely complain about taking too small steps, but they do if steps are too big.

A reflection activity six weeks after the beginning of the semester has shown that the awareness of implicit thinking processes raised among the students. They can now articulate whether they fail during the modelling or the implementation, which makes it easier for lecturers to assist students during teaching units. Additionally, this enables students to identify, categorize and reflect their sources of trouble on their own properly. Compared to previous years, the course evaluation shows a significant improvement regarding the comprehensibility of the lecture and the alignment of lecture and lab.

From our lecturers' perspective, pair-teaching is a great approach. During the lecture, we experienced pair thinking and relaying for example when dealing with students' questions. We were much more effective in correcting students misconceptions immediately. My colleague and I strongly recommend to try this approach and experience great improvement.

In conclusion, this dissertation narrows the research gap regarding the competence of abstract thinking and gives an example of how to define, measure and teach competences in general.

Future Work

Although this dissertation made good progress regarding the competence of abstract thinking, and narrowed the research gap, it opened up opportunities for further research.

In future research, typical techniques and heuristic approaches could be investigated and designed for each competence component. One example is already depicted in this dissertation for the component *Commonalities and Differences*. Parametrisation is a technique to normalise differences. I think, especially for the component *Expand* techniques and heuristic approaches can be useful for novices in software development, as these processes seem to be the most difficult ones for them. Teaching heuristic approaches are already common practice in related subjects, like mathematics, which is a good source to start from.

The ATA has been designed for assessing first-year students' competence of abstract thinking. However, there exist additional scenarios the ATA can be used for, like communicating typical tasks to facilitate the decision of prospective students, or to support the decision of admitting students to a program or not. Furthermore, the assessment tool can be used to measure the effectiveness of teaching practices focussing on abstract thinking competence or to evaluate students' progress in a specific module. Nevertheless, the field of application is limited, because the items reflect the level of knowledge of first-year students. In case of for example monitoring students' progress through their studies, further items need to be developed reflecting an appropriate level of difficulty.

For most future scenarios it is necessary to diminish the evaluation time or to automate the assessment. As long as the evaluation requires human rating the scenarios for application are limited. Through further analyses and research, it might be possible to develop new item formats that are less time intensive in the evaluation. The use of modern techniques, such as text mining, would also facilitate additional use cases.

The analysis has revealed that students show any constellation when combining the scales *Correctness* and *Abstract Thinking Competence*. Several students develop concrete, but incorrect abstractions, others build concrete and correct abstractions, and still other abstractions are abstract, but incorrect. Only some students provide abstract and correct abstractions. The last profile is the goal of teaching abstract thinking competence, but to my knowledge, no research exists on activities necessary to evolve a students profile into another. This dissertation laid the foundation for regularly monitoring students' development, but methods must be designed in a next step to promote students' competence.

It follows from all these findings that students require the competence of abstract thinking to become IT-professionals, but they do no have an appropriate level of the competence yet. This consequently requires a new teaching concept to additionally address abstract thinking competence right from the beginning. However, the data revealed that there is no clear division of students into groups with a comparable level of proficiency. From this I conclude that the cohort is highly heterogeneous. Thus, further research needs to be conducted to work out typical student profiles that can be addressed in teaching. This might also require to think of additional distinctive features, such as other key competences.

When further competences are evaluated at the same time, it would be advantageous to shorten the time frame of the ATA. In this case, I would recommend to only keep the 24 items where students need to build abstractions, as correctness and the level of abstract can be evaluated. For the combination of the two perspectives students' data have shown a serious deficit and recognising abstractions seems to be easy for the students. However, this would require to re-examine the quality of the resulting assessment.

For the design of the lecture, I conducted think-aloud session to make experts' thinking processes transparent. The insights gained into the unconscious processes were used to revise the lecture and accompanying material. These sessions additionally revealed that different experts follow different processes. Through modern technology, it is possible to take many more processes into account during lab-sessions or self-studying phases. However, more cognitive processes need to be investigated in order to get a complete picture. The results would provide an important contribution to dealing with the high heterogeneity lecturers are facing in first-semester classes.

In summary, this dissertation creates the basis for the ongoing debate surrounding abstract thinking competence in the context of computer science.

Appendix

Part VI.

Final Questionnaire

Fragebogen zum abstrakten Denken

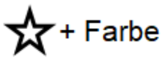
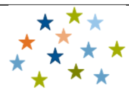
M

Ihre Daten	
Name	
Vorname	
Matrikelnummer	
Studiengruppe	

Bewertung	
Erreichte Punkte	
Ordnen, Analysieren, Begründen	
Zusammenfassen	
Londoner U-Bahn-Netz	
Lückenhafte Codes	
Fehlerhafte Codes	
Maschinen	
Im Raster	
Passwort-Sicherheitsstatus	
Zaubertränke	
Automatisierung	
Smartphone	

Gegeben sind im Folgenden je drei Abbildungen (Bild oder Text). Diese lassen sich jeweils in eine bestimmte Reihenfolge von **1: konkret** bis **3: abstrakt** bringen.

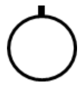
Beispiel

Abbildung			
Reihenfolge	2	1	3

Ordnen

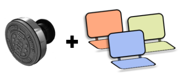
Rank, analyse and reason

Bringen Sie nun alle weiteren Dreiergruppen in eine Reihenfolge, die dem obigen Muster entspricht. Nummerieren Sie dazu die Abbildungen von **1: konkret** bis **3: abstrakt**.

Abbildung			
Reihenfolge			

Rank Apple

Abbildung 1: `https://pixabay.com/images/id-307761/`

Abbildung			
Reihenfolge			

Rank Seal

Stempel: `https://pixabay.com/images/id-2054429/`, Abdruck: `https://pixabay.com/images/id-2054429/`

Abbildung	1x drücken → 1 Eis 2x drücken → 2 Eis 3x drücken → 3 Eis	n Aktionen → n Reaktionen	n mal drücken → n Eis
Reihenfolge			

Rank Vending Machine

Abbildung	x mal ein Zeichen schreiben, Zeilenumbruch einfügen. Anschließend x nach vorgegebener Regel erhöhen und wieder von vorne mit dem Schreiben beginnen.	a aaaa aaaaaaaaa ...	n^2 mal a schreiben Zeilenumbruch n um eins erhöhen von vorne beginnen
Reihenfolge			

Rank Typing

Analysieren

Beschreiben Sie, wie Sie auf die Lösung der obigen Aufgaben gekommen sind. Welche Überlegungen haben Sie angestellt, bevor Sie die Lösung wussten. Stellen Sie Ihre Überlegungen kurz dar.

Gehen Sie hier nicht auf die Regel zur Anordnung der Abbildungen ein.

Reflect upon Ranking

Beschreiben Sie nun kurz, was die Reihenfolgen `aller vier` vorherigen Beispiele ausmacht.

Nicht ausreichend sind Aussagen, wie z.B: Die Abbildungen sind immer von konkret nach abstrakt geordnet.

Generalise Rankings

Ordnen und begründen

Bringen Sie die unten stehenden Abbildungen (Bild oder Text) in eine aufsteigende Reihenfolge.

Nummerieren Sie dazu die Abbildungen entsprechend von **1: konkret** bis **3 bzw. 4: abstrakt**.

Schreiben Sie zu jeder Teilaufgabe einen kurzen Satz, warum Sie sich für diese Reihenfolge entschieden haben.

Abbildung	Mona steckt zwei 5 €-Scheine in den Automaten und erhält: 1x2 €, 1x1 €, 2x20 ct sowie eine Fahrkarte.	Geld in den Fahrkartenautomat einwerfen und Fahrkarte sowie Wechselgeld entnehmen.		Peter wirft 10 € in den Automaten. Er erhält seine Fahrkarte und 3,40 € Wechselgeld.
Reihenfolge				
Begründung				

Rank Ticket

Justify Ticket

Abbildung	**Fisch** schwimmen	**Tiger** brüllen	**Lebewesen** fressen
Reihenfolge			
Begründung			

Rank Species

Justify Species

Summarise

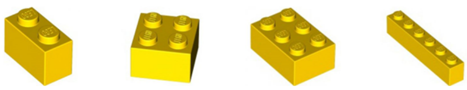
Zusammenfassen

1. Finden Sie für alle Objekte einer Teilaufgabe einen gemeinsamen und bezeichnenden Namen (Oberbegriff).
2. Nennen Sie **4 Merkmale**, die alle Objekte gemeinsam haben.

Sources: https://pixabay.com/images/id-306867/, https://pixabay.com/images/id-309527/, https://pixabay.com/images/id-309527/, https://pixabay.com/images/id-307709/, https://pixabay.com/images/id-2492912/, https://pixabay.com/images/id-3591510/

Name Vehicle	**Name aller Objekte**	
	Merkmal	
Characterise Vehicle (1)		
Characterise Vehicle (2)		
Characterise Vehicle (3)		
Characterise Vehicle (4)		

1. Finden Sie für alle Objekte einer Teilaufgabe einen gemeinsamen und bezeichnenden Namen (Oberbegriff).
2. Nennen Sie **4 Merkmale**, die alle Objekte gemeinsam haben.
3. Kreuzen Sie an, ob die Werte/Ausprägungen der Merkmale für alle Objekte gleich oder unterschiedlich sind.



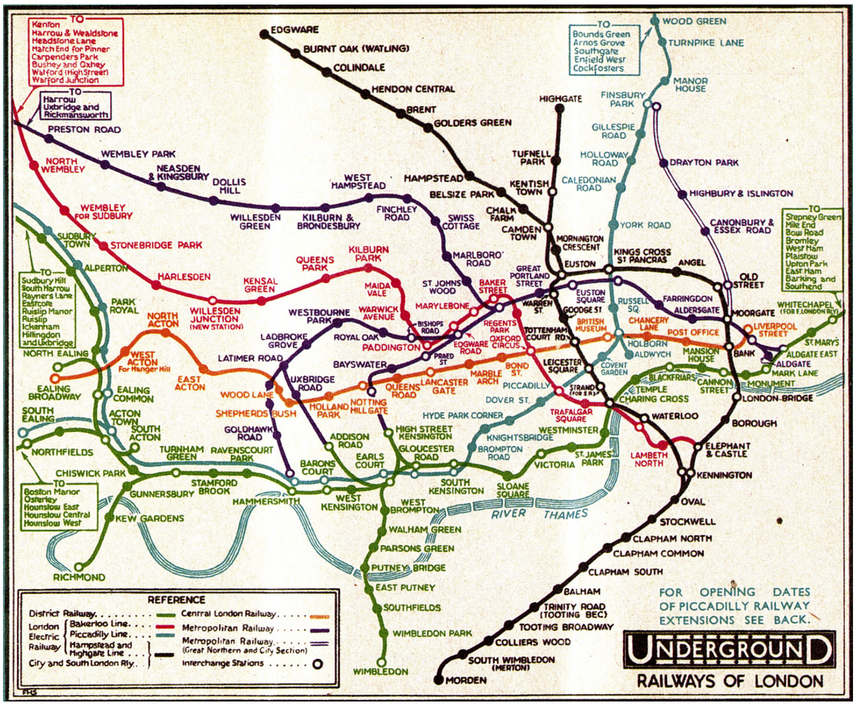
Source: https://www.die-welt-der-kleinen.de/lego-einzelelemente-einzelteile-ersatzteile-de/LEGO-Grundsteine-de/

Name Brick	**Name aller Objekte**		
		für alle Objekte	
	Merkmal	**gleich**	**unterschiedlich**
Characterise and Evaluate Brick (1)		☐	☐
Characterise and Evaluate Brick (2)		☐	☐
Characterise and Evaluate Brick (3)		☐	☐
Characterise and Evaluate Brick (4)		☐	☐

Londoner U-Bahn-Netz

Die Anzahl der Haltestellen auf beiden Plänen ist als identisch zu betrachten!

Londoner U-Bahn-Plan aus dem Jahr 1928.



http://amodern.net/wp-content/uploads/2013/08/3_tubemap.png

Londoner U-Bahn-Plan aus dem Jahr 1931.

http://www.geocities.ws/lhsoicher/images/1933a.jpg

Zwischen den Jahren 1928 und 1931 wurde der Londoner U-Bahn-Fahrplan aktualisiert. In der unten stehenden Tabelle finden Sie eine Liste von Merkmalen die der Fahrplan von 1928 aufweist.

Kreuzen Sie an, welche Merkmale bei der Überarbeitung unverändert blieben, und welche verändert wurden.

	Merkmale	unverändert	verändert
Stations	Haltestellen	☐	☐
Intersections	Kreuzungspunkte der Bahnen (Umsteigemöglichkeit)	☐	☐
Location of stations	Lage der Haltestellen	☐	☐
Distances	Distanzen zwischen den Haltestellen	☐	☐
Representation	Darstellung der Umgebung	☐	☐
Course	Verlauf der Bahnstrecken	☐	☐

Betrachten Sie nun Ihr Ergebnis.
Mit welcher Begründung können alle Veränderungen erklärt werden?

Justify Tube Map

Lückenhafte Codes

Fragmentary Codes

Im Folgenden wurden vier Nachrichten codiert. Unten abgebildet sehen Sie die originale Nachricht und den Code. Doch die Codierungen weisen Lücken auf (leere Zellen). Vervollständigen Sie die Lücken im Code.

Nachricht					Code			
+	+	-	-	→	o	o	p	p
-	-	+	+	→	p			
+	-	-	+	→	o			o

Easy Code

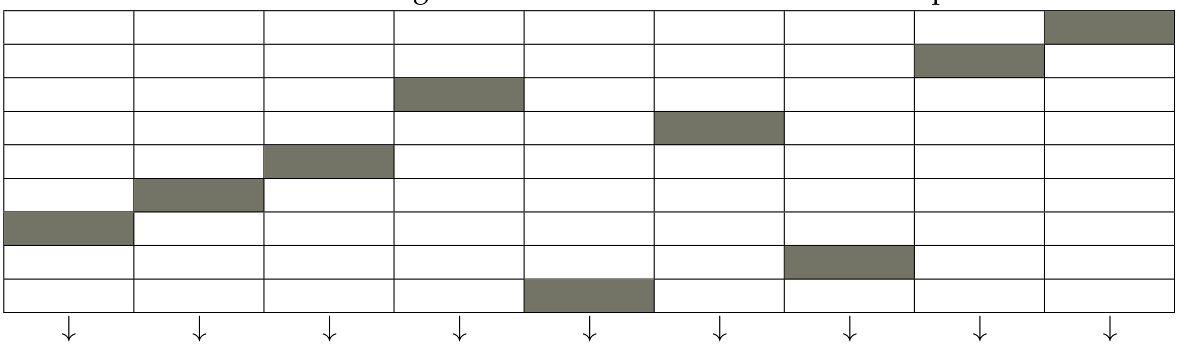
Nachricht						Code
#	#	+	+	+	→	z # d +
+	+	$	$	$	→	z + d $
$	+	+	#	#	→	
#	-	-	-	+	→	e # d - e +

Medium Code

Nachricht								Code
X	X	O	O	O	X	X	→	bxcobx
X	O	O	O	O	O	X	→	axeoax
O	O	I	I	I	I	O	→	
X	O	X	I	X	O	X	→	axaoaxaiaxaoax
X	X	O	O	O	X	X	→	bxcobx

Biber Code

Nun hat sich der Aufbau etwas geändert. Ein Zeichen wird durch eine Spalte codiert.



Vervollständigen Sie den Code! *(Es kommen nur die Buchstaben O, S und K vor.)*

	S	O		S	K	O	O	

Music Code

Faulty Codes

Fehlerhafte Codes

Aus verschiedenen Nachrichten wurden verschlüsselte Codes erstellt (Codierung). Jedoch hat sich dabei **jeweils ein Fehler** eingeschlichen. **Markieren** und **korrigieren** Sie diesen Fehler im Code.

Die grau hinterlegte Nachricht ist als korrekt zu betrachten.

Mark and Correct Binary to Alpha

Nachricht					Code			
00	00	01	10	→	A	A	B	C
01	10	110	110	→	B	C	D	D
01	111	110	10	→	B	E	C	C

Markieren und **korrigieren** Sie den **einen** Fehler im Code. Beschreiben Sie für die folgenden Beispiele **zusätzlich** kurz die **Regel**, die der Codierung zugrunde liegt.

Mark and Correct Counting Code

Nachricht		Code
1111111000001000000011111	→	7 5 1 6 4

Regel

Rule for Counting Code

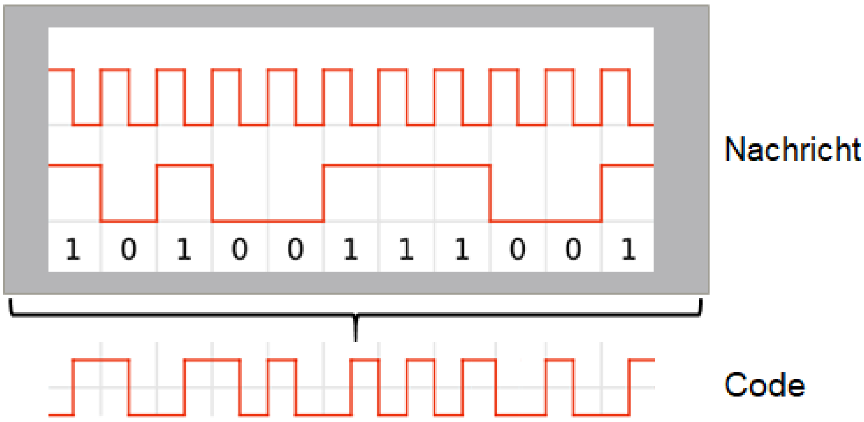
Mark and Correct Caesar Code

Nachricht		Code
MATHE	→	QEXLI
INFORMATIK	→	MRJSVQEWMO
STUDIUM	→	WXYHMYQ

Regel

Rule for Caesar Code

Profistufe!



Mark and Correct Digital Code

Regel

Rule for Digital Code

Machines

Maschinen

Eine Maschine erhält eine Zeile des Inputs und generiert daraus den Output in derselben Zeile. Beschreiben Sie kurz die generelle Funktion der untenstehenden Maschinen.

Beispiel

Input							Output
z	w	w	w	y	y	→	z1w3y2
a	a	y	n	x	x	→	a2y1n1x2
b	d	b	k	m	m	→	b2d1k1m2

Funktion: *Die Maschine gibt einen Buchstaben aus, gefolgt von der Anzahl der Vorkommen in der gesamten Zeile. Ist ein Buchstabe bereits berücksichtigt worden, so wird er nicht mehr ausgegeben.*

Input							Output
z	b	g	e	x	u	→	beguxz
a	k	d	s	f	h	→	adfhks

Funktion

Function Sorting

Input							Output
g	k	g	k	k	m	→	gkm
m	o	d	o	d	d	→	mod
m	m	o	d	o	e	→	mode

Funktion

Function Set

Im Raster

On the Grid

Finden und formulieren Sie eine Regel die beschreibt, wann im folgenden Zellen-Raster ein 'x' gesetzt wird. Jede Zelle wird durch eine Zeilen- und eine Spaltennummer eindeutig gekennzeichnet. Die gefundene Regel sollte so allgemein sein wie möglich.

Beispiel

	Spalte						Regel
		1	2	3	4	5	*Ein 'x' wird gesetzt, wenn die Zeilen-Nummer gleich der Spalten-Nummer ist.*
Z	1	x					
e	2		x				
i	3			x			
l	4				x		
e	5					x	

Tipp: Denken Sie an mathematische Operatoren wie z.B. $+, -$ und $<$. Oder auch an Gleichheit, Ungleichheit oder beispielsweise Teilbarkeit.

Grid Moved Diagonal

	Spalte						Regel
		1	2	3	4	5	*Ein 'x' wird gesetzt,*
Z	1		x				
e	2			x			
i	3				x		
l	4					x	
e	5						

Grid Triangle

	Spalte						Regel
		1	2	3	4	5	
Z	1	x					
e	2	x	x				
i	3	x	x	x			
l	4	x	x	x	x		
e	5	x	x	x	x	x	

Grid Chess

		Spalte					Regel
		1	2	3	4	5	
Z	1	x		x		x	
e	2		x		x		
i	3	x		x		x	
l	4		x		x		
e	5	x		x		x	

Profistufe!

Grid Tree

		Spalte									Regel
		1	2	3	4	5	6	7	8	9	
Z	1					x					
e	2				x	x	x				
i	3			x	x	x	x	x			
l	4		x	x	x	x	x	x	x		
e	5	x	x	x	x	x	x	x	x	x	

Passwort-Sicherheitsstatus

Pass phrase Security Status

Auf vielen Internet-Seiten wird Ihnen angezeigt, ob ein Passwort schwach, mittel oder stark ist. Sie wollen herausfinden, welche Kriterien dem Vorgang einer bestimmten Homepage zugrunde liegen. Dazu geben Sie verschiedene Passwörter ein und notieren sich den angezeigten Status. Das Ergebnis dieser Recherche finden Sie in der folgenden Tabelle.

Passwort	Status
Servus	schwach
Baum36	schwach
T?o8	schwach
$5&-!?	schwach
SteffisPassworT	mittel
!nformatik	mittel
urlaubindien2013	mittel
studium	mittel
20!7-!0	mittel
B3!$p!3l	stark
§Passwort&	stark
TH3OPW!	stark
§4%68?9!a	stark
6PassWort	stark

Aus welchen Bestandteilen setzen sich die verschiedenen Passwörter zusammen?

Pass phrase Components

In welchen Bestandteilen und anderen Merkmalen unterscheiden sich die Passwörter von `schwach` und `mittel`?

Differences weak – medium

In welchen Bestandteilen und anderen Merkmalen unterscheiden sich die Passwörter von `stark` und `mittel`?

Differences medium – strong

Regeln

Wie lautet die vollständige Regel zur Bestimmung des Sicherheitsstatus?

Status	Regel
Rule weak **schwach**	*Ein Passwort wird als schwach eingestuft, wenn ....*
Rule medium **mittel**	*Ein Passwort wird als mittel eingestuft, wenn ....*
Rule strong **stark**	*Ein Passwort wird als stark eingestuft, wenn ....*

Weitere Passwörter

Konstruieren Sie aus jeder Ihrer Regeln ein weiteres Beispiel-Passwort.

Status	Passwort
schwach	
mittel	
stark	

Zaubertränke

Miraculix hat vergessen, wie der Zaubertrank, der die Gallier so stark macht, gebraut wird. Er versucht, sich zu erinnern und braut verschiedene Zaubertränke. Einige zeigen erstaunliche Wirkungen.

Miraculix notiert sich, welcher Zaubertrank welche **Wirkung** hervorruft. Für jeden Zaubertrank benutzt er eine Kombination aus den Buchstaben **X, O und I**. Mit Hilfe dieser Schreibweise kann auch später nachvollzogen werden, welche Wirkung der Trank hat.

https://www.comedix.de/lexikon/db/handzumgrus.php

Zaubertränke	Wirkung		
XIXXXOX	–	–	–
XXIIXXX	–	–	–
OXIXXXX	–	–	–
IIOXOII	–	–	lässt leuchten
IOIIIOI	–	–	lässt leuchten
OXXXXXO	–	–	lässt leuchten
OOOIOOO	–	macht schwerelos	lässt leuchten
OXOIXXO	–	macht schwerelos	–
OOOIXXO	macht kariert	macht schwerelos	–
OXXIXXO	macht kariert	macht schwerelos	lässt leuchten
XXXOXXX	macht kariert	–	lässt leuchten
XXXIXXX	macht kariert	–	lässt leuchten
OXXOXXO	macht kariert	–	lässt leuchten
XXOOOII	macht kariert	–	–
XXXIXOO	macht kariert	–	–
OXXXXXX	macht kariert	–	–

Was haben alle Zaubertränke, die *kariert* machen, gemeinsam?

Potion chequered commonalities

Ist dieses Kriterium ausreichend, um die Zaubertränke die *kariert* machen, von allen Zaubertränken, die nicht kariert machen, zu unterscheiden?

☐ Ja ☐ Nein

Was haben alle Zaubertränke, die *schwerelos* machen, gemeinsam?

Potion weightless commonalities

Ist dieses Kriterium ausreichend, um die Zaubertränke, die *schwerelos* machen, von allen Zaubertränken, die nicht schwerelos machen, zu unterscheiden?

☐ Ja ☐ Nein

Was unterscheidet Zaubertränke, die *leuchten* lassen von solchen, die das nicht bewirken?

Potion luminous differences

Ist dieses Kriterium ausreichend, um alle Zaubertränke, die *leuchten* lassen, zu identifizieren?

☐ Ja ☐ Nein

Automatisierung

Automation

Der folgende Ablauf ist ein Ausschnitt aus einem Computerprogramm. Dessen Ziel ist es, das Wort `INFORMATIK` systematisch auf eine bestimmte Eigenschaft hin zu analysieren.

(1) `Vergleiche I mit K`
(2) `Vergleiche N mit I`
(3) `Vergleiche F mit T`
(4) `Vergleiche O mit A`
(5) `Vergleiche R mit M`
(6) `Vergleiche M mit R`
(7) `Vergleiche A mit O`
(8) `Vergleiche T mit F`
(9) `Vergleiche I mit N`
(10) `Vergleiche K mit I`

Formulieren Sie den obigen Ablauf so, dass er für beliebige Wörter der Länge 10 anwendbar ist.

Palindrome constant length

Lässt sich der obige Analyse-Prozess verkürzen? Begründen Sie Ihre Antwort.

☐ Ja ☐ Nein

Wie müsste der Ablauf aussehen, um für beliebige Wörter mit gerader Anzahl an Buchstaben anwendbar zu sein? Formulieren Sie diesen Ablauf.

Palindrome various length

Smartphone

Smartphone

Beschreiben Sie Ihr Smartphone auf drei verschiedenen Ebenen der Abstraktion. Nutzen Sie dazu lediglich Text. Nennen Sie zu jeder Ebene den Abstraktionsgrad: niedrig, mittel, hoch.

Smartphone – 3 absraction levels

Beschreibung	
Abstraktionsgrad	

Smartphone – 3 absraction levels (continued)

Beschreibung	
Abstraktionsgrad	

Smartphone – 3 absraction levels (continued)

Beschreibung	
Abstraktionsgrad	

Begründen Sie nun, warum Ihre gewählten Beschreibungen drei unterschiedliche Ebenen der Abstraktion darstellen.

Justify Smartphone Levels

Vielen Dank für Ihre Teilnahme!

Part VII.

Coding Manual

The sloppiness of students' answers is transferred to the sample answers.

Ordnen, Analysieren und Begründen (Rank, analyse and reason)

ab Seite 2

Ordnen

Korrektheit	Code
Apfel	$1-1\_rank\_correct$
1, 2, 3	1
andere	0
Stempel	$1-2\_rank\_correct$
2, 3, 1	1
andere	0
Automat	$1-3\_rank\_correct$
1, 3, 2	1
andere	0
Zeichen	$1-4\_rank\_correct$
3, 1, 2	1
andere	0
Permutation	**Code**
Apfel	$1-1\_rank\_perm$
Stempel	$1-2\_rank\_perm$
Automat	$1-3\_rank\_perm$
Zeichen	$1-4\_rank\_perm$
Leer	999
Mehrfachnennung eines Rangs	0
1, 2, 3	1
1, 3, 2	2
2, 1, 3	3
2, 3, 1	4
3, 2, 1	5
3, 1, 2	6

Analysieren (Analyse)

Seite 3

Aufgabe: Überlegungen zum Lösungsprozess		Code
Korrektheit $1-5\_reflect\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Beschreibungen, die sich auf die Anordnung der Abbildungen beziehen und nicht auf den Lösungsprozess bevor eine Anordnung überhaupt klar ist. – *Erst geschaut, welches der 3 Bilder das Ergebnis ist. Dann geschaut wie man es erstellt und daraufhin die allgemeine Beschreibung gewählt.*	0
Richtig	Beschreibungen, die Überlegungen zu einer Herleitung des Sortierkriteriums enthalten. Oder Antworten, die das Finden der zugrundeliegenden Struktur beschreiben.	1
Abstraktionsniveau $1-5\_reflect\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Beschreibungen, die sich auf konkrete Abbildungen beziehen oder jede Reihenfolge einzeln beschreiben.	0
Spezifisch	Beschreibungen, die sich auf drei Abbildungen beziehen. – *Erst geschaut, welches der 3 Bilder das Ergebnis ist. Dann geschaut wie man es erstellt und daraufhin die allgemeine Beschreibung gewählt.* – *Abbildung mit der größten Feature-Dichte identifiziert und als erstes Element gesetzt. Abbildung mit der geringsten Feature-Dichte identifiziert und als drittes Element gesetzt, ...*	1
Generisch	Beschreibung ist unabhängig von der Anzahl der Bilder und deren spezifischen Eigenschaften, wie z.B. Objekt oder Prozess. – *Erst habe ich mir angeschaut, welche Merkmale die einzelnen Abbildungen aufweisen. Danach habe ich die Abbildungen nach der Anzahl der spezifischen Merkmale absteigend sortiert oder nach Generalität der Aussagen aufsteigend.*	2

Aufgabe: Verallgemeinerung der Reihenfolgen		**Code**
Korrektheit $1-6\_analyse\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv, identische Struktur, alles mögliche Beispiele der Anwendung des abstrakten Gegenstandes*	99
Falsch	Die Begründung ... ... ist nicht für alle Abbildungen anwendbar. ... wechselt das Sortierkriterium. ... ist nicht vollständig. – *Von konkreten Features, wie Farbe, Anzahl, nach abstrakter Darstellung eines Algorithmus sortiert.*	0
Richtig	Konsistente und vollständige Begründung, sowie einheitlich genutztes Kriterium zur Sortierung (kein Wechsel der Regel zwischen den Elementen). Kann auch jeden Rang separat begründen. – *Erst sah man ein anschauliches Bild oder einen anschaulichen Vorgang (1). Dieses Bild oder dieser Vorgang wurde dann verallgemeinert ausgedrückt (2) und dann zu einer Regel geformt (3).* – *1 = war eigentlich das Ergebnis* *2 = Beschreibung, wie das Ergebnis entstanden ist* *3 = Verallgemeinerung*	1
Abstraktionsniveau $1-6\_analyse\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv, identische Struktur, alles mögliche Beispiele der Anwendung des abstrakten Gegenstandes*	99
Konkret	Beschreibung der einzelnen Abbildungen ohne Zusammenhang. *1 = war eigentlich das Ergebnis* *2 = Beschreibung, wie das Ergebnis entstanden ist* *3 = Verallgemeinerung*	0
Spezifisch	Beschreibungen, die sich auf drei Abbildungen beziehen. *Erst sah man ein anschauliches Bild oder einen anschaulichen Vorgang (1). Dieses Bild oder dieser Vorgang wurde dann verallgemeinert ausgedrückt (2) und dann zu einer Regel geformt (3).*	1
Generisch	Beschreibung ist unabhängig von der Anzahl der Bilder und deren spezifischen Eigenschaften, wie z.B. Objekt oder Prozess. *Von konkreten Features, wie Farbe, Anzahl, nach abstrakter Darstellung eines Algorithmus sortiert.*	2

Ordnen und begründen (Rank and reason)

Seite 4

Fahrkartenautomat

Fahrkartenautomat - Reihenfolge	Code
Korrektheit	$1-7\_rank\_correct$
1, 3, 4, 2	1
andere	0
Permutation	$1-7\_rank\_perm$
Leer	999
Mehrfachnennung eines Rangs	0
1, 2, 3, 4	1
1, 2, 4, 3	2
1, 3, 2, 4	3
1, 3, 4, 2	4
1, 4, 2, 3	5
1, 4, 3, 2	6
2, 1, 3, 4	7
2, 1, 4, 3	8
2, 3, 1, 4	9
2, 3, 4, 1	10
2, 4, 1, 3	11
2, 4, 3, 1	12
3, 1, 2, 4	13
3, 1, 4, 2	14
3, 2, 1, 4	15
3, 2, 4, 1	16
3, 4, 1, 2	17
3, 4, 2, 1	18
4, 1, 2, 3	19
4, 1, 3, 2	20
4, 2, 1, 3	21
4, 2, 3, 1	22
4, 3, 1, 2	23
4, 3, 2, 1	24

Fahrkartenautomat - Begründung		**Code**
Korrektheit $1-7\_reason\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Die Begründung ... ... ist nicht für alle Abbildungen anwendbar. ... wechselt das Sortierkriterium. ... ist nicht vollständig. ... definiert 2 Abbildungen als äquivalent.	0
Richtig	Konsistente und vollständige Begründung, sowie einheitlich genutztes Kriterium zur Sortierung (kein Wechsel der Regel zwischen den Elementen). Kann auch jeden Rang separat begründen.	1
Abstraktionsniveau $1-7\_reason\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Beschreibung der einzelnen Abbildungen ohne Zusammenhang. – *1. Komplizierter Vorgang* *2. Gleicher Vorgang vereinfacht* *3. Vorgang verallgemeinert & reduziert* *4. Abstrakte Beschreibung auf ein Bild reduziert*	0
Spezifisch	Beschreibungen, die sich auf drei Abbildungen beziehen. – *Der größte Informationsgehalt steckt in (1), dann (2) → Bei (2) weiß man nicht genau, wie das Wechselgeld aufgeteilt ist. (3) ist nur eine Beschreibung der Tätigkeit und (4) eine grafische Darstellung ohne Details.* – *1. wird genau beschrieben, wer wieviel in den Automaten wirft => dies wird immer weiter abstrahiert bis Nr. 4, wo man nicht erkennt, was das zu bedeuten hat, da das Bild auch etwas anderes bedeuten könnte als ein Ticket zu kaufen.*	1
Generisch	Beschreibung ist unabhängig von der Anzahl der Bilder und deren spezifischen Eigenschaften. – *Der Vorgang des Ticketkaufs wird mit immer weniger Details beschrieben. Die Detailgenauigkeit bildet das Sortierkriterium. Je weniger Details, desto abstrakter.*	2

Lebewesen

Klassen von Lebewesen - Reihenfolge	Code
Korrektheit $1-8\_rank\_correct$	
2, 1, 3	1
andere	0
Permutation $1-8\_rank\_perm$	
Leer	999
Mehrfachnennung eines Rangs	0
1, 2, 3	1
1, 3, 2	2
2, 1, 3	3
2, 3, 1	4
3, 2, 1	5
3, 1, 2	6

Klassen von Lebewesen - Begründung		Code
Korrektheit $1-8\_reason\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Unklare, unpräzise oder fachliche falsche Antworten, wie z.B. Beschreibung der Begriffe. – *Tiger ist ein ganz bestimmtes Tier und Lebewesen ist die Obergruppe von Tiger und Fisch.* – *Tiger ist ein ganz bestimmtes Tier, brüllen eine mögliche Handlung eines Tigers.*	0
Richtig	Die Argumentation in konsistent und die Regel zur Begründung der Reihenfolge wird nicht gewechselt. – *Lebewesen beinhaltet beides, Tiger und Fische. Fisch ist eine Klasse (mehrere Arten möglich). Tiger ist eine Art der Klasse Säugetier.* – *Die Abbildungen sind anhand ihres biologischen Hierarchielevels von einer konkreten Art zur Oberklasse aufsteigend sortiert.*	1

Abstraktionsniveau $1 - 8\_reason\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv, Hierarchie*	99
Konkret	Beschreibung der einzelnen Abbildungen ohne Zusammenhang. – *Tiger ist ein ganz bestimmtes Tier, brüllen eine mögliche Handlung eines Tigers.* – *Tiger ist ein ganz bestimmtes Tier und Lebewesen ist die Obergruppe von Tiger und Fisch.* – *Lebewesen beinhaltet beides, Tiger und Fische. Fisch ist eine Klasse (mehrere Arten möglich). Tiger ist eine Art der Klasse Säugetier.* – *1. vor 2. da die Aktion des brüllenden Tigers seltener ist, als das Schwimmen eines Fisches (das tut der immer) 3. generelle Formulierung für die Gruppe deren Vatergruppen Fische und Tiger angehören.*	0
Spezifisch	Beschreibungen, die sich auf drei Abbildungen beziehen. – *Vom Tier über die Tierart hin zum Überbegriff aller Tiere.*	1
Generisch	Beschreibung ist unabhängig von der Anzahl der Bilder und deren spezifischen Eigenschaften. – *Die Abbildungen sind anhand ihres biologischen Hierarchielevels (von einer konkreten Art zur Oberklasse) aufsteigend sortiert.*	2

Zusammenfassen (Summarise)

Seite 5

Fahrzeuge

Name		Code
Korrektheit $2 - 1\_summarise\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Namen werden als falsch bewertet, wenn sie nicht alle Beispiele umfassen oder auf einer zu hohen Abstraktionsebene sind. *Ding*	0
Richtig	So genau wie möglich, so abstrakt wie nötig. *Fahrzeug(e), motorisiertes Vehikel, Verkehrsmittel*	1

Merkmale		Code
Für jedes Merkmal separat zu kodieren. Wird das gleiche Merkmal mehrmals genannt, so wird es einmal kodiert und alle weiteren Vorkommen mit *leer* kodiert.		
Korrektheit $2 - 1\_component\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Merkmale werden als falsch bewertet, wenn sie nicht alle Beispiele umfassen.	0
Richtig	Ein Merkmal wird als richtig kodiert, wenn es in jedem der Bilder vorkommt, oder für in der westlichen Hemisphäre sozialisierte Personen sinnvoll hinein interpretierbar ist. Zählende Merkmale wie Räder werden auch als korrekt bewertet, wenn die Anzahl bei einzelnen Gruppen 0 ist. *Räder, Gaspedal, Fenster, Farbe, PS, Marke/Hersteller, Fahrzeugtyp, Anzahl Ränder, Anzahl Sitze, Beförderung von Personen*	1

Merkmale		**Code**
Für jedes Merkmal separat zu kodieren. Wird das gleiche Merkmal mehrmals genannt, so wird es einmal kodiert und alle weiteren Vorkommen mit *leer* kodiert.		
Art des Merkmals $2-1\_component\_type$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Komponente	*Räder, Gaspedal, Fenster, Sitze, Motor, Dach*	1
Beschreibung	*Farbe, PS, Marke/Hersteller, Benutzungsort, Spaßfaktor, Gewicht*	2
Verb	*fahren, Kraftstoff wird verbraucht*	3
Zählend	*Anzahl Ränder, Anzahl Sitze, Räder pro Achse, Personenanzahl*	4
Kategorie	*Fahrzeugtyp*	5
Zweck	*Beförderung von Personen*	6

Legosteine

Name		Code
Korrektheit $2 - 2\_summarise\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Namen werden als falsch bewertet, wenn sie nicht alle Beispiele umfassen oder auf einer zu hohen Abstraktionsebene sind. *Ding*	0
Richtig	So genau wie möglich, so abstrakt wie nötig. *(gelbe) Lego-Steine, Spielsteine, Bauklötzchen*	1

Merkmale		Code
Für jedes Merkmal separat zu kodieren. Wird das gleiche Merkmal mehrmals genannt, so wird es einmal kodiert und alle weiteren Vorkommen mit *leer* kodiert.		
Korrektheit $2 - 2\_component\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Merkmale werden als falsch bewertet, wenn sie nicht alle Beispiele umfassen.	0
Richtig	Ein Merkmal wird als richtig kodiert, wenn es in jedem der Bilder vorkommt, oder für in der westlichen Hemisphäre sozialisierte Personen sinnvoll hinein interpretierbar ist. Zählenden Merkmalen werden auch als korrekt bewertet, wenn die Anzahl bei einzelnen Gruppen 0 ist. *Kanten, Ecken, "Noppen", rechteckig, gelb, Anzahl Verbindungsstellen*	1

Merkmale		Code
Für jedes Merkmal separat zu kodieren. Wird das gleiche Merkmal mehrmals genannt, so wird es einmal kodiert und alle weiteren Vorkommen mit *leer* kodiert.		
Art des Merkmals $2-2\_component\_type$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Komponente	*Verbindungsstellen, Kanten, Ecken, "Noppen"*	1
Beschreibung	*rechteckig, gelb, kompatibel, Material, Höhe, Breite, Zustand*	2
Verb		3
Zählend	*Anzahl "Noppen"*	4
Kategorie		5
Zweck		6

Bewertung der Gleichheit		Code
Für jedes Merkmal separat zu kodieren. Wird das gleiche Merkmal mehrmals genannt, so wird es einmal kodiert und alle weiteren Vorkommen mit *leer* kodiert.		
Korrektheit $2-2\_commOrDiff\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Bewertung abhängig vom genannten Merkmal entscheiden. Ist die Bewertung des Merkmals in Bezug auf die gegebenen Beispiele falsch, entscheidet der Kodierer.	0
Richtig	Bewertung abhängig vom genannten Merkmal entscheiden. Ist die Bewertung des Merkmals in Bezug auf die gegebenen Beispiele korrekt, entscheidet der Kodierer.	1

Londoner U-Bahn-Netz (Tube Map)

Seite 6 – 7

Evaluation der Überarbeitung

Merkmale	unverändert	verändert
Haltestellen	☒	☐
Kreuzungspunkte der Bahnen (Umsteigemöglichkeit)	☒	☐
Lage der Haltestellen	☐	☒
Distanzen zwischen den Haltestellen	☐	☒
Darstellung der Umgebung	☒	☐
Verlauf der Bahnstrecken	☐	☒

Korrektheit	Code
Haltestelle	$3-1-1\_change\_correct$
unverändert	1
verändert	0
Kreuzungspunkte	$3-1-2\_change\_correct$
unverändert	1
verändert	0
Lage	$3-1-3\_change\_correct$
verändert	1
unverändert	0
Distanz	$3-1-4\_change\_correct$
verändert	1
unverändert	0
Darstellung	$3-1-5\_change\_correct$
unverändert	1
verändert	0
Verlauf	$3-1-6\_change\_correct$
verändert	1
unverändert	0

Begründung der Veränderungen

Begründung - Zweck		Code
Korrektheit $3 - 2\_reason\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv* – *Übersichtlichkeit* – *Durch eine bessere Technik bzw. Kartografie hat sich der Kartenaufbau komplett verändert. Somit ist es fast selbstverständlich, dass die Karten von Grund auf verschieden sind. Stichwort: Fortschritt.* – *Das Netz hat sich nicht geändert nur die Darstellung.* – *Der alte Fahrplan wurde komplett neu dargestellt, hierbei sind die Veränderungen zurück zu führen auf eine andere Skalierung und man wollte versuchen den Stil von Karte zu Plan zu wechseln, was mit den geraden Strichen (keine wilden Kurven) gelungen ist.* – *Neue Haltestellen entstehen, aufgrund Zuwachs von Einwohnern in einer bestimmten Gegend und somit muss der Verlauf entsprechend umgebaut werden bzw. hinzugefügt werden.*	99
Falsch	Antworten sind falsch, wenn einer der folgenden Aspekte fehlt: * Zweck: Übersichtlichkeit erhöhen, o.ä. * Prozess: Weglassen oder Hervorheben von Informationen o.ä. – *Durch die angestrebte bessere Benutzerfreundlichkeit, beispielsweise durch die bessere Übersicht und die bessere Lesbarkeit.* – *Alle Änderungen haben zu mehr Übersichtlichkeit beigetragen.* – *Die abstrakte moderne Darstellung erlaubt den gewünschten Bahnhof oder die geplante Strecke schneller optisch zu erfassen.* – *Die Übersichtlichkeit wurde deutlich verbessert. Es ist leichter den Weg von A nach B zu finden.*	0
Richtig	Antworten sind richtig, wenn Ziel, Prozess und Zweck erkennbar sind. – *Es ist nur notwendig zu wissen, wo man abfährt und ankommt und die ungefähre Lage zu wissen, da der Plan so einerseits übersichtlicher gestaltet werden kann somit schneller verstanden und gelesen werden kann ohne relevante Informationen auszulassen.* – *Da man sich während dem U-Bahnfahren der genauen Umgebung und der Distanz zwischen den Haltestellen nicht bewusst sein muss, ist eine vereinfachte Darstellung vorzuziehen, die die wichtigen Informationen (Haltestellen und Verbindungen) schneller erkennen lässt.*	1

Lückenhafte Codes (Fragmentary Codes)

Seite 8

Easy Code

Code vervollständigen		Code
Korrektheit 4 — 1_*relation_correct*		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	*alle anderen*	0
Richtig	`poopp`	1
Descriptive 4 — 1_*relation_answer*		
Antwort erfassen		
Code fortlaufend und ohne Leerzeichen erfassen. Fehlende Stellen werden mit – gekennzeichnet.		

Medium Code

Code vervollständigen		Code
Korrektheit 4 — 2_*relation_correct*		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	*alle anderen*	0
Richtig	`e$z+z#`	1
Descriptive 4 — 2_*relation_answer*		
Antwort erfassen		
Code fortlaufend und ohne Leerzeichen erfassen.		

Biber Code

Code vervollständigen		Code
Korrektheit 4 − 3_*relation_correct*		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	*alle anderen*	0
Richtig	`bodiao`	1
Descriptive 4 − 3_*relation_Answer*		
Antwort erfassen *Code fortlaufend und ohne Leerzeichen erfassen.*		

Music Code

Code vervollständigen		Code
Korrektheit 4 − 4_*relation_correct*		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	*alle anderen*	0
Richtig	`KSK`	1
Descriptive 4 − 4_*relation_Answer*		
Antwort erfassen *Code fortlaufend und ohne Leerzeichen erfassen. Fehlende Stellen werden mit – gekennzeichnet.*		

Fehlerhafte Codes (Faulty Codes)

Seite 9

Binär nach Buchstaben - Alphabet Code

Markieren und korrigieren		Code
Korrektheit $5-1\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	*alles andere*	0
Richtig	C in Zeile 3 - Spalte 3 markieren und durch ein D korrigieren.	1

Counting Code

Markieren und korrigieren		Code
Korrektheit $5-2-1\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	*alle anderen*	0
Richtig	Die letzte Ziffer des Codes (4) markieren und durch eine 5 korrigieren.	1

Regel		Code
Korrektheit $5-2-2\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Regel, die kein Regelsystem beschreiben, sondern nur die korrekte Lösung nennen. *Am Ende stehen 5 Einser.* Regel, die nicht die Anzahl und den Wechsel von 1 auf 0 und umgekehrt beinhaltet. *24 ist nicht durch 5 teilbar => Es ist eine Zahl zu viel.*	0
Richtig	Die Regel wird explizit beschrieben, nicht nur angedeutet. *Auch korrekt, wenn die Regel bei Code beginnend mit 0 eine nicht eindeutige Lösung liefert.* – *Die Ziffern beschreiben abwechselnd das auftauchen einer Ziffer 1 oder 0.* – *In der Nachricht wechseln immer 1 und 0 - die Anzahl der 1 und 0 ergibt den Code.* – *Anzahl der 1 am Anfang erste Zahl, Anzahl der 0 danach 2. Zahl usw.*	1

Regel		Code
Abstraktionsniveau $5-2-2\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv, Anzahl 0er und 1er*	99
Konkret	Regelbeschreibung argumentiert mit konkreten Text/Codeworten, ohne explizite Formulierung einer Regel. Regel ähnelt einer Handlungsanweisung. – *4 müsste 5 sein, da 5x1* – *7x "1", 5x "0", ...*	0
Spezifisch	Regelbeschreibung stellt Regel auf, argumentiert aber mit konkreten Anzahlen, z.B. 5 für die letzte Folge von Einsen. Es kommen spezifische Formulierungen neben generischen Formulierungen vor. – *Es werden die 1 nacheinander gezählt, falls die Reihe durch eine 0 unterbrochen wird zählt man die 0-Reihe bis die wieder durch eine 1 unterbrochen wird. Dies wiederholt sich bis zum Ende.* – *Anzahl der 1 am Anfang erste Zahl, Anzahl der 0 danach 2. Zahl usw.*	1
Generisch	Regelbeschreibung bezieht sich allgemein auf die Anzahl der 0er und 1er und ist unabhängig z.B. vom Beginn der Nachricht. – *Die Ziffern beschreiben abwechselnd das Auftauchen einer Ziffer 1 oder 0.* – *In der Nachricht wechseln immer 1 und 0 - die Anzahl der 1 und 0 ergibt den Code.* – *Die Zahl im Code beschreibt, wieviele boolesche Werte ununterbrochen aufeinander folgen.*	2

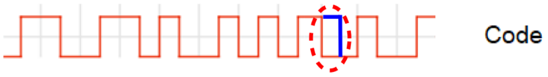
Caesar Code

Markieren und korregieren		Code
Korrektheit $5-3-1\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	*alle anderen*	0
Richtig	Das W im Code zu Informatik markieren und durch ein X korrigieren.	1

Regel		Code
Korrektheit $5-3-2\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv,* *Jeder Buchstabe der Nachricht bekommt einen Buchstaben den Allerbesten neu zugeordnet.*	99
Falsch	Regel, die kein Regelsystem beschreiben, sondern nur die korrekte Lösung nennen. – *Das T ist bei Studium und Mathe ein X $\rightarrow$ kann kein W sein da nur ein Fehler.* – *Zwischen T und W stehen nicht drei Buchstaben.* – *R $\equiv$ entweder U oder V*	0
Richtig	Regelbeschreibung nutzt Mehrheitsentscheid oder bezieht sich auf eine Ersetzung der Zeichen durch im Alphabet folgende Buchstaben. – *Bei der Übersetzung von links nach rechts muss jeder Buchstabe mit dem Buchstaben ersetzt werden, der im Alphabet 3 Buchstaben danach kommt.* – *M = Q, A = E, ...* – *T wird in MATHE und STUDIUM durch X ersetzt. Muss also auch in "INFORMATIK" auf X gemappt werden.*	1

Regel		Code
Abstraktionsniveau $5-3-2\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Regel, die kein Regelsystem beschreiben, sondern nur einzelne konkrete Vorschriften nennen. – *M = Q, A = E, ...* – *Zwischen T und W stehen nicht drei Buchstaben.* – *R* $\equiv$ *entweder U oder V* – *Das W müsste ein X sein, da bei Mathe & Studium statt T ein X steht.* – *Das T ist bei Studium und Mathe ein X* $\rightarrow$ *kann kein W sein da nur ein Fehler.*	0
Spezifisch	Antworten die zeigen, dass eine Abbildungsvorschrift existiert, aber explizit keine oder eine spezielle Ersetzungstabelle beschreiben. – *Ein Buchstabe repräsentiert immer einen anderen Buchstaben. T wird als X dargestellt, nicht als W.* – *Der Code ist so aufgebaut, dass der Buchstabe der Nachricht für einen entsprechenden Buchstaben im Code steht. So ist der Buchstabe "A" im Code ein "E". Dies wird dann einfach runtergezählt, sodass bei "B" im Code dann ein "F" steht.* – *Jedem Buchstaben wird ein anderer zugeordnet.* – *Ein Buchstabe entspricht einem anderen Buchstaben, z.B. T* $\triangleq$ *X*	1
Generisch	Antworten die eine Abbildungsvorschrift nennen und alle nötigen Kriterien beschreiben. – *Bei der Übersetzung von links nach rechts muss jeder Buchstabe mit dem Buchstaben ersetzt werden, der im Alphabet 3 Buchstaben danach kommt.* – *Buchstaben shift von 4*	2

Digital Code

Markieren und korregieren		**Code**
Korrektheit $5 - 4 - 1\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	*alle anderen*	0
Richtig		1

Regel		**Code**
Korrektheit $5 - 4 - 2\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv* *Bei eine 1 ist die Linie oben (Zustand ist ein) bei einer Null (Zustand ist aus) ist sie unten. Ist die Zahl zweimal die selbe ändert sich der Zustand nicht.*	99
Falsch	Regel, die kein Regelsystem beschreibt, sondern nur die korrekte Lösung nennt. Antworten, die undefinierte Begriffe nutzen, wie z.B. Schritte. – *Beim Phasenwechsel zwischen 0 und 1 wird die Spannung gehalten. Bei aufeinanderfolgenden gleichen Zahlen wechselt die Spannung in jedem Schritt.* – *Der Code legt beide Graphen ineinander. 1: Der obere Graph dominiert den unteren in der Nachricht und löscht den unteren. 0: Für 0 setzt*	0
Richtig	Regel, die ein vollständiges Regelsystem beschreibt, dass für das Beispiel gilt. Gleiche Signale in der Nachricht werden auf 0 abgebildet und unterschiedliche auf 1. Entspricht XOR oder dualer Addition. – *Legt man beide Muster übereinander, so werden im Code doppelte Striche als Strich unten abgebildet und einfache Striche als Strich oben.* – *0 & 0 => 0; 0 & 1 => 1; 1 & 0 => 1; 1 & 1 => 0* – *Eine 1 entspricht _|; eine 0 entspricht |_* – *Steht eine 1 unten, so wird das Muster in der Zeile oben umgedreht: aus |_ wird _| steht eine 0 unten, so bleibt das Muster gleich.* – *Vermutlich eine Kombination aus den unteren "Wellen" und den oberen, die 1 und 0 geben den "Verlauf" der unteren Welle an. Die Bildungsregel der neuen Welle erfolgt nach XOR (0+0=0, 0+1=1, 1+0=0, 1+1=0* – *Zwei Hochstellen erzeugen eine Tiefstelle. Zwei Tiefstellen erzeugen eine Tiefstelle. Ist nur eine Stelle eine Hochstelle, erzeugt das eine Hochstelle.*	1

Regel		Code
Abstraktionsniveau $5-4-2\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort In der Antwort wird nicht auf den Zusammenhang zwischen Nachricht und Code eingegangen, sondern nur die Nachricht beschrieben. – *Bei eine 1 ist die Linie oben (Zustand ist ein) bei einer Null (Zustand ist aus) ist sie unten. Ist die Zahl zweimal die selbe ändert sich der Zustand nicht.* – *Der Code legt beide Graphen ineinander. 1: Der obere Graph dominiert den unteren in der Nachricht und löscht den unteren. 0: Für 0 setzt*	99
Konkret	Nennung aller einzelnen Abbildungsvorschriften. – *0 & 0 => 0; 0 & 1 => 1; 1 & 0 => 1; 1 & 1 => 0* – *Zwei Hochstellen erzeugen eine Tiefstelle. Zwei Tiefstellen erzeugen eine Tiefstelle. Ist nur eine Stelle eine Hochstelle, erzeugt das eine Hochstelle.*	0
Spezifisch	Regel beinhaltet spezielle Charakteristika, wie z.B. die Repräsentation der Nachricht und des Codes. – *Legt man beide Muster übereinander, so werden im Code doppelte Striche als Strich unten abgebildet und einfache Striche als Strich oben.* – *eine 1 entspricht _|; eine 0 entspricht |_* – *Steht eine 1 unten, so wird das Muster in der Zeile oben umgedreht: aus |_ wird _| steht eine 0 unten, so bleibt das Muster gleich.* – *Beim Phasenwechsel zwischen 0 und 1 wird die Spannung gehalten. Bei aufeinanderfolgenden gleichen Zahlen wechselt die Spannung in jedem Schritt.*	1
Generisch	Antwort ist unabhängig von der Repräsentation oder dem Trägersignal. – *Gleiche Signale in der Nachricht werden auf 0 abgebildet und unterschiedliche auf 1.* – *Aus der Nachricht wird mittels dualer Addition der Code erzeugt.* – *Vermutlich eine Kombination aus den unteren "Wellen" und den oberen, die 1 und 0 geben den "Verlauf" der unteren Welle an. Die Bildungsregel der neuen Welle erfolgt nach XOR (0+0=0, 0+1=1, 1+0=0, 1+1=0*	2

Maschinen (Machines)

Seite 11

Sortieren

Regel		Code
Korrektheit $6-1\_relation\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Unklare, unpräzise oder fachliche falsche Antworten, wie z.B. fehlendes oder unvollständiges Sortierkriterium oder fehlende Nennung der zu sortierenden Elemente. – *Die Maschine fängt bei den ersten beiden Buchstaben an, wählt eins von beiden aus, danach nimmt er jeden zweiten Buchstaben, falls die Zeile zu Ende geht, nimmt er die restlichen Buchstaben von links nach rechts.* – *Gebe Buchstaben in zufälliger neuer Reihenfolge aus.* – *Sortiert nach Alphabet.* – *Die Maschine gibt die Buchstaben nach ihrer Stelle im Alphabet aus.*	0
Richtig	Elemente die sortiert werden und Aussage über das Sortierkriterium. *Die Maschine sortiert die Buchstaben in (aufsteigender) alphabetischer Reihenfolge.*	1
Abstraktionsniveau $6-1\_relation\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Handlungsbeschreibung in einer Schritt-für-Schritt-Anleitung. *Die Maschine fängt bei den ersten beiden Buchstaben an, wählt eins von beiden aus, danach nimmt er jeden zweiten Buchstaben, falls die Zeile zu Ende geht, nimmt er die restlichen Buchstaben von links nach rechts.*	0
Spezifisch	Regelbeschreibung, die sich auf Buchstaben stützt und für alle Beispiele gültig ist. – *Die Maschine sortiert die Buchstaben in (aufsteigender) alphabetischer Reihenfolge.* – *Gebe Buchstaben in zufälliger neuer Reihenfolge aus.* – *Die Maschine gibt die Buchstaben nach ihrer Stelle im Alphabet aus.*	1
Generisch	Regelbeschreibung, die auch für andere zu sortierende Elemente, wie z.B. Worte angewendet werden kann. – *Die Maschine sortiert den Input alphabetisch.*	2

Set

Regel		Code
Korrektheit $6-2\_relation\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Unklare, unpräzise oder fachliche falsche Antworten, wie z.B. fehlende Aussage über die Reihenfolge im Output oder das Duplikate entfernt werden. – *Die Maschine gibt die Buchstaben aus, welche bei der Eingabe benutzt wurden. Es wird die Reihenfolge beachtet, in welcher die Maschine auf einen neuen unbekannten Buchstaben trifft.* – *Die Maschine gibt die Buchstaben aus, die im Input vorkommen. Die Reihenfolge ist dabei die Leserichtung.* – *Die Funktion gibt beim Output jeden enthaltenen Buchstaben aus. Die Reihenfolge wird durch das erstmalige Auftreten eines Buchstaben bestimmt.* – *Es wird nur berücksichtigt, ob ein Buchstabe vorkommt, nicht wie oft. Danach werden sie in der Reihenfolge ihres Erst-Auftretens geordnet.* – *Die Maschine gibt nur Buchstaben aus, die nicht bereits ausgegeben wurden.* – *Doppelte Buchstaben werden gelöscht.* – *Die Maschine liest Buchstaben von links nach rechts ein und ignoriert dabei Buchstaben, die schon einmal eingelesen worden sind.* – *Die Maschine gibt die verschiedenen Zeichen des Inputs an.*	0
Richtig	Antworten die zeigen, dass die Reihenfolge erhalten bleibt und Duplikate entfernt werden. – *Die Maschine gibt jeden Buchstaben nur einmal aus. Wenn ein doppelter Buchstabe erkannt wird, wird dieser übersprungen und der nächste neue Buchstabe wird ausgegeben.*	1
Abstraktionsniveau $6-2\_relation\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Handlungsbeschreibung in einer Schritt-für-Schritt-Anleitung. *Die Maschine ließt Buchstaben von links nach rechts ein und ignoriert dabei Buchstaben, die schon einmal eingelesen worden sind.*	0
Spezifisch	Regelbeschreibung, die sich auf Buchstaben stützt und für alle Beispiele gültig ist. – *Die Maschine gibt nur Buchstaben aus, die nicht bereits ausgegeben wurden.* – *Die Maschine gibt jeden Buchstaben nur einmal aus. Wenn ein doppelter Buchstabe erkannt wird, wird dieser übersprungen und der nächste neue Buchstabe wird ausgegeben.*	1
Generisch	Regelbeschreibung, die auch für andere zu sortierende Elemente, wie z.B. Worte angewendet werden kann. – *Die Maschine gibt die verschiedenen Zeichen des Inputs an.*	2

Raster (On the Grid)

Seite 12 – 13

Verschobene Diagonale

Regel		Code
Korrektheit 7 − 1_*relation_correct*		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Unklare, unpräzise oder fachliche falsche Antworten. – *wenn die Spalten-Nr. > der Zeilen-Nr. ist.* – *wenn die Zeilen-Nr. um 1 größer ist als die Spalten-Nr.* – *fängt bei 2 an, und wird +1 nach rechts verschoben, sowie +1 nach unten.*	0
Richtig	Spaltennummer - 1 == Zeilennummer Spaltennummer == Zeilennummer + 1 – *... wenn die Spaltennummer eins höher ist als die Zeilennummer.* – *... wenn die Spaltennummer + 1 gleich der Zeilennummer ist.* – *... wenn die Zeilennummer eins kleiner als die Spaltennummer ist.* – *... wenn die Zeilennummer - 1 gleich der Spaltennummer ist.* – *Es wird ein x auf der um 1 nach oben verschobenen Nebendiagonale gesetzt.*	1
Abstraktionsniveau 7 − 1_*relation_level*		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Handlungsbeschreibung *fängt bei 2 an, und wird +1 nach rechts verschoben, sowie +1 nach unten.*	0
Spezifisch	Einschränkung auf Beispiele, wie z.B. Größe des Rasters.	1
Generisch	Allgemeine Regelbeschreibung oder Nutzung von Formeln / Ausdrücken. – *(Start mit Zeile und Spalte = 1) wenn die Zeilen-Nummer gleich der Spaltennummer weniger 1 ist.* – *... wenn die Zeilennummer eins kleiner als die Spaltennummer ist.* – *Es wird ein x auf der um 1 nach oben verschobenen Nebendiagonale gesetzt.*	2

Dreieck

Regel		Code
Korrektheit $7 - 2\_relation\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv, Dreieck*	99
Falsch	Unklare, unpräzise oder fachliche falsche Antworten. * Zeilen-Nr. <= Spalten-Nr. * Spalten-Nr. - Zeilen-Nr. >= 0 – *Die Zeilennummer zeigt in wie vielen Spalten ein x gesetzt wird.* – *Ein 'x' wird gesetzt, wenn die Zahl bei den Zeilennummern, die Anzahl der Spalte entsprechen.* – *Die gesamte Spalte wird mit 'x' gefüllt, sobald Spaltennr. = Zeilennr.*	0
Richtig	Angabe einer fachlich korrekten Regel. * Zeilen-Nr. $\geq$ Spalten-Nr. * Spalten-Nr. $\leq$ Zeilen-Nr. – *Nummer der Zeile >= Nummer der Spalte* – *Ein x wird in die Zeilen gesetzt, bis hin zu der Zelle, deren Spalte die selbe Nummer wie die Zeile hat.* – *Ein x wird gesetzt, wenn die Spaltennummer in die Zeilennummer passt.* – *x füllt alle Einträge in der unteren 5x5 Dreiecksmatrix aus.* – *Ein x wird gesetzt, bis Zeile == Spalte*	1
Abstraktionsniveau $7 - 2\_relation\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv, Dreieck*	99
Konkret	Handlungsbeschreibung – *Ein x wird in die Zeilen gesetzt, bis hin zu der Zelle, deren Spalte die selbe Nummer wie die Zeile hat.* – *x wird gesetzt, wenn die Zeilen-Nr. gleich der Spaltennummer ist zusätzlich werden alle leeren Spalten links des x mit einem x gesetzt.* – *Eine Spalte wird mit so vielen 'x' aufgefüllt, wie die Zeilennummern vorweg angibt.*	0
Spezifisch	Einschränkung auf Beispiele, wie z.B. Größe des Rasters. – *x füllt alle Einträge in der unteren 5x5 Dreiecksmatrix aus.*	1
Generisch	Allgemeine Regelbeschreibung oder Nutzung von Formeln / Ausdrücken. – *Nummer der Zeile >= Nummer der Spalte* – *Die Zeilennummer zeigt in wie vielen Spalten ein x gesetzt wird.* – *Ein x wird gesetzt, bis Zeile == Spalte*	2

Schachbrett

Regel		Code
Korrektheit $7 - 3\_relation\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv, Schachbrettmuster*	99
Falsch	Unklare, unpräzise oder fachliche falsche Antworten. – *x wird gesetzt, bei ungerader Zeilenzahl, bei Zahlen die nur durch sich selbst teilbar sind. Bei gerader Zeilenzahl wird x gesetzt bei Zahlen, die durch 2 teilbar sind.* – *Ein x wird gesetzt, wenn das Ergebnis aus Spalten- und Zeilennummer gerade ist.* – *Ein x in jede Zelle, deren Spaltennummer gleich der Zeilennummer ist und in jede Zelle einer Zeile, deren Spaltennummer um 2 kleiner oder größer ist als deren Zeilennummer.* – *Wenn die Zahlennr. gerade ist, bestehen Spaltennr. nur aus geraden Zahlen und umgekehrt.* – *Ein 'x' wird dann gesetzt, wenn die Zeile % 2 = 0 ist.* – *Ist die Spalten-Nummer gerade, so werden alle Spalten-Nummern sowie Zeilennummern, die nicht über die Start-Nummer hinausgehen gesetzt.* – *Wenn die Spalte ungerade ist, sind die Zeichen der Zeile ungerade. Selbes für gerade Zahlen.*	0
Richtig	Spalten- und Zeilennummer gleichzeitig gerade oder ungerade. Ergebnis von Spaltennummer + Zeilennummer ist gerade. – *Ein x wird gesetzt wenn Spaltennummer + Zeilennummer eine gerade Zahl ergeben.* – *Ein x wird gesetzt, wenn Zeilen-Nr. ungerade und Spalten-Nr. auch ungerade. Zeilen-Nr. gerade und Spalten-Nr. auch gerade.* – *Ein x wird gesetzt, wenn in Spalte und Zeile ein ungerades oder gerades Zahlenpaar entsteht.* – *Ein 'x' wird dann gesetzt, wenn die Zeile % 2 = Spalte % 2 ist.* – *Ein 'x' wird dann gesetzt, wenn Zeilennummer + Spaltennummer eine gerade Zahl ergibt.* – *x werden im Schachbrettmuster gesetzt, beginnend bei der linken oberen Ecke.*	1

Regel		Code
Abstraktionsniveau $7 - 3\_relation\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv, Schachbrett(muster)*	99
Konkret	Handlungsbeschreibung oder nur teilweise Verallgemeinerung kombiniert mit konkreten Werten. – *Bei allen geraden Zeilennummern wird ein 'x' bei der Spalte 2 und 4 gesetzt. Bei allen ungeraden Zeilennummern wird ein 'x' bei 1, 3, 5 gesetzt.* – *x werden im Schachbrettmuster gesetzt, beginnend bei der linken oberen Ecke.*	0
Spezifisch	Einschränkung auf Beispiele, wie z.B. Größe des Rasters. *Ein x in jede Zelle, deren Spaltennummer gleich der Zeilennummer ist und in jede Zelle einer Zeile, deren Spaltennummer um 2 kleiner oder größer ist als deren Zeilennummer.*	1
Generisch	Allgemeine Regelbeschreibung oder Nutzung von Formeln / Ausdrücken. – *Ein x wird gesetzt, wenn Zeilen-Nr. ungerade und Spalten-Nr. auch ungerade. Zeilen-Nr. gerade und Spalten-Nr. auch gerade.* – *Ein x wird gesetzt wenn Spaltennummer + Zeilennummer eine gerade Zahl ergeben.* – *Ein 'x' wird dann gesetzt, wenn die Zeile % 2 = 0 ist.* – *Falls Zeilenindex gerade → Bei allen ein Kreuz in der Zeile setzt wo Spaltenindex gerade.* – *Wenn die Spalte ungerade ist, sind die Zeichen der Zeile ungerade. Selbes für gerade Zahlen.*	2

Baum

Regel		Code
Korrektheit $7 - 4\_relation\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Unklare, unpräzise oder fachliche falsche Antworten. – *#Zeile = Interval(6 - #Zeile, 6 + #Zeile)* – *Ein x wird gesetzt, wenn die Summe der Zeilenzahl und die Spaltenzahl eine Zahl ergibt, die gleich oder größer 6 ist.* – *Je Zeile wird von 5 addiert und subtrahiert in Zeile 1 um 0, Zeile 2 um 1, usw.* *5* *4 -1 +1 16* *[...]* *Das Ergebnis der vorherigen Zeile bleibt im Term der nächsten Berechnung.*	0
Richtig	Fachlich korrekte Antworten beinhalten * Zusammenhang zwischen Spalten- und Zeilenzahl * Bedingung gibt ein Intervall an – *Sei i die Zeilennummer und j die Spaltennummer. Das Muster ist möglich für k Zeilen und $2k-1$ Spalten. (Spalte k ist in der Mitte, hier k=5) Ein x wird in Zeile i und Spalte j gesetzt, wenn $k-i+1 \leq j \leq k+i-1$.* – *In Zeile i die Spalte aus Intervall [k-i+1, k+i-1], bei Mittelpunkt k.*	1
Abstraktionsniveau $7 - 4\_relation\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Handlungsbeschreibung – *Je Zeile wird von 5 addiert und subtrahiert in Zeile 1 um 0, Zeile 2 um 1, usw.* *5* *4 -1 +1 16* *[...]* *Das Ergebnis der vorherigen Zeile bleibt im Term der nächsten Berechnung.*	0
Spezifisch	Einschränkung auf Beispiele, wie z.B. Größe des Rasters oder konkreten Mittelpunkt. – *Ein x wird gesetzt, wenn die Summe der Zeilenzahl und die Spaltenzahl eine Zahl ergibt, die gleich oder größer 6 ist.*	1
Generisch	Allgemeine Regelbeschreibung oder Nutzung von Formeln / Ausdrücken. – *Sei i die Zeilennummer und j die Spaltennummer. Das Muster ist möglich für k Zeilen und $2k-1$ Spalten. (Spalte k ist in der Mitte, hier k=5) Ein x wird in Zeile i und Spalte j gesetzt, wenn $k-i+1 \leq j \leq k+i-1$.*	2

Passwort-Sicherheitsstatus (Pass phrase security status)

Seite 14 – 15

Passwort	Status	Capital letter	Small letter	Char	Digit	Length
Servus	weak	x	x			6
Baum36	weak	x	x		x	6
T?o8	weak	x	x	x	x	4
$5&-!?	weak			x	x	6
SteffisPassworT	medium	x	x			15
!nformatik	medium		x	x		10
urlaubindien2013	medium		x		x	16
studium	medium		x			7
20!7-!0	medium			x	x	7
B3!$p!3l	strong	x	x	x	x	8
§Passwort&	strong	x	x	x		10
TH3OPW!	strong	x		x	x	7
§4%68?9!a	strong		x	x	x	9
6PassWort	strong	x	x		x	9

Passwort	Status	Length Abstract	Number of Components	Number of Changes
Servus	schwach	<= 6	2	1
Baum36	schwach	<= 6	3	2
T?o8	schwach	<= 6	4	3
$5&-!?	schwach	<= 6	2	4
SteffisPassworT	mittel	> 6	2	4
!nformatik	mittel	> 6	2	1
urlaubindien2013	mittel	> 6	2	1
studium	mittel	> 6	1	0
20!7-!0	mittel	> 6	2	4
B3!$p!3l	stark	> 6	4	6
§Passwort&	stark	> 6	3	3
TH3OSPW!	stark	> 6	3	3
§4%68?9!a	stark	> 6	3	7
6PassWort	stark	> 6	3	4

Bestandteile identifizieren

Bestandteile		Code
Korrektheit $8-1\_component\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Unklare, unpräzise oder fachliche falsche Antworten, sowie Antworten die – ... die Länge nennen. – ... weniger als 4 Bestandteile nennen.	0
Richtig	Muss folgende Bestandteile enthalten – *Großbuchstabe* – *Kleinbuchstabe* – *Ziffern* | *Zahlen* – *Sonderzeichen* | *Satz- und Sonderzeichen* und die Länge darf nicht genannt werden.	1
Abstraktionsniveau $8-1\_component\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Nennung aller einzelnen Zeichen aus den Beispielen ohne Oberbegriffe zu finden oder Oberbegriffe, wie z.B. Worte verwendet.	0
Spezifisch	Nennung des Oberbegriffes und anschließende Einschränkung auf Zeichen die in den Beispielen vorkommen. – *Aus Groß- und Kleinbuchstaben, Ziffern und Sonderzeichen in unserem Fall !,?,&,§,%*	1
Generisch	Die einzelnen Bestandteile sind als solche abstrahiert und in der Sprache verwendet. – *Buchstaben, Ziffern, Sonderzeichen* – *Aus Groß- und Kleinbuchstaben, Ziffern und Sonderzeichen*	2

Unterschiede erkennen und beschreiben

Unterschied schwach und mittel		**Code**
Korrektheit $8-2-1\_relation\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Unklare, unpräzise oder fachliche falsche Antworten, wie z.B. Beschreibung einer einzelnen Passwort-Gruppe. – *Mittel hat mindestens 7 Stellen.* – *Länge, Sonderzeichen* – *Willkürliche Groß-/Kleinschreibung und Sonderzeichen als Ersatz für Buchstaben und Zahlen genutzt.*	0
Richtig	Antworten, die die Länge als Unterscheidungsmerkmal nennen. Schlagwörter werden akzeptiert. – *Es werden mehr Zeichen verwendet.* – *(Passwort)Länge* – *Die Länge des PW.* – *Entscheidend ist hier die Länge der Passwörter. Wenn sie 7 oder mehr Zeichen haben werden sie als mittel eingestuft.*	1
Abstraktionsniveau $8-2-1\_relation\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv, Länge*	99
Konkret	Charakterisierung einzelner Beispielpasswörter oder Passwort-Gruppen durch Nutzung mehrerer Fallunterscheidungen. – *Mittel hat mindestens entweder zwei Großbuchstaben, nur Kleinbuchstaben oder mehr als eine Zahl, bei Passwörtern ohne Buchstaben.* – *Im Mittel sind sie meist länger und haben mehrsilbige Wörter mit Groß- und Kleinbuchstaben bunt gemixt oder mal eine Zahl oder ein Satzzeichen statt Buchstaben.*	0
Spezifisch	Beschreibung der beiden Passwort-Gruppen separat. – *Mittel hat mindestens 7 Stellen* – *Die schwachen sind maximal 6 Charakter lang, die mittleren mindestens 7*	1
Generisch	Benennung der Eigenschaft als solche. – *Alle Passwörter von Mittel sind länger.* – *Entscheidend ist hier die Länge der Passwörter. Wenn sie 7 oder mehr Zeichen haben, werden sie als mittel eingestuft.* – *Die Länge des PW* – *Länge*	2

Unterschied stark und mittel		Code
Korrektheit $8-2-2\_relation\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv, Zeichen*	99
Falsch	Unklare, unpräzise oder fachliche falsche Antworten, wie z.B. Beschreibung einer einzelnen Passwort-Gruppe. – *Zahlen als Buchstaben verwendet, Sonderzeichen gemischt mit diesen "Wörtern".* – *Ein starkes Passwort muss 7 oder mehr Zeichen haben, wovon ein Zeichen ein Sonderzeichen sein muss, eins eine Zahl und ein anderes ein Buchstabe.* – *Mindestens 1 Zahl oder 1 Sonderzeichen bei stark. 8 Stellen* – *3 der oben genannten Bestandteile müssen für stark gewählt sein, bei Mittel ist das egal.* – *Stark: besteht aus mindestens 3 verschiedenen Bestandteilen, schwach: besteht aus höchstens 2 verschiedenen Bestandteilen.* – *Sie unterscheiden sich an der Anzahl der Zeichen und Zahlen und der Abwechslung von Buchstaben, Zahlen und Zeichen.* – *Groß/Kleinschreibung, Anzahl Zeichen und Kombination von Zeichen und Buchstaben.*	0
Richtig	Bei starken Passwörtern werden mindestens 3 Bestandteile verwendet, bei mittleren maximal 2. *Es werden mehr Bestandteile verwendet.*	1
Abstraktionsniveau $8-2-2\_relation\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Charakterisierung einzelner Beispielpasswörter oder Passwort-Gruppen durch Nutzung mehrerer Fallunterscheidungen. – *Gleiche Bestandteile. Starke Passwörter wechseln öfter zwischen den Bestandteilen hin und her. Alle mittleren Passwörter benutzen Wörter die eine feste definierte Bedeutung haben. Starke Passwörter sind frei erfundene sinnfreie Aneinanderreihungen von Buchstaben, Sonderzeichen und Zahlen.* – *Zahlen als Buchstaben verwendet, Sonderzeichen gemischt mit diesen "Wörtern".* – *Ein starkes Passwort muss 7 oder mehr Zeichen haben, wovon ein Zeichen ein Sonderzeichen sein muss, eins eine Zahl und ein anderes ein Buchstabe.*	0
Spezifisch	Beschreibung der beiden Passwort-Gruppen separat. – *3/4 Bestandteilbenutzung bei stark, 2/4 bei mittel*	1
Generisch	Benennung der Eigenschaft als solche. – *Es werden mehr Bestandteile verwendet* – *Anzahl der verwendeten Zeichentypen*	2

Regeln ableiten

Regel schwache Passwörter		Code
Korrektheit $8-3-1\_relation\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Ungenau, fehlerhaft, unpräzise, mehrere nicht notwendige oder nicht alle notwendigen Regeln genannt. Ein Passwor wird als schwach eingestuft, wenn ... – *... es 6 Zeichen lang ist oder kürzer, ganze, von der Groß- und Kleinschreibung richtig geschriebene Wörter beinhaltet.* – *... es 6 Zeichen oder weniger hat und bei über/gleich 5 Zeichen keine Abwechslung im Zeichentyp liefert.* – *... es kurz, leicht zu erraten und zu wenig komplex ist.* – *... es zu kurz ist.* – *... es kurz ist und keine oder kaum Variation an Bestandteilen aufweist.* – *... deutsches Wort korrekt geschrieben (evtl. hinten mit Zahl).* – *... die Zeichen oder die Buchstaben oder die Zahlen nur einmal vorkommen.*	0
Richtig	Hinreichend: weniger als 7 Zeichen | Länge ≤ 6 *keine weiteren Regeln notwendig.* Ein Passwor wird als schwach eingestuft, wenn ... *... es unter 7 Zeichen besitzt.*	1
Abstraktionsniveau $8-3-1\_relation\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Eigenschaften wie Länge nicht genau spezifiziert oder bekannte Konzepte wie 'Wörter' zu Erklärung herangezogen. Ein Passwor wird als schwach eingestuft, wenn ... – *es 6 Zeichen lang ist oder kürzer, ganze, von der Groß- und Kleinschreibung richtig geschriebene Wörter beinhaltet.* – *... es kurz, leicht zu erraten und zu wenig komplex ist.* – *... es zu kurz ist.* – *... aus einem Wort besteht.*	0
Spezifisch	Zusätzliche Regeln eingeführt, um die gegebenen Beispiele spezifischer zu beschreiben. Ein Passwor wird als schwach eingestuft, wenn ... – *... weniger als 7 Zeichen benutzt worden sind, nur ein Großbuchstabe vorkommt, das Passwort mit einem Großbuchstaben anfängt, weniger als 2 Zahlen verwendet werden.* – *... die Zeichen oder die Buchstaben oder die Zahlen nur einmal vorkommen.*	1
Generisch	Regelbeschreibung, die alle gegebenen Beispiele klassifiziert ohne zusätzliche spezifische Einschränkungen zu beschreiben. Ein Passwor wird als schwach eingestuft, wenn ... – *es unter 7 Zeichen besitzt.*	2

Regel mittlere Passwörter		**Code**
Korrektheit $8-3-2\_relation\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Ungenau, fehlerhaft, unpräzise, mehrere nicht notwendige oder nicht alle notwendigen Regeln genannt.. Ein Passwor wird als mittel eingestuft, wenn ... – *... es mindestens 7 Zeichen hat* – *... mehrere Wörter aneinander gereiht werden; Sonderzeichen als Buchstaben bzw. Ziffernersatz benutzt werden.* – *... mehr als 7 Zeichen benutzt worden sind, zwischen Groß- und Kleinschreibung gewechselt wird, einzelne Buchstaben eines Wortes mit klarer Definition durch Sonderzeichen ausgetauscht wurden.* – *... es länger ist und eine höhere Komplexität aufweist.*	0
Richtig	Länge > 6 UND maximal 2 verschiedene Bestandteile Ein Passwor wird als mittel eingestuft, wenn ... – *... es mindestens 7 Zeichen hat und 2 der 4 Bestandteile nutzt.*	1
Abstraktionsniveau $8-3-2\_relation\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Beschreibung der Wörter, aber keine Regel. Ein Passwor wird als mittel eingestuft, wenn ... – *... mehr als 7 Zeichen benutzt worden sind, zwischen Groß- und Kleinschreibung gewechselt wird, einzelne Buchstaben eines Wortes mit klarer Definition durch Sonderzeichen ausgetauscht wurden.* – *... mehrere Wörter aneinander gereiht werden; Sonderzeichen als Buchstaben bzw. Ziffernersatz benutzt werden.* – *... es lang genug ist, aber keine Spezialzeichen vorhanden sind.* – *... es nur mindestens 7 Zeichen hat oder nur mind 7 Zeichen und Groß-Kleinschreibung hat oder nur 1 Sonderzeichen, Rest Kleinschreibung und mindestens 7 Zeichen hat.*	0
Spezifisch	Zusätzliche Regeln eingeführt, um die gegebenen Beispiele spezifischer zu beschreiben. Ein Passwor wird als mittel eingestuft, wenn ... – *... mehr als 7 Zeichen, unregelmäßige Groß- und Kleinschreibung.* – *... es 7 Zeichen oder mehr hat, aber keine Ziffer und/oder keinen Großbuchstaben besitzt.*	1
Generisch	Regelbeschreibung, die alle gegebenen Beispiele klassifiziert ohne zusätzliche spezifische Einschränkungen zu beschreiben. Ein Passwor wird als mittel eingestuft, wenn ... – *... es mindestens 7 Zeichen hat und 2 der 4 Bestandteile nutzt.* – *... es mehr als 6 Zeichen beträgt, aber nur aus Zeichen von bis zu 2 verschiedenen Zeichengruppen besteht.*	2

Regel starke Passwörter		Code
Korrektheit $8-3-3\_relation\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Ungenau, fehlerhaft, unpräzise, mehrere nicht notwendige oder nicht alle notwendigen Regeln genannt. Ein Passwor wird als stark eingestuft, wenn ... – *... es kein Wort ergibt.* – *... hohe Anzahl aller 4 Bestandteile hat.* – *... mehr als zwei unterschiedliche Arten von verwendbaren Zeichen verwendet wurden.* – *... es 7 Zeichen oder mehr hat und mindestens eine Ziffer und mindestens einen Großbuchstaben besitzt.* – *... Anzahl der verschiedenen Zeichentypen* ≥ 3 *und Länge* ≥ 8 – *... es sehr komplex ist, nicht zu erraten ist und eine optische Logik aufweist, die durch einen Computer nicht erkennbar ist.* – *... alle Kriterien für 'mittel' kombiniert werden.* – *... es aus allen möglichen Bestandteilen und mindestens 8 Buchstaben besteht.* – *... viele unterschiedliche Zeichen vorkommen, keine Wdh. vorhanden sind.* – *... es länger als 6 Zeichen ist und je mindestens 3x den Zeichentyp wechselt.* – *... es mehr als 6 Charaktere hat und 3 Bestandteilswechsel.* – *... es mindestens 7 Zeichen hat und 3 der 4 Bestandteile nutzt.* – *... es länger als 6 Zeichen ist und je mindestens 3x den Zeichentyp wechselt.* – *... es mehr als 6 Charaktere hat und 3 Bestandteilswechsel.* – *... es mindestens 7 Zeichen hat und 3 der 4 Bestandteile nutzt.*	0
Richtig	Mindestens 3 verschiedene Bestandteile und Länge > 6 Ein Passwor wird als stark eingestuft, wenn ... – *... es ein mittleres Passwort ist und mindestens 3 der Bestandteile beinhaltet.* – *... es mehr als 6 Zeichen und mindestens 3/4 Bestandteile hat.*	1

Regel starke Passwörter		**Code**
Abstraktionsniveau $8-3-3\_relation\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Eigenschaften wie Länge nicht genau spezifiziert oder bekannte Konzepte zur Erklärung herangezogen. Ein Passwor wird als stark eingestuft, wenn ... – *... es kein Wort ergibt.* – *... hohe Anzahl aller 4 Bestandteile hat.* – *... Sonderzeichen, Großbuchstaben, Kleinbuchstaben, Zahl* $\leftarrow$ *3 davon* – *... es sehr komplex ist, nicht zu erraten ist und eine optische Logik aufweist, die durch einen Computer nicht erkennbar ist.* – *...*$\geq$ *7 Zeichen; alle/fast alle möglichen Bestandteile enthalten; Worte nur in Kombination mit Zeichen / Großbuchstaben.* – *... viele unterschiedliche Zeichen vorkommen, keine Wdh. vorhanden sind.*	0
Spezifisch	Zusätzliche Regeln eingeführt, um die gegebenen Beispiele spezifischer zu beschreiben. Oder die Bestandteile einzeln genannt statt den übergeordneten Begriff 'Bestandteile' zu verwenden. Ein Passwor wird als stark eingestuft, wenn ... – *... es 7 oder mehr Zeichen hat, wobei ein Sonderzeichen, ein Buchstabe und eine Zahl vorhanden sein muss.* – *... es mindestens 2 Sonderzeichen und Groß-/Kleinschreibung enthält oder Zahl und Groß-/Kleinschreibung.*	1
Generisch	Regelbeschreibung, die alle gegebenen Beispiele klassifiziert ohne zusätzliche spezifische Einschränkungen zu beschreiben. Die einzelnen Bestandteile sind als solche abstrahiert und in der Sprache verwendet. Ein Passwor wird als stark eingestuft, wenn ... – *... es mehr als 6 Charaktere hat und (mindestens) 3 Bestandteile hat.* – *... es ein mittleres Passwort ist und mindestens 3 der Bestandteile beinhaltet.* – *... es länger als 6 Zeichen ist und je mindestens 3x den Zeichentyp wechselt.* – *... es mehr als 6 Charaktere hat und 3 Bestandteilswechsel.* – *... es mindestens 7 Zeichen hat und 3 der 4 Bestandteile nutzt.*	2

Zaubertränke (Potions)

Seite 16 – 17

Hinweis: Hier werden die Regeln und die Bewertung als Einheit betrachtet.

Kariert – Gemeinsamkeit

kariert – gemeinsam		**Code**
Korrektheit $9 - 1\_relation\_correct$ Die MC-Frage, ob das Kriterium genau klassifiziert, muss auch korrekt beantwortet werden.		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Kombination aus Regel und Bewertung ist falsch. – *2. und 5. Stelle ein X | NEIN* – *Gerade Anzahl von X* *OOOIXXO macht kariert* – *mindestens 2 X und 1 O* *XXXIXXX macht kariert* Es lassen sich Gegenbeispiele für die Aussage finden.	0
Richtig	*Eines dieser* – *Bestehen aus O, X und/oder I | NEIN* – *Max. 2 I | NEIN* – *Gerade Anzahl von X | NEIN* – *Anzahl* $X \geq 2, < 7$ *| NEIN* – *Mindestens ein X | NEIN* – *Mehrere aufeinanderfolgende X | NEIN* – *Mindestens 2 X nebeneinander | NEIN*	1

kariert – gemeinsam		**Code**
Abstraktionsniveau $9-1\_relation\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Nutzung der gegebenen Zaubertränke oder Regelbeschreibung mit 3 oder mehr Fallunterscheidungen (grenzt an Aufzählung). – *Mindestens 2 X und 1 I oder 4 X (fiktives Beispiel)*	0
Spezifisch	Arbeit mit konkreten Positionen und spezifischen Anzahlen, anstelle von Klassifizierungen. – *Ein I an der 5. Stelle* – *2. und 5. Stelle ein X*	1
Generisch	Nutzung allgemeiner Formulierungen wie z.B. Mitte unabhängig von der Länge des Wortes oder gerade Anzahl. – *Ein I in der Mitte* – *Gerade Anzahl von X* – *Mindestens 2 X und 1 O* – *Anzahl X* $\geq 2, < 7$ – *Mindestens ein X* – *Mehrere aufeinanderfolgende X* – *Mindestens 2 X nebeneinander*	2

Schwerelos – Gemeinsamkeit

schwerelos – gemeinsam		**Code**
Korrektheit $9-2\_relation\_correct$ Die MC-Frage, ob das Kriterium genau klassifiziert, muss auch korrekt beantwortet werden.		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Kombination aus Regel und Bewertung ist falsch. – *I in der Mitte, O an beiden Enden | NEIN*	0
Richtig	*Eines dieser* – *I in der Mitte, O an beiden Enden | JA* – *I an Position 4, O an Position 1 und 7 | JA* – *I an Position 4 | NEIN* – *I in der Mitte | NEIN* – *O an beiden Enden | NEIN* – *Mindestens 2 O, ein I an Stelle 4 | NEIN* – *Mindestens ein I, ein O am Anfang und Ende | JA*	1
Abstraktionsniveau $9-2\_relation\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Nutzung der gegebenen Zaubertränke oder Regelbeschreibung mit 3 oder mehr Fallunterscheidungen (grenzt an Aufzählung).	0
Spezifisch	Arbeit mit konkreten Positionen und spezifischen Anzahlen, anstelle von Klassifizierungen. – *I an Position 4, O an Position 1 und 7* – *I an Position 4* – *Mindestens 2 O, ein I an Stelle 4*	1
Generisch	Allgemeine Formulierung wie z.B. Mitte unabhängig von der Länge des Wortes. – *I in der Mitte, O an beiden Enden* – *I in der Mitte* – *O an beiden Enden* – *Mindestens ein I, ein O am Anfang und Ende*	2

Leuchten – Unterschied

leuchten – unterscheiden		Code
Korrektheit $9-3\_relation\_correct$ Die MC-Frage, ob das Kriterium genau klassifiziert, muss mit JA beantwortet werden.		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Nutzung eines falschen Begriffs oder Beschreibung von Zaubertränken, die gar keine Wirkung haben. Wurde NEIN angekreuzt, ist die Antwort immer als falsch zu bewerten. – *Anagramm* – *Ein Buchstabe kommt mindestens 4x vor | NEIN* – *An erster und letzter Stelle entweder ein O oder ein I | NEIN* – *Entweder nur ein Buchstabe eines Wertes oder keiner. 2 Buchstaben beliebig oft, einer 1 oder kein Mal.*	0
Richtig	Antworten die JA angekreuzt haben und eine der folgenden Eigenschaften als Unterscheidungsmerkmal nennen: symmetrisch, gespiegelt, von rechts und links gelesen das gleiche Wort, Palindrom. – *An mittlerer Position gespiegelt.* – *Von vorne oder hinten gelesen identisch.* – *Aufbau von Buchstaben symmetrisch.* – *Buchstabenfolge symmetrisch* – *Symmetrische Anordnung* – *Palindrom* – *Kann von vorne und hinten gelesen werden ohne dass sich etwas ändert.* – *Gespiegelt an der 4. Stelle als Spiegelachse.*	1

leuchten – unterscheiden		**Code**
Abstraktionsniveau $9 - 3\_relation\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Nutzung der gegebenen Zaubertränke oder Regelbeschreibung mit 3 oder mehr Fallunterscheidungen. (grenzt an Aufzählung) – *Entweder nur ein Buchstabe eines Wertes oder keiner. 2 Buchstaben beliebig oft, einer 1 oder kein Mal.*	0
Spezifisch	Arbeit mit konkreten Positionen (z.B. 4 als Mittelpunkt) und spezifischen Anzahlen, anstelle von Klassifizierungen. – *Gespiegelt an der 4. Stelle als Spiegelachse.*	1
Generisch	Allgemeine Formulierung, ohne Bezug auf konkrete Beispiele oder Nutzung von Begriffen. – *Aufbau von Buchstaben symmetrisch.* – *Buchstabenfolge symmetrisch* – *Symmetrische Anordnung* – *Palindrom* – *Anagramm* – *Kann von vorne und hinten gelesen werden ohne dass sich etwas ändert.* – *Ein Buchstabe kommt mindestens 4x vor.* – *An erster und letzter Stelle entweder ein O oder ein I.*	2

Automatisierung (Automation)

Seite 18

Konstante Länge

<table>
<tr><th colspan="2">Wörter der Länge 10</th><th>Code</th></tr>
<tr><td colspan="3">Korrektheit $10 - 1\_relation\_correct$</td></tr>
<tr><td>Leer</td><td>Leeres Antwortfeld oder Antwort vollständig durchgestrichen</td><td>999</td></tr>
<tr><td>Ohne Bezug</td><td>Antwort ohne Bezug zur Frage oder Schlagwort
Weiß ich nicht, Intuitiv</td><td>99</td></tr>
<tr><td>Falsch</td><td>Unvollständige Beschreibungen oder nicht korrekte Parametrisierung.
– n=0; Vergleich Buchstabe (n+1) mit Buchstabe (10-n). Danach n um 1 hochzählen, solange n=10.
Start bei 0, Ende bei 10 → 11 Vergleich statt 10.</td><td>0</td></tr>
<tr><td>Richtig</td><td>Vollständige und korrekte Handlungsbeschreibung.
Alle Schritte müssen genannt werden!!
– Vergleiche Buchstabe 1 mit Buchstabe 10
Vergleiche Buchstabe 2 mit Buchstabe 9
– Vergleiche den ersten mit dem letzten Buchstaben, den zweiten mit dem vorletzten Buchstaben, ...

Korrekte Handlungsbeschreibung mit Definition und korrekter Wahl der Parameter (Start, Ende, Position der Buchstaben, Inkrement).

<table>
<tr><th>Start n</th><th>Ende</th><th>Erster Buchstabe</th><th>Letzter Buchstabe</th></tr>
<tr><td><code>n=0</code></td><td><code>n=9</code></td><td><code>n+1</code></td><td><code>10-n</code> ODER <code>länge-n</code></td></tr>
<tr><td><code>n=0</code></td><td><code>n=9</code></td><td><code>n</code></td><td><code>9-n</code> ODER <code>länge-n-1</code></td></tr>
<tr><td><code>n=1</code></td><td><code>n=10</code></td><td><code>n</code></td><td><code>10-n+1</code> ODER <code>länge-n+1</code></td></tr>
<tr><td><code>n=1</code></td><td><code>n=10</code></td><td><code>n-1</code></td><td><code>10-n</code> ODER <code>länge-n</code></td></tr>
</table>

– n=1; Vergleiche n mit 10-n+1; erhöhe n um 1 und beginne wieder mit dem Vergleich, solange bis n=10
– n=0; Vergleiche Buchstabe n+1 mit Buchstabe Länge-n; erhöhe n um 1; vergleiche erneut, solange bis n=9
– for(int i=0; i<wort.length; i++) {
    Vergleiche wort(i) mit wort(length-1-i)
}</td><td>1</td></tr>
</table>

Wörter der Länge 10		**Code**
Abstraktionsniveau $10 - 1\_relation\_level$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Vollständige Handlungsbeschreibung mit konkreten Plätzen der Buchstaben. – *Vergleiche Buchstabe 1 mit Buchstabe 10.* *Vergleiche Buchstabe 2 mit Buchstabe 9, ...* (*Vollständige Liste bis mindestens Schritt 5.*) – *Vergleiche den ersten mit dem letzten Buchstaben, den zweiten mit dem vorletzten Buchstaben, ...* (*Vollständige Liste bis mindestens Schritt 5.*)	0
Spezifisch	Handlungsbeschreibung, welche die Länge konstant hat, aber die Plätze nicht mehr explizit nennt. – *n=1; Vergleiche n mit 10-n+1; erhöhe n um 1 und beginne wieder mit dem Vergleich*	1
Generisch	Handlungsbeschreibung, welche Parameter für die Position sowie die Länge einführt. – *Vergleiche Buchstabe n+1 mit Buchstabe Länge-n;* *erhöhe n um 1; vergleiche erneut, solange bis n den Wert Länge/2 hat* – *for(int i=0; i<wort.length; i++) {* *Vergleiche wort(i) mit wort(length-1-i)* *}*	2
Darstellungsform $10 - 1\_relation\_type$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Text	*Vergleiche Buchstabe n+1 mit Buchstabe Länge-n;* *erhöhe n um 1; vergleiche erneut, solange bis n den Wert Länge/2 hat*	0
Diagramm		1
Programm-Code	*for(int i=0; i<wort.length; i++) {* *Vergleiche wort(i) mit wort(length-1-i)* *}*	2

Variable Length

Wörter gerader Länge		Code
Korrektheit $10-2\_relation\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Nicht korrekte oder fehlende Parametrisierung. – *n=0; Vergleich Buchstabe (n+1) mit Buchstabe (Länge-n). Danach n um 1 hochzählen, solange bis n=Länge.* *Start bei 0, Ende bei Länge → Länge+1 Vergleich statt Länge.* – *n=0; Vergleich Buchstabe (n+1) mit Buchstabe (Länge-n). Danach n um 1 hochzählen, solange bis n=9.* – *Vergleiche Buchstabe 1 mit Buchstabe 10* *Vergleiche Buchstabe 2 mit Buchstabe 9* – *Vergleiche den ersten mit dem letzten Buchstaben, den zweiten mit dem vorletzten Buchstaben, ...* (*Vollständige Liste bis mindestens Schritt 5.*)	0
Richtig	Korrekte und parametrisierte Handlungsbeschreibung mit Definition der Parameter (Start, Ende, Buchstaben, Erhöhung). – *n=1; Vergleiche n mit Länge-n+1; erhöhe n um 1 und beginne wieder mit dem Vergleich, solange bis n=Länge* – *n=0; Vergleiche Buchstabe n+1 mit Buchstabe Länge-n; erhöhe n um 1; vergleiche erneut, solange bis n=Länge-1* – *for(int i=0; i<wort.length; i++) {* *Vergleiche wort(i) mit wort(length-1-i)* *}* – *Das Wort wird halbiert. Die zweite Hälfte wird umgedreht. Nun kann Stelle 1 bis 5 der jeweiligen Hälften verglichen werden.*	1

Wörter gerader Länge		Code
Abstraktionsniveau 10 − 2_*relation_level*		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Konkret	Versuch eine vollständige Liste von Handlungsschritten aufzuschreiben. *Fehlendes Konzept um mit variabler Länge umzugehen.* – *Vergleiche Buchstabe 1 mit Buchstabe n* *Vergleiche Buchstabe 2 mit Buchstabe n-1* – *Das Wort wird halbiert. Die zweite Hälfte wird umgedreht. Nun kann Stelle 1 bis 5 der jeweiligen Hälften verglichen werden.*	0
Spezifisch	Unvollständige Parametrisierung *Zurückgreifen auf konkrete Zahlen.* *n=0; Vergleiche n+1 mit 10-n; erhöhe n um 1 und beginne wieder mit dem Vergleich, solange bis n=Länge/2*	1
Generisch	Handlungsbeschreibung, welche Parameter für die Position sowie die Länge einführt. – *n=1; Vergleiche n mit Länge-n+1; erhöhe n um 1 und beginne wieder mit dem Vergleich, solange bis n=Länge* – *n=0; Vergleiche Buchstabe n+1 mit Buchstabe Länge-n; erhöhe n um 1; vergleiche erneut, solange bis n=Länge-1* – *for(int i=0; i<wort.length; i++)* *Vergleiche wort(i) mit wort(length-1-i)* – *n=1; Vergleiche n mit Länge-n+1; erhöhe n um 1 und beginne wieder mit dem Vergleich, solange bis n=Länge/2 (abgerundet)*	2
Darstellungsform 10 − 2_*relation_type*		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Text	*Vergleiche Buchstabe n+1 mit Buchstabe Länge-n;* *erhöhe n um 1; vergleiche erneut, solange bis n den Wert Länge/2 hat*	0
Diagramm		1
Programm-Code	*for(int i=0; i<wort.length; i++)* *Vergleiche wort(i) mit wort(length-1-i)*	2

Smartphone (Smartphone)

Seite 19

Hinweis: Um eine Antwort einzuschätzen, hilft die Überlegung einer Vererbungshierarchie.
Ein (1) ist ein (2) und das ist ein (3)

Beispiel von Abstraktionsgrad niedrig → hoch sortiert.

Smartphone auf 3 Ebenen		Code
Korrektheit 11 − 1_*correct*		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv*	99
Falsch	Eine Antwort ist falsch, wenn * sich der beschriebene Prozess oder das beschriebene Objekt ändert oder wenn eine Ereignisfolge beschrieben wurde. * nicht das Smartphone selbst abstrahiert wird, sondern z.B. das Besitzverhältnis oder eine seiner Funktionen. * die angegebene Reihenfolge fachlich falsch ist. – *Mensch tippt auf App Button → Software arbeitet im Hintergrund → Kondensatoren, speichern, löschen usw.* – *Samsung Model xx mit LED (oder so) Bildschirm. → Ein Android-Smartphone mit einem xx-Zoll Display. → Ein Smartphone mit großem Bildschirm.* – *Mein Smartphone ist ein schwarzes Samsung Galaxy S5 mit einer FMA Plastikhülle → Mein Smartphone ist ein Samsung Galaxy S5. → Mein Smartphone ist rechteckig und hat einen Touch-Screen.* – *Mein Smartphone ist das Modell Huawei P8 lite und ich besitze es seit einem Jahr und 10 Monaten. → Mein Smartphone ist von der Marke Huawei und ich besitze es seit ungefähr 2 Jahren. → Ich besitze mein Smartphone schon recht lange.* – *Senden von Text, Bild, Video und anderem → Senden von Nachrichten → Übertragung von Binärcodes* – *Gutes Design → Gute Hardware → Schnelle Software* – *Ein Foto von Freunden machen → Mit Freunden telefonieren → Auf Google Play eine App herunterladen* – *Ein mobiles Gerät welches mit bildlicher Darstellung und dem reagieren auf Drücken funktioniert und zum Telefonieren sowie verschicken von Nachrichten verwendet werden kann. → Durch Drücken auf Bildschirm werden Reaktionen hervorgerufen und bildlich dargestellt. → Aktion auf Fläche -> visuelle Reaktion*	0

Smartphone auf 3 Ebenen		Code
Richtig	Eine Antwort ist richtig, wenn der beschriebene Prozess oder das Objekt dasselbe bleibt. Die Nutzung des Wortes Smartphone in der Antwort ist zulässig. – *iPhone 6 Plus mit 64 GB Speicher in Space Grau → iPhone mit mehr als 32GB Speicher → Irgendein Smartphone* – *Smartphone der Marke Samsung. Teil der S-Reihe in der 7. Generation. Neuere Edge-Ausführung mit xGB RAM und 16GB Speicherplatz. Mit hochauflösender Kamera. → Samsung S7 Edge mit hochauflösender Kamera und 16 GB Speicherplatz. → Samsung S7 Edge.* – *schwarzes rechteckiges Objekt, welches mittels Tasten ein- und ausgeschaltet werden kann, mittels Touch bedient und Töne abspielen kann [...] → schwarzes rechteckiges Objekt mit Tasten und Oberflächen, welche bestimmte Funktionen haben. → schwarzes rechteckiges Objekt* – *Multi-Mediales mobiles Kommunikationsgerät → Taschencomputer → Rechenmaschine - Verarbeitung von 0 und 1* – *iPhone → Multifunktionsgerät zum Telefonieren → Telefon* – *iPhone 10 → Smartphone → Elektronisches Gerät* – *Ein Gerät, dass mit Touch-Eingabe bedient wird und auf denen sog. Apps installiert werden, um verschiedene Funktionen auszuführen. → Elektronisches Gerät, dass so aufgebaut ist, dass sie verschiedenste Operationen ausführen, Eingaben verstehen und Lösungen ausgeben können. → Mikrocontroller, der seinen Input verarbeitet und per Interface sein Output übermittelt.*	1

Begründung		Code
Korrektheit $11 - 2\_correct$		
Leer	Leeres Antwortfeld oder Antwort vollständig durchgestrichen	999
Ohne Bezug	Antwort ohne Bezug zur Frage oder Schlagwort *Weiß ich nicht, Intuitiv* – *Weil sie das halt macht.* – *weils mir so am besten eingefallen ist und für einen Nutzer ist die Funktionsweise abstrakter als das Drücken eines App-Buttons.*	99
Falsch	Unvollständig oder keine übergeordnete Beschreibung der Reihenfolge, Wiedergabe der Beispiele. – *Weil mit Mobiltelefon bsp. jedes Tasten- oder Touchscreen gemeint sein kann.* – *Der Touchscreen ist leicht zu benutzen, die Kamera auch aber ist kompliziert in normaler Einstellungen. Das Smartphone ist schwer zu benutzen.* – *Einmal die konkrete Darstellung meines Handys. Einmal das Runterbrechen des Handys auf Grundelemente. Zuletzt das Grundlegende Konzept von Handy ohne das sofort erkennbar ist um was es sich handelt.*	0
Richtig	Nachvollziehbare Begründung, warum die Beschreibungen 3 verschiedene Abstraktionsstufen darstellen. – *Da es unterschiedlich konkret beschrieben wurde und somit verschieden viel Interpretationsraum lässt.* – *Es wird immer allgemeiner beschrieben und somit schwerer genauer zu identifizieren.* – *Beschreibung wird immer genauer und zutreffender [...]* – *Ich habe immer weniger Details über mein Smartphone aufgeschrieben.* – *Die erste Beschreibung legt dar, was man mit dem Gerät genau machen kann, die zweite fasst die meisten Funktionen zusammen, die dritte ist lediglich der allgemein gebräuchliche Name für alle Geräte dieser Art.*	1

Bibliography

[1] Philip Ackermann and Leo Leowald. *Schrödinger programmiert Java: das etwas andere Fachbuch*. Galileo Press, 2014.

[2] Philip Adey and Michael Shayer. *Really raising standards: Cognitive intervention and academic achievement*. Routledge, 2006.

[3] Alfred V. Aho and Jeffrey D. Ullman. *Foundations of computer science*. Computer Science Press, Inc., 1992.

[4] Amodern.net. London underground 1928. `http://amodern.net/wp-content/uploads/2013/08/3_tubemap.png`, January 2017.

[5] Lorin W. Anderson, David R. Krathwohl, and Benjamin Samuel Bloom. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Allyn & Bacon, 2001.

[6] Roy Andersson and Lars Bendix. Pair teaching–an extreme teaching practice. *4: e Pedagogiska Inspirationskonferensen*, 2006.

[7] Rüdeger Baumann. *Didaktik der Informatik*. Klett Stuttgart, 1996.

[8] Thomas Bedürftig and Roman Murawski. *Philosophie der Mathematik*. Walter de Gruyter GmbH & Co KG, 2015.

[9] Jens Bennedsen and Michael E. Caspersen. Abstraction ability as an indicator of success for learning object-oriented programming? *SIGCSE Bull.*, 38(2):39–43, June 2006.

[10] John Martin Bland and Douglas G. Altman. Statistics notes: Cronbach's alpha. *The BMJ*, 314(7080):572, 1997.

[11] Andrea Bohn, Gudula Kreykenbohm, Marion Moser, and Anna Pomikalko. *Modularisierung in Hochschulen. Handreichung zur Modularisierung und Einführung von Bachelor-und Master-Studiengängen. Erste Erfahrungen und Empfehlungen aus dem BLK-Programm" Modularisierung"*. Bonn: BLK, 2002.

[12] Jürgen Bortz and Nicola Döring. *Forschungsmethoden und Evaluation*. Springer, 2016.

[13] Axel Böttcher, Kathrin Schlierkamp, Veronika Thurner, and Daniela Zehetmeier. Teaching abstraction. In *International Conference on Higher Education Advances (HEAd)*, 2016.

[14] Axel Böttcher, Matthias C. Utesch, and Austin Moore. Erfahrungen mit pair teaching für software engineering: Kooperation von hochschule und industrie. In *SEUH*, pages 5–15, 2009.

[15] Paolo Bucci, Timothy J. Long, and Bruce W. Weide. Do we really teach abstraction? In *ACM SIGCSE Bulletin*, volume 33, pages 26–30. ACM, 2001.

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

[17] Markus Bühner. *Einführung in die Test- und Fragebogenkonstruktion*. Pearson Deutschland GmbH, 2011.

[18] BWINF, Gesellschaft für Informatik e.V. Die "Bundesweiten Informatikwettbewerbe". `http://informatik-biber.de/`.

[19] Ibrahim Cetin and Ed Dubinsky. Reflective abstraction in computational thinking. *The Journal of Mathematical Behavior*, 47:70–80, 2017.

[20] Timothy Colburn and Gary Shute. Abstraction in computer science. *Minds and Machines*, 17(2):169–184, 2007.

[21] Student Success Collaborative. The murky middle project, 2014.

[22] Allan Collins, John Seely Brown, and Susan E Newman. Cognitive apprenticeship. *Thinking: The Journal of Philosophy for Children*, 8(1):2–10, 1988.

[23] Charles T. Cook, Svetlana Drachova, Jason O. Hallstrom, Joseph E. Hollingsworth, David P. Jacobs, Joan Krone, and Murali Sitaraman. A systematic approach to teaching abstraction and mathematical modeling. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, pages 357–362. ACM, 2012.

[24] Valentina Dagienė and Gerald Futschek. Bebras international contest on informatics and computer literacy: Criteria for good tasks. In *Proceedings of the 3rd International Conference on Informatics in Secondary Schools - Evolution and Perspectives: Informatics Education - Supporting Computational Thinking*, ISSEP '08, pages 19–30, Berlin, Heidelberg, 2008. Springer-Verlag.

[25] Don Davis, Timothy Yuen, and Matthew 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, pages 325–330, New York, NY, USA, 2014. ACM.

[26] Philip J. Davis and Reuben Hersh. *Erfahrung Mathematik*. Springer-Verlag, 2013.

[27] Vasily Vasilovich Davydov. Soviet studies in mathematics education volume 2., 1990.

[28] Keith Devlin. Special issue: Why universities require computer science students to take math - introduction. *Communications of the ACM*, 46(9), 2003.

[29] Christina Dörge. *Informatische Schlüsselkompetenzen: Konzepte der Informationstechnologie im Sinne einer informatischen Allgemeinbildung*. Universitätsverlag Potsdam, 2012.

[30] John Erpenbeck and Lutz von Rosenstiel, editors. *Handbuch Kompetenzmessung – Erkennen, verstehen und bewerten von Kompetenzen in der betrieblichen pädagogischen und psychologischen Praxis*. Schäffer-Poeschel Verlag Stuttgart, 2007.

[31] Joseph L. Fleiss and Jacob Cohen. The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability. *Educational and psychological measurement*, 33(3):613–619, 1973.

[32] Geocities Web Archive Project - Prof. Leonard H. Soicher. London underground 1931. `http://www.geocities.ws/lhsoicher/images/1933a.jpg`, January 2017.

[33] Gesellschaft für Informatik (GI). Empfehlungen für Bachelor- und Masterprogramme im Studienfach Informatik an Hochschulen, 2016.

[34] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of software engineering*. Prentice Hall PTR, 2002.

[35] Norman E. Gronlund. *Assessment of student achievement*. ERIC, 1998.

[36] Mark Guzdial and Elliot Soloway. Teaching the nintendo generation to program. *Communications of the ACM*, 45(4):17–21, 2002.

[37] Michael Häder. *Empirische Sozialforschung: Eine Einführung*. Springer, 2015.

[38] Sabine Hammer, Daniela Zehetmeier, Axel Böttcher, and Veronika Thurner. Evaluation of a diagnostic test for cognitive competences that are relevant for computer science. In *2018 IEEE Global Engineering Education Conference (EDUCON)*, pages 545–554, April 2018.

[39] Johannes Hartig and Eckhard Klieme. *Kompetenz und Kompetenzdiagnostik*, chapter Kompetenz und Kompetenzdiagnostik, pages 127–143. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[40] Johannes Hartig and Eckhard Klieme. *Möglichkeiten und Voraussetzungen technologiebasierter Kompetenzdiagnostik: Eine Expertise im Auftrag des Bundesministeriums für Bildung und Forschung*. BMBF, Referat Öffentlichkeitsarbeit, 2007.

[41] Werner Hartmann, Michael Näf, and Raimond Reichert. *Informatikunterricht planen und durchführen*. Springer-Verlag, 2007.

[42] Michael Haungs, Christopher Clark, John Clements, and David Janzen. Improving first-year success and retention through interest-based CS0 courses. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 589–594. ACM, 2012.

[43] Orit Hazzan and Jeff Kramer. Abstraction in computer science & software engineering: A pedagogical perspective. *Frontier Journal*, 4(1):6–14, 2007.

[44] Orit Hazzan and Jeff Kramer. Assessing abstraction skills. *Commun. ACM*, 59(12):43–45, December 2016.

[45] Rina Hershkowitz, Baruch B. Schwarz, and Tommy Dreyfus. Abstraction in context: Epistemic actions. *Journal for Research in Mathematics Education*, pages 195–222, 2001.

[46] Peter Hubwieser. *Didaktik der Informatik Grundlagen, Konzepte, Beispiele*. Springer, 2007.

[47] Ludger Humbert. *Didaktik der Informatik - mit praxiserprobtem Unterrichtsmaterial*. Teubner, 2006.

[48] Ludger Humbert. Informatische Bildung: Fehlvorstellungen und Standards. In *MWS – Münsteraner Workshop zur Schulinformatik 2006*, pages 37–46, 2006.

[49] Bärbel Inhelder and Jean Piaget. An essay on the construction of formal operational structures. the growth of logical thinking: From childhood to adolescence (a. parsons & s. milgram, trans.), 1958.

[50] Päivi Kinnunen and Lauri Malmi. Why students drop out CS1 course? In *Proceedings of the second international workshop on Computing education research*, pages 97–108. ACM, 2006.

[51] Eckhard Klieme, Johannes Hartig, and Dominique Rauch. The concept of competence in educational contexts. *Assessment of competencies in educational contexts*, pages 3–22, 2008.

[52] Karoline Koeppen, Johannes Hartig, Eckhard Klieme, and Detlev Leutner. Current issues in competence modeling and assessment. *Zeitschrift für Psychologie/Journal of Psychology*, 216(2):61–73, 2008.

[53] Jeff Kramer. Is abstraction the key to computing? *Communications of the ACM*, 50(4):36–42, 2007.

[54] Jeff Kramer and Orit Hazzan. Questions for an abstraction test. `https://docs.google.com/forms/d/e/1FAIpQLSc3KjvqkUfdLY5w2Q_lAdKDfQ-c0Q6J9M4pYMeso2l0xWT3tA/viewform`.

[55] Barry L. Kurtz. Investigating the relationship between the development of abstract reasoning and performance in an introductory programming class. In *ACM SIGCSE Bulletin*, volume 12, pages 110–117. ACM, 1980.

[56] J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.

[57] David K. Lewis. On the plurality of worlds. vol. 322, 1986.

[58] Barbara Liskov and John Guttag. *Program development in Java: Abstraction and specification in program development*, volume 180. MIT press Cambridge, 2002.

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

[60] Paul Lorenzen. *Lehrbuch der konstruktiven Wissenschaftstheorie*. Springer, 1987.

[61] Sana Loue and Martha Sajatovic. *Encyclopedia of aging and public health*. Springer Science & Business Media, 2008.

[62] Merriam-Webster. Dictionary. `http://www.merriam-webster.com/`, 2017.

[63] Bertrand Meyer. Towards an object-oriented curriculum. In *TOOLS (11)*, pages 585–594. Citeseer, 1993.

[64] Ministère de l'Éducation nationale, de l'Enseignement supérieur et de la Recherche. European higher education area and bologna process. `http://www.ehea.info`, March 2017.

[65] Faron Moller and Georg Struth. *Modelling Computing Systems: Mathematics for Computer Science*. Springer Science & Business Media, 2013.

[66] Dung Nguyen and Stephen 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.

[67] Stellan Ohlsson and Erno Lehtinen. Abstraction and the acquisition of complex ideas. *International Journal of Educational Research*, 27(1):37–48, 1997.

[68] Rachel Or-Bach and Ilana Lavy. Cognitive activities of abstraction in object orientation: An empirical study. *SIGCSE Bull.*, 36(2):82–86, June 2004.

[69] David Pace and Joan K. Middendorf. *Decoding the disciplines: Helping students learn disciplinary ways of thinking*. Jossey-Bass, 2004.

[70] Miranda C. Parker, Mark Guzdial, and Shelly Engleman. Replication, validation, and use of a language independent CS1 knowledge assessment. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ICER '16, pages 93–101, New York, NY, USA, 2016. ACM.

[71] Roy D. Pea. Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research*, 2:25–36, 1986.

[72] Franz Petermann and Monika Daseking. *Diagnostische Erhebungsverfahren*. Hogrefe Verlag, 2015.

[73] Oxford University Press. Oxford English Dictionary. `http://www.oed.com/`.

[74] Jürgen Rost. *Lehrbuch Testtheorie-Testkonstruktion (2., vollständig überarbeitete und erweiterte Aufl.)*. 2004.

[75] Nathan Rountree, Janet Rountree, and Anthony Robins. Predictors of success and failure in a cs1 course. *ACM SIGCSE Bulletin*, 34(4):121–124, 2002.

[76] Niclas Schaper, Oliver Reis, Johannes Wildt, Eva Horvath, and Elena Bender. Fachgutachten zur kompetenz-orientierung in studium und lehre. 2012.

[77] Neal Schmitt. Uses and abuses of coefficient alpha. *Psychological assessment*, 8(4):350, 1996.

[78] Carsten Schulte and Jens Bennedsen. What do teachers teach in introductory programming? In *Proceedings of the second international workshop on Computing education research*, pages 17–28. ACM, 2006.

[79] Yvonne Sedelmaier, Sascha Claren, and Dieter Landes. Welche Kompetenzen benötigt ein Software Ingenieur? In *SEUH*, pages 117–128, 2013.

[80] Norbert M. Seel, editor. *Encyclopedia of the Sciences of Learning*. Springer Science & Business Media, 2011. States open questions.

[81] Richard J. Shavelson. On the measurement of competency. *Empirical research in vocational education and training*, 2(1):41–63, 2010.

[82] IEEE Computer Society, Pierre Bourque, and Richard E. Fairley. *Guide to the Software Engineering Body of Knowledge (SWEBOK®): Version 3.0*. IEEE Computer Society Press, Los Alamitos, CA, USA, 3rd edition, 2014.

[83] Juha Sorva. The same but different - students' understandings of primitive and object variables. In *Proceedings of the 8th International Conference on Computing Education Research*, Koli '08, pages 5–15, New York, NY, USA, 2008. ACM.

[84] Peter Sprague and Celia Schahczenski. Abstraction the key to cs1. *Journal of Computing Sciences in Colleges*, 17(3):211–218, 2002.

[85] Mohsen Tavakol and Reg Dennick. Making sense of cronbach's alpha. *International journal of medical education*, 2:53, 2011.

[86] Allison Elliott Tew and Mark Guzdial. Developing a validated assessment of fundamental CS1 concepts. In *Proceedings of the 41st ACM technical symposium on Computer science education*, SIGCSE '10, pages 97–101, New York, NY, USA, 2010. ACM.

[87] Allison Elliott Tew and Mark Guzdial. The FCS1: A language independent assessment of CS1 knowledge. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 111–116, New York, NY, USA, 2011. ACM.

[88] The College Board. AP computer science principles. `https://secure-media.collegeboard.org/digitalServices/pdf/ap/ap-computer-science-principles-course-and-exam-description.pdf`, May 2017.

[89] The College Board. AP students. `https://apstudent.collegeboard.org`, 2018.

[90] The Joint Task Force on Computing Curricula Association for Computing Machinery (ACM) IEEE Computer Society. Computer science curricula 2013. `https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf`, 2012.

[91] The Joint Task Force on Computing Curricula IEEE Computer Society Association for Computing Machinery. Computer science curricula 2001. `https://www.acm.org/binaries/content/assets/education/curricula-recommendations/cc2001.pdf`, 2001.

[92] Veronika Thurner and Axel Böttcher. An "objects first, tests second" approach for software engineering education. In *2015 IEEE Frontiers in Education Conference (FIE)*, pages 1–5. IEEE, 2015.

[93] Veronika Thurner, Axel Böttcher, and Andreas Kämper. Identifying base competencies as prerequisites for software engineering education. In *Global Engineering Education Conference (EDUCON), 2014 IEEE*, pages 1069–1076, April 2014.

[94] Philip R. Ventura. Identifying predictors of success for an objects-first CS1. 2005.

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

[96] Nicole Weicker, Botond Draskoczy, and Karsten Weicker. Fachintegrierte Vermittlung von Schlüsselkompetenzen der Informatik. In *HDI*, pages 51–62, 2006.

[97] Julius Weinberg. Abstraction in the formation of concepts. In Philip P. Wiener, editor, *Dictionary of the History of Ideas: Studies of Selected Pivotal Ideas*. Charles Scribner, 1973-1974.

[98] Franz E. Weinert. Concept of competence: A conceptual clarification. In *Defining and selecting key competencies*, pages 45–65. Hogrefe & Huber Publishers, 2001.

[99] Jeannette M. Wing. Computational thinking. *Commun. ACM*, 49(3):33–35, March 2006.

[100] Daniela Zehetmeier, Axel Böttcher, and Anne Brüggemann-Klein. Designing lectures as a team and teaching in pairs. In *4th International Conference on Higher Education Advances (HEAD'18)*, pages 873–880. Editorial Universitat Politècnica de València, 2018.

[101] Daniela Zehetmeier, Axel Böttcher, Anne Brüggemann-Klein, and Veronika Thurner. Defining the competence of abstract thinking and evaluating CS-students' level of abstraction. In *Conference on Software Engineering Education and Training (CSEE&T)*, 2019.

[102] Daniela Zehetmeier, Axel Böttcher, and Veronika Thurner. Design and evaluation of a test for assessing CS-first-year students' cognitive competences. In *Global Engineering Education Conference (EDUCON), 2017 IEEE*, April 2017.

[103] Daniela Zehetmeier, Axel Böttcher, and Veronika Thurner. Differenzierte Übungsblätter – Für Experten und solche die es werden wollen. In *MINT-Symposium, Nürnberg, Germany*, pages 94–98, 2017.

[104] Daniela Zehetmeier, Marco Kuhrmann, Axel Böttcher, Kathrin Schlierkamp, and Veronika Thurner. Self-assessment of freshmen students' base competencies. In *Global Engineering Education Conference (EDUCON), 2014 IEEE*, pages 429–438, April 2014.