Empirical Definition of Object-oriented Programming Competencies

Combining Automated Assessment and Item Response Theory to Define Competencies for Implementing Abstract Data Types

PETER HUBWIESER, Technical University of Munich, Germany
JOHANNES KRUGEL, Leibniz University Hannover, Germany
MIKE TALBOT, Technical University of Munich, Germany
MICHAEL STRIEWE, University of Duisburg-Essen, Germany
MICHAEL GOEDICKE, University of Duisburg-Essen, Germany
CHRISTOPH OLBRICHT, University of Duisburg-Essen, Germany

International large-scale educational investigations and the focus on learners' competencies powered a veritable revolution in teaching and learning approaches as well as in educational research methodologies. In the relatively young field of computer science education research, however, there is a considerable lack of empirical studies on the definition and measurement of competencies. The central goal of the presented research project is to identify, describe, and measure competencies for object-oriented programming, in particular for implementing abstract data types.

We use an automated assessment system to evaluate and score a large number of students' solutions of programming tasks. Item Response Theory analyses of the results identify subsets of tasks suitable for defining typical programming competencies. Further qualitative analyses reveal the internal structure of the competencies and allow a classification in a competency structure model. This article presents in detail our rigorous methodology and exemplary results for the empirical definition and decomposition of the competency named "Ability to implement the abstract data type *Binary Search Tree*".

CCS Concepts: • **Applied computing** → *E-learning*; *Interactive learning environments*; • **Social and professional topics** → **Student assessment**; **CS1**.

Additional Key Words and Phrases: programming education, competency definition, competency measurement, automated assessment, CS1

1 INTRODUCTION

At the beginning of the new century, several international large scale investigations on learning results like the PISA studies of the OECD had revealed that many learning processes, in particular in schools, had produced mostly "Inert Knowledge", which students could express, but not apply in the "real-world". Aiming to describe and relieve these deficits, supported by constructivist learning approaches from late 1980s, the concept of "competence" (or "competency", see Section 2.2) powered a veritable revolution in teaching and learning approaches as well as in the methodologies of educational research [61]. Now, the primary goal of teaching was that students learn to utilize their knowledge to solve complex problems in "real-world" like situations.

In many countries, this development was supported in the first decade of the new century by large public funding programs, which enabled numerous projects that developed sophisticated competency frameworks and measured learning results extensively for many school subjects, e.g.

Authors' addresses: Peter Hubwieser, peter.hubwieser@tum.de, Technical University of Munich, Arcisstr. 21, Munich, Germany, 80333; Johannes Krugel, krugel@dei.uni-hannover.de, Leibniz University Hannover, Appelstr. 9a, Hannover, Germany; Mike Talbot, mike.talbot@tum.de, Technical University of Munich, Arcisstr. 21, Munich, Germany; Michael Striewe, michael.striewe@s3.uni-due.de, University of Duisburg-Essen, Gerlingstr. 16, Essen, Germany; Michael Goedicke, michael.goedicke@s3.uni-due.de, University of Duisburg-Essen, Gerlingstr. 16, Essen, Germany; Christoph Olbricht, christoph.olbricht@s3.uni-due.de, University of Duisburg-Essen, Gerlingstr. 16, Essen, Germany.

mathematics [42]. Unfortunately, as computer science (CS) was not a regular school subject in most countries at that time, competency research in CS was left behind. In consequence, there is still a considerable lack of empirical studies on the definition and measurement of computer science competencies up to now. There is a need for research not only on the definition and measurement of competencies, but also on their internal structure and their interrelationships, especially on competency structure models. Unfortunately, empirical research on competencies using current state-of-the-art methods based on *Item Response Theory* (IRT, see Section 2.3), requires large case numbers in the range of several hundred test subjects.

Based on several years of preparatory collaboration, our two research groups have launched the project AKoFOOP, which aims for the evidence-based definition and measurement of competencies in the field of object oriented programming (OOP). For this purpose we have combined three research fields:

(1) the automated analysis and evaluation of program code,
(2) the evidence-based definition and measurement of competencies, and
(3) the generation of helpful feedback based on such competency definitions.

Each of the two research groups involved brings specific expertise and experience to the project. The Computer Science Education Research (CSER) group at Technical University of Munich had been researching CS skills with large case numbers for several years and developed a MOOC for the introduction of object-oriented programming [28, 29]. The group of Software Technology researchers (STR) at University of Duisburg-Essen had been conducting a large CS1 lecture for several years and developed the Automated Assessment System (AAS) JACK [15], which is used to assess the students' submissions to programming assignments from a *Computer Science 1 (CS1)* lecture, comprising five so called *mini-projects* (MPs). The central topics of these MPs corresponded with the first subdimension of the competency model for object oriented programming (OOP) proposed by Kramer et al. [27], see Table 1: *Array* (MP1), *Linked List* (MP2), *Binary (Search) Tree* (MP3), *Inheritance* (MP4), and *Database Implementation* (MP5).

Combining research on competencies with the use of automated assessment systems seems to be particularly useful. The primary goal of most automated assessment systems [21, 55] is to generate feedback for students that allows them to improve their solutions [19, 40, 59]. A high granularity of automated assessment is usually achieved by a correspondingly elaborate specification of the desired properties of a program in the form of test rules, test cases, sample solutions, and similar artifacts. As far as it does not concern trivial errors, the analysis procedures give however only in rare cases information about the actual line of code within the solution that is possibly wrong [16, 56]. Moreover, even in simple cases they do not provide explicit information at all about what the conceptual causes of errors are in terms of missing competencies.

Essentially, AKoFOOP is based on the idea of using JACK to very quickly evaluate and score a large number of students' solutions to programming tasks syntactically and semantically. An IRT analysis of the results is used to identify subsets of tasks that are suitable for defining typical programming competencies. Further qualitative analyses of the student solutions identify components of these competencies that allow a classification in a suitable competency structure model. In addition, interviews with students were conducted and all solutions are analyzed for deficits, knowledge gaps, and misconceptions. With the knowledge gained in this way about the programming competencies, the respective task sets are then in turn to be further developed in such a way that they are suitable as tests in the sense of IRT. In the end, the automated feedback of JACK is to be adjusted to deficits in the acquired competencies in order to really help students learn. The project is funded by the German Research Foundation (DFG). So far, we have analyzed a large part of the data that was collected during the lectures delivered in four winter terms, attended by more than 11,000 students

and 48,000 solutions submitted to the AAS. Due to the mass of data and intermediate results, we will restrict the presentation in this paper to an exemplary mini-project that was designed to learn the implementation of binary trees. For this project, we will present the entire analysis process of our project, starting at the qualitative data analysis of assignments and expert solutions, proceeding by investigating the item demands of the assignments, the statistical evaluation of students' responses on the tasks and finally the definition and decomposition of the resulting competency named "Ability to implement the abstract data type *Binary Search Tree*". To the best of the authors knowledge, this is the first time that a competency in the area of OOP has been defined by rigorous empirical methods. This paper thus deliberately not only contributes the resulting competency definition, but also a detailed description of the methodology.

2 BACKGROUND

2.1 Data Types and Data Structures

As mentioned in the introduction, (abstract) data types and data structures play a central role in our project. The software crisis of the mid-1970s forced computer scientists to work on the quality of their programs. As part of this effort, a number of very useful concepts and techniques emerged, including data type abstraction. A comprehensive overview of abstract data types (ADTs) with a lot of subtypes and variants was compiled by Jürgen Uhl and Hans Albrecht Schmid [58], who categorized all relevant operations on the covered ADTs (pp. 54ff): *Constructors / Operations Based on Indices / Access by Position Count / Selector Operations / Iterators / Reduction / Find, Skip and Count / Existential and Universal Quantifiers / Order Dependent Operations / Hash Operation*. Today, the MIT maintains a much simpler categorization in reading 8 of the lecture 6.005 on Software Construction[1]; the operations on ADTs are classified as *Creators*, *Producers*, *Observers*, and *Mutators*.

For our project the ADTs *Array*, *Linked List* and *Binary Tree* are of particular importance due to the focus of the respective mini-projects. Since we want to limit the presentation of our research activities in this article to an exemplary mini-project on binray trees, this data structure will be briefly defined here, see [8, p. 1088f]:

> A binary tree T is a structure defined on a finite set of nodes that either contains no nodes, or is composed of three disjoint sets of nodes: a root node, a binary tree called its left subtree, and a binary tree called its right subtree.

With the help of binary search trees, one can realize very efficient searches, see [52, p. 396]:

> A binary search tree (BST) is a binary tree where each node has a comparable key (and an associated value) and satisfies the restriction that the key in any node is larger than the keys in all nodes in that node's left subtree and smaller than the keys in all nodes in that node's right subtree.

2.2 Competencies and Competency Models

The concept of competence (or competency) plays an important role in current educational research. Unfortunately, however, the term is interpreted in very different ways, see, e.g. [23]. There is not even a clear distinction between the terms "competence" and "competency". In this article we will interpret a *competency* as a context-specific cognitive performance disposition as defined by Weinert, originally in German [61, p. 27], translated by Klieme et al. [22, , p. 65] as follows:

> We define competencies as the cognitive abilities and skills possessed by or able to be learned by individuals that enable them to solve particular problems, as well as

[1] https://web.mit.edu/6.005/www/fa14/classes/08-abstract-data-types/

> the motivational, volitional and social readiness and capacity to use the solutions successfully and responsibly in variable situations".

In many cases, competencies are defined by detailed descriptions of intended behavior based on extensive empirical research, e.g., in the form of educational standards [22] or in voluminous subject-specific frameworks as in the PISA surveys [42]. In order to make such definitions of competencies accessible to measurement, for the test items it must then be demonstrated through extensive validation that they measure precisely these competencies, see [30], [53].

Already for the measurement of one single competency a whole set of items is necessary. According to Klieme et al. [22, p. 66],

> Competencies cannot be reflected by or assessed in terms of a single, isolated performance. Rather, the range of situations in which a specific competence takes effect always spans a certain spectrum of performance. Narrow assessments cannot meet the requirements of competency models. [..] Competence must be assessed by an array of tasks and tests that do more than simply tap factual knowledge.

Obviously, it has to be assured by the validation that the solution of this "array of tasks" requires predominantly certain levels of precisely the competency that should be measured. Since such validations are time-consuming and cost-intensive, some researchers take a different approach. According to Schott & Azizi Ghanbari [51, p. 15], competencies can be defined straightforward as the ability to solve a set of certain tasks:

> A competence consists of certain amounts of tasks that can be carried out if you have the competence.

In consequence, we can consider certain sets of "suitable" tasks as representations of a specific competency. However, there are certain preconditions for this, in particular a sufficient psychometric homogeneity of this set of tasks, which means that for each set the probability that an individual is able to solve these tasks depends essentially only on the level of the respective competency that is defined by this set. This requirement is very similar to the criterion of internal consistency of classical tests, which can be calculated by the Cronbach's alpha coefficient [9]. Alpha will be negative whenever there is greater within-subject variability than between-subject variability. The common rule of thumb for Internal Consistency is "excellent" for $\alpha \geq 0.9$, "good" for $\alpha \geq 0.8$ and "acceptable" for $\alpha \geq 0.7$.

Competence models are indispensable for structuring competencies. Klieme et al. [23, pp 10f] distinguished three types: models of *competence levels*, *competence structures*, and *competence development*. For our purpose, competency structure models are the type we need for the investigations on programming skills. Unfortunately, very few solid research results on empirically founded competencies or models for computer science can be found at the international level so far. For our project, competence structure models and competence level models in the sense of Klieme & Leutner [24, pp. 6f] are of particular importance. Some research in this direction has been carried out within the framework of the MoKoM project, which has resulted in a first empirically and theoretically based competence structure model for two sub-areas of computer science (modelling and system comprehension) as well as a test instrument for them. It turned out, however, that the dimensionality of the measured competencies was not sufficiently clear [38], [39].

Based on the same competency definition, other researchers investigated the potential cognitive facets of OOP by analyzing the structures of declarative knowledge by comparing concept maps of first-year students with very different computer science background [37]. Incorporating these findings, Kramer et al. developed a competency model for OOP by combining extensive literature studies, also on other school subjects, and additional qualitative empirical research [27]. For our

Table 1. Competency Structure Model of OOP, adapted from [27]

(Sub-)Dimension	Exemplary Manifestations
1. CS knowledge and skills [37]	
1.1. Data structure	Primitive data types, array, list, tree, graph
1.2. Class & object structure	Classes, objects, associations, inheritance, polymorphism, abstract and generic concepts
1.3. Algorithmic structure	Sequences, loops, conditionals, nesting, recursion, concurrency
1.4. Notional machine [4, 54]	Acting robots or other objects, Register or Turing Machine, "Real" Computer
2. Representation language [50]	
2.1. Semantic Paradigm	OO, Declarative, Imperative, Functional
2.2. Modality	Enactive – Iconic – Symbolic [7], Textual – Visual
2.3. Level of Formalisation	Mathematical representation – Programming language – Pseudocode – Everyday language
3. Cognitive operations	
3.1. CS problem solving stage [49, 62]	Analysis: understanding the problem – Design: determine how to solve the problem – Coding: translating into a computer language program – Testing and debugging the program
3.2. PISA problem solving stage [41]	Exploring and understanding – Representing and formulating – Planning and executing – Monitoring and reflecting
3.3. Cognitive process category [12]	Interpreting (Remember, Understand, Analyze, Evaluate) – Producing (Apply, Create)

project, we have extended and adapted this model based on the preliminary research results as displayed in Table 1.

2.3 Measurement of Competencies

Competency models and procedures to measure competencies have been developed in particular for the scientific foundation of the OECD PISA studies. The theoretical and methodological foundations can be found in [18], among others.

A typical use case might look like this. Aiming to measure a psychometric construct (latent trait), a number of persons are exposed to a test sheet and asked to respond to its items. This results in a response matrix, in which each response of a person corresponds to a row and each item in the test sheet to a column. In the case of dichotomous scale levels, all cells contain either 1 or 0 (apart from missing answers). To measure a competency, one basically needs a whole set of tasks, as stated by Klieme et al. [22, p. 66]. These tasks have to be constructed and validated very carefully to ensure that the ability to solve them actually depends primarily on the individual expression of the intended competency. However, if a competency is defined directly by a set of tasks in the sense of [51, p. 15] (see Section 2.2), then these tasks can obviously also be used directly for measurement. According to Weinert's definition in Section 2.2, competencies are by their nature quite complex. For a precise analysis of competencies, one must therefore dissect both competencies and tasks in order to gain insights into their internal structure. First, the tasks are split in "atomic" subtasks, which are representing the items in this case. Second, the specific cognitive requirements of each item are identified as *item demands*. By evaluating a set of solutions to these items in terms of their

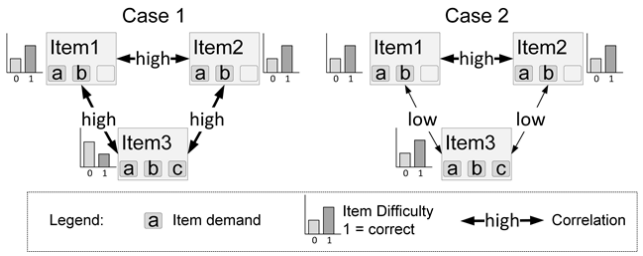


Fig. 1. Item demands in different cases [17]

item demands, one can then draw conclusions about the cognitive structure (i.e., the "components") of the particular competency being measured, see [17].

However, a given set of items can only contribute to these insights if the items differ either in their difficulty (case 1) or in their mutual correlation over all students' solutions (case 2). Figure 1 illustrates these two cases in more detail for a set of three items, where items 1 and 2 require the same cognitive abilities (item demands) a and b, while item 3 requires additionally item demand c. In case 1 all three items show a high mutual correlation, but item 3 is clearly more difficult. In this case, the requirements of all three items seem to be unidimensional (homogeneous). They presumably test the same competency, with c making higher demands. In case 2, the three items are of approximately equal difficulty, but the mutual correlation of item 1 and item 2 with item 3 is weak. In this case, we suspect multidimensional requirements, so item 3 additionally might test (with item demand c) a different competency than the other two items.

An interesting question in the context of this project concerns the level of granularity at which such differences in item demands can be found by comparing the individual solutions of the students across all items. At the coarsest level, almost all items of a mini-project will measure more or less the same competencies, e.g. "implement the abstract data type list with the most important operations", while at the finest level all students are likely to have mastered all requirements, e.g. "implement a conditional statement in Java". In between, however, there must be a level at which relevant differences in both item demands and individual abilities can be found. Our hypothesis is that these are at the level of certain patterns required for the solutions. Potential candidates for such patterns would be the design patterns of software engineering, see e.g. [13], computational thinking patterns by Basawapatna et al. [3] or programming abilities like "implementing the ADT BST".

According to *Classical Test Theory*, the psychometric construct of interest (e.g. intelligence, motivation or competency) is considered to be measured directly by item scores, although the results are considered to be error-prone. It is obvious that this straight-forward approach is not suitable for measuring such complex constructs as competencies. In contrast, *Item Response Theory* (IRT) treats the constructs of interest as latent psychometric constructs that cannot be measured directly [46]. Instead, IRT considers the conditional probability $P(X_{i,k} = 1 \mid \Theta_i, \beta_k)$ that a person i with the manifestation Θ_i of the psychometric construct (person parameter) responds on an item k with the item difficulty β_k (item parameter) in a certain way (e.g. by a correct answer = 1) as a function of Θ_i and β_k.

$$P(X_{i,k} = 1 \mid \Theta_i, \beta_k) = f(\Theta_i, \beta_k) \tag{1}$$

The function $f(\Theta_i, \beta_k)$ is determined by the chosen psychometric model (e.g. the Rasch Model (RM), see below) that is assumed to fit the observations in the best way [46]. Its graph is called *Item Characteristic Curve* (ICC). One of the advantages of this approach is that Θ_i and β_k are measured

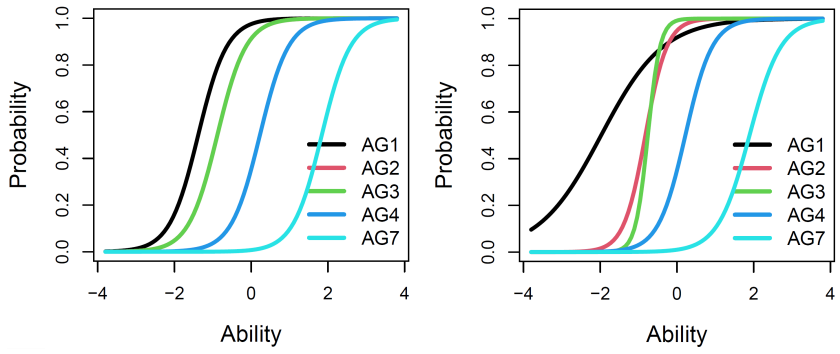


Fig. 2. Exemplary ICCs of 1-Parameter and 2-Parameter Model for the same set of subtasks (AGs)

basically on the same scale. Unfortunately, for most models, the person and item parameters have to be estimated by elaborate numerical approximations.

Since we aim to define each programming competency by a set of tasks assigned to it, we are particularly interested in unidimensional (homogenous) item sets that measure predominantly one competency. For such sets, the suitable psychometric model would be the "basic" unidimensional (monofactorial) Rasch Model (RM) with one parameter (1F1P) [45]:

$$P(X_{i,k} = 1 \mid \Theta_i, \beta_k) = \frac{\exp(\Theta_i - \beta_k)}{1 + \exp(\Theta_i - \beta_k)} \tag{2}$$

As Θ_i is regarded as the variable in this formula, β_k is the (only) parameter. In the ICCs of this model, variation of β_k causes a horizontal shift of the curves. As there isn't any other parameter, all ICCs have the same slope. Figure 2 displays a typical ICC on its left side.

Provided that this model is applicable, some convenient simplifications can be made. For example, the sum over the scores of all individual items is a sufficient statistics, which means that the (estimated) person parameter depends only on the total number of correct answers of this person. It does not matter, which questions the person had responded to correctly.

However, if different slopes of the ICCs are expected, an extended psychometric model is needed. In the two-parametric *Birnbaum Model* (BM), the additional *discrimination Parameter* δ_k represents this variation of the items in slope (1F2P) [6]:

$$P(X_{i,k} = 1 \mid \Theta_i, \beta_k) = \frac{\exp(\delta_k(\Theta_i - \beta_k))}{1 + \exp(\delta_k(\Theta_i - \beta_k))} \tag{3}$$

Obviously, the single-parameter RM is a special case of the BM for $\delta_k = 1$. The right side of Figure 2 shows an ICC of the BM for the same item combination as the RM-ICCs on the left side to visualize the approximate nature of the RM-ICCs.

In connection with the application of psychometric models, a number of criteria are often mentioned which, depending on the context, can be interpreted as prerequisites for this application or as properties of the models:

> A central assumption of most IRT models is *local stochastic independence*. The exact meaning of this axiom is that response variables are conditionally independent of each other, the condition being the value of the latent variable Θ. The main application of this axiom is the fact that the probability of a response pattern, originating from a single person, is written as the product of the probabilities of the item responses. [60, p. 181]

> *Separability* denotes the property that person and item effects on the response behavior can be isolated from each other. [48, p. 28]
>
> Rasch introduced the term *specific objectivity* to address the important property of measurement models to assure that the measure of one sort of objects (e.g., persons) is independent of the other objects (e.g., items) involved in the procedure of measurement. [48, p. 28]

After ensuring that the selected model basically fits, one would like to calculate the values of the person and item parameters that result from the actual measurements. Given these values, it would be possible to calculate how likely the observed values of the response matrix would be under these conditions, i.e. how well the selected model with these parameter values fits the actual measurement results. However, unfortunately, the person or item parameters cannot be calculated directly from the IRT model equations in most cases. Instead, complex numerical approximation methods are needed, which can only estimate their values. Basically, these methods look for a set of parameter values that maximizes the probability that the selected model with exactly these values predicts the observed response matrix. To avoid confusion with a person's probability of a particular response to an item, this probability is called "Likelihood". Accordingly, such approximation methods are called "Maximum Likelihood (ML) Methods" [36]. The calculations cannot be explained at this point due to limited space, thus we refer to for the details [36]. The good news is that for most IRT models, all these calculations can be performed quite easy by applying the functions of certain packages for R, for example eRm [32] or ltm [47].

2.4 Latent Trait Analysis

To investigate the homogeneity of a set of items (see Section 2.3), traditionally classical explorative factor analysis would be applied. Yet, as the scale is dichotomous in this case, this is not applicable. The reasons are explained in detail by Bartholomew in [2, pp. 212ff]. Therefore, we applied the methodology of *Latent Trait Analysis* (LTA) as presented in Chapter 8 of [2].

According to this methodology, it is assumed that the responses of the persons to a given set of items can be described by a certain psychometric model, for example by the monofactorial Rasch Model (1F1P) (see Section 2.3). Under this assumption, one can estimate all person and item parameters based on the scoring matrix of the responses (by ML methods, see also Section 2.3). Using the estimated values of the parameters, by calculating the probability $P(X_{i,k} = 1 \mid \Theta_i, \beta_k)$, the expected number of occurrences $E(r)$ of all possible response patterns r (e.g. 01101 in the case of 5 items) can be calculated. For p dichotomous items, there are 2^p response patterns (i.e. combinations of 0s and 1s with the length p). For each pattern r, its expected frequency $E(r)$ is compared to the actually observed pattern frequency $O(r)$. For the differences, the log-likelihood test statistic G^2 and the common χ^2 statistic are calculated that both describe the differences of the expected and the measured values. Both statistics are distributed approximately according χ^2. Thus, we can estimate the goodness of fit of the applied model by this way.

$$G^2 = 2\sum_{r=1}^{2^p} O(r)\ln\frac{O(r)}{E(r)}; \qquad \chi^2 = 2\sum_{r=1}^{2^p}\frac{(O(r)-E(r))^2}{E(r)} \tag{4}$$

As both statistics are approximately χ^2 distributed, we can estimate the goodness of fit of with df degrees of freedom as follows [2],

$$df = 2^p - p(q+1) - 1 \tag{5}$$

where p is the number of items and q the number of psychometric factors (e.g. 1 in the case of a unidimensional model). The precondition for this calculation is a sufficient number of observations.

According to [2], it has to be large enough to ensure that the frequency of each pattern has an expectation value of more than 5, equivalent to

$$n \geq 5 \cdot 2^p \tag{6}$$

In the case of 6 items for example, this results in a minimum of 320 data sets.

2.5 Model Tests

As already explained, in any case the fit of the chosen IRT model has to be validated for the specific situation it has been applied. Even if the LTA has proven that the model was able to predict the actually observed responses of the students quite well for the investigated item set (i.e. within the χ^2 limits according to $p < 0.05$ for each response pattern, see Section 2.4), some residual uncertainty still remains due to eventually unproven application preconditions of the respective model, e.g. unidimensionality. And although if we would have proven that we have found a suitable model, there may be an even better fitting one (e.g. the Birnbaum Model with two item parameters instead of the Rasch Model). Therefore, a set of suitable model tests has to be performed that can be grouped in three categories:

(1) tests based on classical test theory respectively descriptive statistics (*classic tests*),
(2) tests according IRT, based on the numerical estimation of item and person parameters of different models (*parametric tests*), see Section 2.3,
(3) tests according IRT, but based on the comparison of the actual observed response matrices with randomly produced "synthetic" response matrices (*nonparametric tests*).

From classical tests, we will apply Cronbach Alpha [9], Phi-correlation [11] and point-biserial correlation [57] between items, as well as the distribution and variance of the “classical” item difficulties.

The parametric tests investigate, based on the estimated parameter values, how far specific objectivity and homogeneity hold for this test result, assuming that if they hold, the Likelihood of a well-fitting model should be nearly the same for any subgroup of participants or items. For this purpose, either the person sample in the case of Andersen’s Likelihood-Ratio-Test [1] or the item set in the case of Martin Löf’s Test [35] is split into subgroups according to different splitting criteria, for example above/below median (respectively mean) of total score over persons respectively over items, or according to gender. Afterwards, the Conditional Likelihood (CL) values of these subgroups are compared with the respective value of the total person or item set. For the subgroups, a test-specific statistic, basically representing the Likelihood of this model given the estimated parameters, is calculated. Finally, the p-value for the hypothesis that the statistic would be equal for all subgroups is calculated. The hypothesis (and thus the model) is rejected if the result is below the desired significance level (e.g. 5%), see also [14].

As mentioned above, parameter estimation of IRT models requires comparatively large samples (see Section 2.3). Unfortunately, this is not always feasible in practice. If the sample is smaller, the results of test evaluation may be incorrect, it may even be that the items cannot be evaluated at all with the chosen model, making the use of parametric tests inadvisable or even impossible. Under such circumstances, the applied nonparametric tests have to rely on other criteria that do not require the estimation of the model parameters. The principle of the nonparametric tests is to calculate certain test statistics T on the observed matrix A_0 (resulting in T_0) as well as on *nsim* synthetically generated matrices A_s (resulting in T_s) and to compare the results to find out, for how many of the A_s the RM fits worse than to A_0. If the result of T is worse for most of the A_s compared to A_0, we can conclude that the RM fits comparably well to A_0. The details of all nonparametric-tests we will apply in Section 4.4 can be found in [26].

3 TEACHING CONTEXT

For the bachelor programs "Applied Computer Science" and "Business Informatics" at the University of Duisburg-Essen, the course "Computer Science 1" (CS1) is mandatory in the first semester.

In addition to lectures, an optional 2-hour consultation session is offered each weekday by student teaching assistants to help students learn to program with Java. Throughout the semester, more than 80 small exercise assignments are provided in JACK. Working on these exercises is not mandatory, but strongly recommended to the students. Some of these exercises are discussed in detail in the recitation sessions. Each exercise is associated with several test cases within JACK. The system evaluates the solutions based on the test cases and additional static code checks and provides corresponding feedback on every submission. The students can submit a solution and receive immediate feedback, that consists of textual messages (a list of failed test cases and static code checks) and a result score in the range from 0 to 100.

Beside these small exercises, larger exercises called mini-projects (MP) are provided every two weeks in JACK. Each mini-project addresses a specific subject area and is split into five to ten subtasks of increasing difficulty. The mini-projects provide an opportunity for students to work on a cohesive set of tasks, as opposed to the small and isolated exercises mentioned above. Compilable *solution templates* (Java class files) are provided for all MPs, declaring all names of classes and methods that JACK needs for its evaluations. The students may submit any number of solutions to JACK for evaluation. The system provides extensive feedback, so that the students can improve their solution with each submission.

Each mini-project is designed to prepare the students for the following corresponding small exam, called *attestation*. It is mandatory to make at least one submission (regardless of the score achieved) for the corresponding mini-project to take part in the attestation. The unchanged solution template is also accepted for this purpose. The submission deadline for admission to the attestation (referred to as *attestation admission deadline* for short) was usually 6-8 days after the start of the MP (due to holidays sometimes even 2-3 weeks).

The students can get a maximum of 100 points for each attestation. For the admission to the final examination, at least 330 points are required. In each attestation, students have to solve a given task within 45 minutes, working in a computer pool under exam conditions without Internet access. They must use Eclipse as an integrated development environment (IDE). Each student may submit several solutions to JACK, where the number of points is calculated from his/her best submitted solution. Students may review their submitted solutions, scores and feedback under the supervision of an undergraduate teaching assistant in a consulting session after the exam.

The following topics are covered during the 14 weeks of the CS1 course, with the corresponding mini-projects indicated in parentheses:

- 1, 2: Introduction to Java, enabling students to write first small Java programs without digging into conceptual details,
- 3, 4: Classes, objects, attributes, variables, primitive data types and control structures,
- 5, 6: Arrays (MP1) and linked lists (MP2),
- 7, 8: Binary (search) trees and recursion (MP3),
- 9, 10: Inheritance and interfaces (MP4),
- 11, 12: Generics and exceptions (MP5 on database implementation),
- 13, 14: Lambda expressions and graphs.

In the context of AKoFOOP, we evaluate the data of the course runs in the winter terms 2015/16, 2017/18, 2018/19, and 2019/20. In winter term 2016/17, the course was held by a different lecturer, who changed the course structure and the attestation topics substantially. Yet, during the evaluated terms, some other relevant changes occurred. All attestation tasks were revised or replaced by

Table 2. Number of students and submissions per term for mini-projects

No.	Number of students					Number of submissions				
(Variant)	*15/16*	*17/18*	*18/19*	*19/20*	*Total*	*15/16*	*17/18*	*18/19*	*19/20*	*Total*
MP0 (v1)	669				669	2413				2413
MP0 (v2)		477			477		921			921
MP0 (v3)			479		479			874		874
MP0 (v4)				462	462				1431	1431
MP1	580	663	662	630	2535	2362	2142	2235	2114	8853
MP2	483	579	607	596	2265	3643	3105	4655	3188	14591
MP3 (v1)	399	434			833	1630	1794			3424
MP3 (v2)			496	505	1001			3756	2499	6255
MP4	316	334	381	417	1448	1090	955	1482	1620	5147
MP5	260	248	311	354	1173	1005	1070	1463	1223	4761

new ones for the winter term 2017/18, without changing the main topics of each attestation. In winter term 2018/19, the tasks for mini-project 3 were replaced again, because it was perceived as too difficult for the students. In winter term 2019/20, the introduction to Java was changed from a separate "Java crash course" to a regular part of the lecture, now covering less Java constructs and more conceptual details on variables, primitive data types, control structures and arrays. Week three and four are now used solely for object-oriented concepts and lists.

Table 2 provides the figures of students and submissions per term for the mini-projects.

4 METHODOLOGY AND RESULTS

In this project, we follow a research methodology that consists of many different steps. Each step produces results on which the next step is based. Separating the presentation of methodology and results therefore makes little sense at this point. Consequently, in this chapter we describe our research strategy with the intermediate results of the respective steps. However, only the steps that have actually been used for the purpose of defining competencies will be discussed here.

One of the main objectives of this project is to define competencies in the field of object-oriented programming as described by [51]. For this purpose, we need an item set for each such competency, which is sufficiently homogeneous in the sense of the IRT, i.e. that the probability that a person can solve these items is determined predominantly by the manifestation of this respective competency. In some cases, we have already succeeded in doing so. This section is intended to illustrate the whole research path from extracting the raw data to the definition and analysis of such competencies based on the mini-project MP 3 of the generation 2019/20.

4.1 Analysis of Item Demands

The obvious starting point of the data analysis was to identify the cognitive requirements to solve the mini-projects. For this purpose, we first carried out a qualitative analysis of the task sheets and the expert solutions of the mini-projects, which were created by the lecture team. For illustration, Appendix A.1 shows the worksheet from MP3 from the 2019/20 winter semester.

All solution steps that required more than one line in the program were coded as *competency candidates* (CCs), supplementary requirements as *SUPP*, and smaller, single-line solution steps as *elementary skills* (ES). This resulted in a set of 58 competency candidates and 184 elementary skills (including all variants) over all mini-projects of all generations. Second, two researchers discussed the requirements of the task sheets and expert solutions to determine the respective item demands.

Table 3. Item demands of MP3 (19/20)

Cat.	Coded Description	AG1	AG2	AG3	AG4	AG5	AG6	AG7
CC	Insert an element in BST	X						X
CC	Define recursive method	X	X	X	X	X	X	X
CC	Traverse a path in BST	X	X				X	X
CC	Search for elements in BST		X		X	X	X	X
CC	Count all entries in BST			X				
CC	Traverse BST			X	X	X		
CC	Filter elements in BST				X			
CC	Concatenate two arrays				X			
CC	Remove element from BST and re-sort						X	X
CC	Change element in BST and re-sort							X
ES	Use count variable				X			
ES	Apply reference comparison (equals)				X	X		
ES	Instantiation	X			X			
ES	Conditional branch if-else	X	X	X	X	X	X	X
ES	Nested conditional branching	X	X			X	X	X
ES	Check for null	X	X	X	X	X	X	X
ES	Comparison operator	X	X	X	X	X	X	X
ES	Variable declaration	X			X		X	
ES	Pass parameter values to variable	X			X		X	X
ES	Assign variable to an attribute	X					X	X
ES	return statement		X	X	X			
ES	Array initialization				X			
ES	Count loop				X			
ES	Determining the length (of an array)				X			
ES	Apply methods of a referenced object	X	X	X	X	X	X	X
ES	Array iteration				X			
SUPP	Separate treatment of individual elements					X		X

The results were mapped to the originally coded ESs and CCs. The synthesis formed the final list of item demands for further work, see Table 3.

The mutual relationship between the individual demands was also coded and represented as a graph. The nodes represent demands or items, the edges a dependency in the sense of a precondition. In this sense, demand A is a precondition of demand B, if the program code of A is included in the code of B. The semantics is that the mastery of A is a prerequisite for mastering B. Since our focus is on more complex skills, we limited this step to the CCs. For simplification and to support publications, the original German identifiers of the CCs have been replaced by English names. Figure 3 displays the result for the subtasks (items) AG 1, .. AG 7 of MP3.

In other work still in progress, a selection of student solutions is being analyzed in terms of content to find variations in the solution of MPs and typical errors or problem areas and to determine their prevalence.

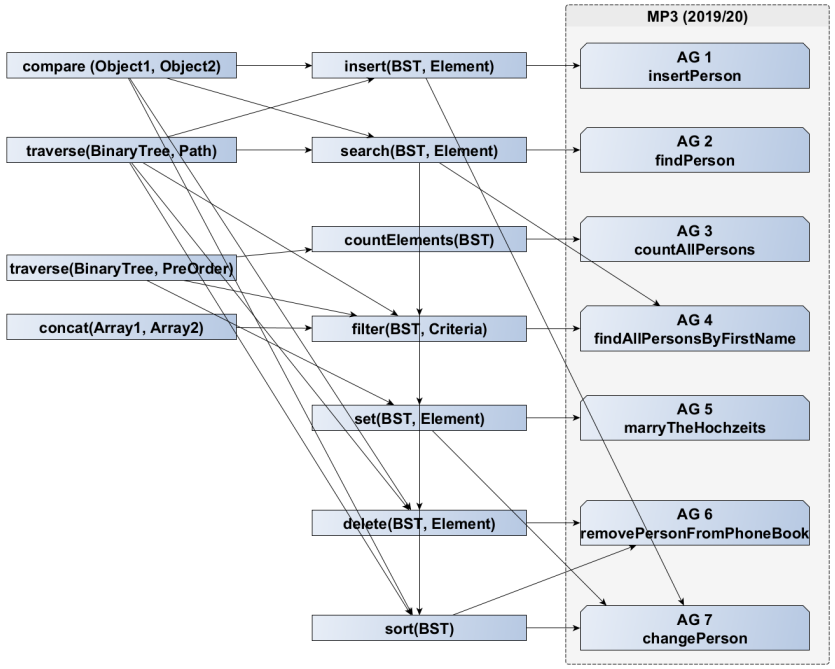


Fig. 3. The competency candidates as item demands of MP3 (2019/20)

4.2 Data Preparation

For persistence, all program files submitted by students to JACK were first exported to a unified file system, where each individual submission is accessible via a path according to the following scheme: *"year/exercise-id/student-id/submission-no/"*. These unique folders contain the submitted source code files and a text file containing all automated feedback produced by JACK on that submission. The student folders use pseudonyms as ids, so that submissions from different exercises or years belonging to the same person can be identified without actually identifying that person.

If a student solution is compilable, a series of static and dynamic tests are run based on test cases defined specifically for the respective subtask. A test case can fail with several different error messages. On the other hand, several test cases are usually necessary to check the correctness of a solution with respect to a subtask. The results of these tests are recorded for each MP in a special *sequence table*, where each line represents one submission and each column either a test case or a sub-task. A "1" in a cell indicates that no negative feedback has been generated for that test case or sub-task in the respective submission. If there was negative feedback, the corresponding cell contains a "0". In addition, JACK calculates a total score from 0 to 100 points for each solution from the results of the test cases, according to special rules for each subtask.

A statistical analysis of the sequence tables showed that the number of solutions per student uploaded to each mini-project varied widely. Overall, the 500 students had uploaded 2421 submissions, resulting in an average value of 4.8 submissions per student and a median value of 2. Most students (194) had uploaded only 1 submission, while the maximum number was 62 in one case.

From the IRT analysis of the sequence tables, we expected valid information about the coherence between subtasks (items) based on students' individual cognitive performance on these tasks. However, due to the complicated submission process and weak control over students during

submission, the sequence tables had several serious weaknesses in this regard. In detail we faced the following problems:

(1) Unlimited submission period: Students can submit solutions of all MPs until the end of the semester. Yet, after the attestation admission deadline, their further learning process was no longer subject to any control. In consequence, the later submissions do not provide reliable information on the respective competencies.
(2) Uncompletely evaluated submissions: The evaluation of the solutions by JACK can fail in several ways, e.g. due to compiler errors or missing class files, which cannot be seen from the sequence tables alone. However, for the IRT evaluation we can only use solutions that have been fully evaluated by applying all cases of JACK's dynamic tests.
(3) Copied solutions: Many students solve the MPs by copying and pasting other students' programs or the unmodified solution templates. Both approaches produce large amounts of more or less identical submissions that are useless as descriptions of the submitters' individual learning processes.
(4) Multiple solutions per student: The number of submissions of the students varied greatly. The cognitive performance of students with many submissions would be overweighted.
(5) Many trivial solutions: Due to trial-and-error strategies or by uploading the empty solution templates, a large number of (totally incorrect) rows in the sequence tables contained only the value 0 in all columns. Also, numerous (totally correct) rows contained only the value 1, due to the comparably easy tasks or by frequently copying correct solutions from other students. These rows do not provide any information about the specific inter-relationships between items that we need to define a competency, but only cause seemingly high (purely numerical) correlation values between items.

To solve these problems, we reduced the rows of the sequence tables using filtering strategies:

(1) *Cut*: All solutions submitted after the deadline for the attestation were removed.
(2) *T3*: We removed all solutions except those 2180 submissions, where all dynamic tests of JACK had been fully executed (called solutions of Type 3).
(3) *JP*: Using JPlag[2] [44] plagiarism software, which provides a percentage measure of agreement, we compared each submission against all others. We then grouped all solutions that were 100% concordant according to JPLAG. However, because JACK provided different total scores even within these groups of seemingly identical solutions, we further divided these groups into subgroups according to the total scores. Finally, we selected one submission from each subgroup to represent all their (nearly identical) solutions.
(4) *Last*: In order to avoid the disadvantage of the last items of the MP as far as possible, only the *last* solution submitted was retained for every student.
(5) *R*: All totally correct or totally incorrect solutions were removed, except a certain rest *nrest* for numerical reasons, where $nrow(SeqTab)$ ist the number of rows of the sequence table and $ncol(SeqTab)$ its number of columns:

$$nrest = floor\left(\frac{5 \cdot nrow(SeqTab)}{2^{ncol(SeqTab)}}\right). \tag{7}$$

Because these filtering operations are not necessarily commutative, they have to be applied in a specific order to ensure proper selection results. By combining the operators where appropriate, we obtained four different filtering strategies for the submissions considered: Cut-T3-Last, Cut-T3-Last-R, Cut-T3-JP-Last, and Cut-T3-JP-Last-R, where the order of application corresponds with the reading direction from left to right. For each of these strategies, the overall results were collected

[2]https://jplag.ipd.kit.edu/

Table 4. Number of submissions in the response matrices of MP3 (19/20)

Strategy	Submissions	Tot.incorrect	%	Tot.correct	%
Cut-T3-Last	471	145	30.8	122	25.9
Cut-T3-JP-Last	274	23	8.3	63	23.0
Cut-T3-Last-R	240	18	7.5	18	7.5
Cut-T3-JP-Last-R	208	10	4.8	10	4.8

in a table called *response matrix*, which was used as data source for the IRT analysis. The response matrices are designated according to their respective strategies. Table 4 lists the resulting numbers of total, totally incorrect and correct submissions for each considered response matrix.

4.3 Latent Trait Analysis

Unfortunately, the LTA-methodology proposed by [2] is confirmatory in nature and therefore requires an a priori defined set of items to be tested. We solved this problem by a brute force approach: calculating both statistics G^2 and χ^2 for all possible combinations of items consisting of suitable numbers of tasks k. As these item sets should be used for competency definition and measurement, we regarded 5 levels of the personal ability Θ as the absolute minimum, which requires at least 4 items. The maximum of k was determined by the lowest acceptable expectation values of pattern frequency, being $5 \cdot 2^k$. Applied to the number of n submissions in the respective Response Matrix, this results in the condition for minimal $n > 5 \cdot 2^k$ and thus for maximal $k < \log_2(n/5)$, see Equation 6 in Section 2.4.

Our brute-force LTA resulted in a list of item combinations of length k that had produced acceptable values for the statistics G^2 and χ^2 according [2]. More precisely, we have selected all combinations of k items where both G^2 and χ^2 did not exceed the χ^2 limits for the respecting values of df, see Equations 4 and 5 in Section 2.4). For the calculations we applied the ltm package in GNU R [47]. In the following, we will call these acceptable item sets *Candidate Item Combinations* (CIC). Applying the strictest selection strategy CUT-T3-JP-LAST-R to MP3 of 2019/20, we found 13 CICs of length $k = 4$ and 5 CICs of maximum possible length $k = 5$ (see above). Intending to find the longest possible item combinations, we focused on the 5-item combinations from this point on, see Table 5. The value of *Chiquote* gives the ratio of the actual value of χ^2 divided by the maximum value according to $p < 0.05$ and the given degrees of freedom df.

Table 5. LTA results of MP3 2019/20, Filter Strategy CUT-T3-JP-LAST-R

CombID	Item Combination	Missing Items	Chiquote
5_1	AG1 AG2 AG3 AG4 AG5	AG6 AG7	0.76
5_2	AG1 AG2 AG3 AG4 AG6	AG5 AG7	0.64
5_3	AG1 AG2 AG3 AG4 AG7	AG5 AG6	0.82
5_7	AG1 AG2 AG4 AG5 AG6	AG3 AG7	0.47
5_8	AG1 AG2 AG4 AG5 AG7	AG3 AG6	0.64

Since the length $k = 5$ of the longest possible CIC was smaller than the total number $n = 7$ of items in the MP, a question is whether each item is included in at least one of the CICs, which is the case here.

Table 6. Classic item difficulties of MP3 (19/20)

	Cut-T3-Last	Cut-T3-JP-Last	Cut-T3-Last-R	Cut-T3-JP-Last-R
AG 1	0.665	0.850	0.871	0.865
AG 2	0.597	0.752	0.738	0.736
AG 3	0.580	0.715	0.704	0.688
AG 4	0.480	0.562	0.508	0.486
AG 5	0.520	0.617	0.588	0.558
AG 6	0.342	0.387	0.238	0.255
AG 7	0.285	0.285	0.125	0.120
StDev	0.138	0.202	0.272	0.266

Table 7. Inter-item Phi-Correlations of MP3 (19/20)

	AG 1	AG 2	AG 3	AG 4	AG 5	AG 6	AG 7
AG 1	1	0.34	0.16	0.27	0.22	0.23	0.15
AG 2	0.34	1	0.4	0.3	0.41	0.28	0.12
AG 3	0.16	0.4	1	0.3	0.4	0.13	0.03
AG 4	0.27	0.3	0.3	1	0.48	0.36	0.2
AG 5	0.22	0.41	0.4	0.48	1	0.34	0.21
AG 6	0.23	0.28	0.13	0.36	0.34	1	0.56
AG 7	0.15	0.12	0.03	0.2	0.21	0.56	1

4.4 Rasch Model Tests

Before looking at the results of the model tests, we will briefly compare the classic difficulties of the items of MP3 (19/20). As is well known, these are represented simply by the respective proportion of correct solutions. Higher values consequently mean easier items in terms of the requirements. Because different submissions are used for each student's solutions depending on the selection strategy, the item difficulties naturally differ depending on this strategy. In the case of the reduced response matrices (selection step "R") one has to take into account that possibly numerous rows were deleted in which all cells contained the same value (1 or 0). Table 6 shows the values for four selection strategies respectively response matrices.

Obviously, in all matrices, item AG 1 is the easiest, followed by AG 2 and AG 3, which are very close to each other. The most difficult is AG 7. The standard deviation is rather small in Cut-T3-Last, but can be doubled by reducing the size of the matrices. In the strictest selection Cut-T3-JP-Last-R the item difficulties are also very evenly distributed. AG 2 and AG 3 are still closest to each other.

Table 8 lists the results of the model tests of the CICs in Table 5 that are based on classical statistics or on the LTM-Package [47], all computed with R. The row identifiers in Table 8 have the following meanings, where "AG xy..." abbreviates "AG x AG y ...".

1F2P ICCs: Crossings ICCs according to the 1F2P-Model (Birnbaum Model),
Phi-Corr:] Item pairs with weak (≤ 0.1) or negative mutual Phi-Correlations,
PBisCorr: Items with weak (< 0.5) Point Biserial Correlation with total score, where the respective the item is excluded from total score calculation,
Cronbach Alpha: Value of Cronbach's Alpha Coefficient [9],

Table 8. Model fit: Classic and LTM results

MP3-19-20	5_1	5_2	5_3	5_7	5_8
Items	AG 12345	AG 12346	AG 12347	AG 12456	AG 12457
1F2P ICCs	AG 2345	AG 23	AG 23		
Phi-Corr			AG 37		
PBisCorr	AG 134	all	all	AG 12	AG 1247
Cronbach Alpha	0.7121	0.6549	0.6021	0.7062	0.6586
PairAss		AG 36	AG 37, 27, 17		AG 27, 17
2-Marg		AG 36	AG 37		
3-Marg	AG 345, 145	AG 236, 136, 346	AG 137, 237, 347		
LRT: Best Model	1F1P	2F	1F1P	1F1P	1F1P
LRT: p-value	p<0.248	p<0.011	p<0.275	p<0.957	p<0.403

PairAss: χ^2 p-values for pairwise associations between items, corresponding to 2×2 contingency tables for all item pairs, non significant results may reveal problematic items [47, p. 6],

2-/3-Marg: two/three-way margins in which residuals have unacceptable values (≥ 3.5), calculated by constructing all possible 2×2 contingency tables for the available items and checking model fit using the Pearson's 2 statistic [47, p. 8],

Best LRT Model: Result of a Likelihood Ratio Test (LRT) that compares three models pairwise by calculation of AIC and BIC and Monofactorial RM with 1 parameter (1F1P), Birnbaum Model (1F2P), 2-factorial RM [47, p. 11],

LRT p-value: p-result of the (LRT) above, indicating the significance that the second (less constrained model) fits better (1F1P vs. 1F2P, 1F2P vs. 2F) [47, p. 11].

Furthermore, we performed a series of parametric model tests of the CICs in Table 5, using the eRm package for R [31, 33]. All of these tests yield a p-value as a result, which establishes the significance for rejecting the hypothesis "The simple Rasch model 1F1P fits adequately for this data set." In other words, if $p < 0.05$, the 1F1P model does not fit sufficiently well according to the respective test with a probability of 95%. Table 9 shows the results, where the row identifiers have the following meaning, for more details, see [31, 33, 34]:

LRT rs median: Andersen's Likelihood-Ratio-Test (LRT) [1] with median of the raw scores (rs) of submissions (number of correctly solved items) as splitting criterion, potentially with excluded items due to "inappropriate response patterns",

LRT rs mean: LRT with mean of the raw scores as splitting criterion, "ex" see above,

LRT full rs: The submissions are split in groups according all values of raw scores except 0 and p (number of items) e.g. $rs = 1, 2, 3, 4$ for 5 items,

LRT mean ts: LRT with the mean of total score (ts, calculated by weighting items) as splitting criterion,

LRT median nr subs: LRT with the median of the number of submissions of the respective student as splitting criterion,

LRT split items: Consecutive LRT using each of the items as splitting criterion (submissions with value 1 vs. submissions with value 0), only problematic items are listed with their p-values,

Waldtest mean: Wald-Test [14] on the level of single items with the splitting criterion mean of raw score,

Table 9. Model fit: parametric tests by eRm

MP3-19-20	5_1	5_2	5_3	5_7	5_8
Items	AG 12345	AG 12346	AG 12347	AG 12456	AG 12457
LRT rs median	0.169	0.801 ex AG 6	0.905 ex AG 7	0.346 ex AG6	0.262 ex AG 7
LRT rs mean	0.169	0.128 ex AG 1	0.905 ex AG 7	0.346 ex AG 6	0.262 ex AG 7
LRT full rs	0.515	0.178	0.349	0.181	0.208
LRT mean ts	0.221	0.005	0.102	0.009	0.298
LRT median subs	0.512	0.277	0.274	0.298	0.361
LRT split items	all passed	AG 1: 0.052 AG 5: 0.022	AG 1: 0.052 AG 3: 0.032	all passed	all passed
Waldtest rs mean	AG 5 0.051	AG 3 0.025	all passed	all passed	all passed
MLT rs mean	0.438	0.449	0.506	0.223	0.903
MLT rs median	0.23	0.881	0.957	0.223	0.299

MLT mean: Martin-Löf-Test (MLT) [35] on the level of item set with the splitting criterion mean of raw score,
MLT median: Martin-Löf-Test [35] with the splitting criterion median of raw score.

Finally, several nonparametric (NP) model tests were conducted on the same set of CICs [26], using the eRm package for R again [31, 32], with the following matrix simulation parameters: starting value for the random number generator $burn_{in} = 200$, number of generated matrices $n_{\text{eff}} = 1000$, number of void (skipped) matrices $step = 64$, and starting value for the random number generator $seed = 123$ (see [26, 33, 60]). In principle, these NP tests return a p-value with the same meaning as in Table 9. For the tests on item level, the cells of Table 10 indicate additionally the problematic item pairs with $p < 0.05$. For more information, see [26, 33, 43, 60]. Table 10 shows the results with the following identifiers (according to [26]):

T10: A global test (on the whole item set) for the investigation whether there exists a violation of the assumption of measurement invariance, split criterion was median of raw score,
T4: A test at the item level for the detection of too many or too few positive answers within a group, split criterion was median of raw score,
T11: A global test for inappropriate inter-item correlations, as explained in Section 2.5,
nT1: A test at the item level for too many equal response patterns 00 and 11,
T1m: A test at the item level for too few equal response patterns 00 and 11,
T1l: A test at the item level for too many equal response patterns 11,
T2: test at the item level for a too large variance in the raw score of a subscale,
T2m: test at the item level for a too low variance in the raw score of a subscale,
Tmd: A global test for inappropriate correlations of subscales,
Tpbis: A test at the item level for inappropriate patterns.

In addition to the model tests, the ICCs of the Birnbaum model (see Figure 4) can provide valuable information on the fit of the simple Rasch model 1F2P to the CICs. However, it must be taken into account that the course of these ICCs is based on only 6 support points each because of the small number of items. In Figure 5, these points are drawn for the combination 5_7 as an example.

Table 10. Model fit: nonparametric tests of R-package eRm

MP3-19-20	5_1	5_2	5_3	5_7	5_8
Items	AG 12345	AG 12346	AG 12347	AG 12456	AG 12457
T10	1,000	0.285	0.487	0.944	0.961
T4	1,000	0.581	0.703	0.729	0.803
T11	0.093	0.114	0.134	0.633	0.423
T1	AG 45: 0.034	None	None	None	None
T1m	AG 13: 0.022	AG 13: 0.018	AG 13: 0.021	None	None
		AG 36: 0.006	AG 37: 0.023		
T1l	AG 45: 0.034	None	None	None	None
T2	AG 45: 0.034	None	None	None	None
T2m	AG 13: 0.022	AG 36: 0.006	AG 13: 0.021	None	None
			AG 37: 0.023		
Tmd	0.069	0.539	0.412	0.121	0.047
Tpbis	None	AG 3: 0.019	None	None	None

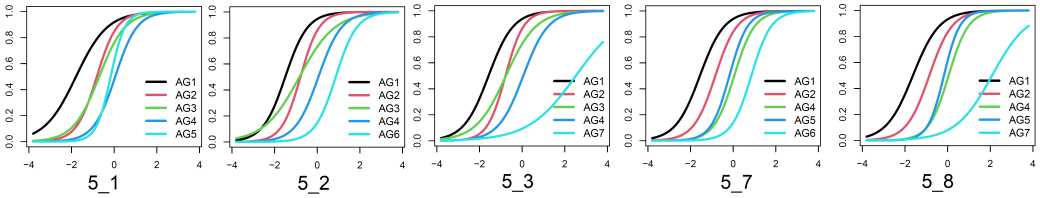


Fig. 4. ICCs of our CICs by applying the Birnbaum Model

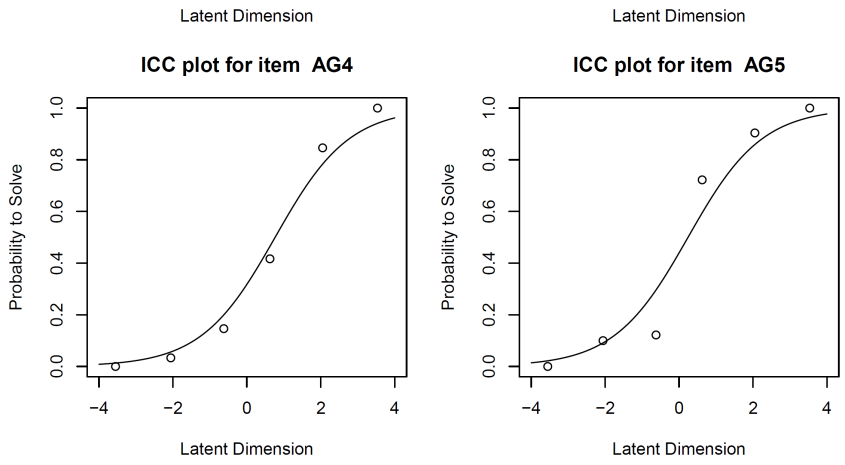


Fig. 5. Support points of some ICCs of CIC 5_7 according the the 1F1P-Model

4.5 Definition of Competencies

As stated in Section 1, the first goal of this project was to define competencies by appropriate, sufficiently homogeneous itemsets. On the basis of the model tests, it can now be determined which itemsets of MP3 (19/20) are suitable for this purpose.

Apparently, all itemsets cause problems with one test or another, with CIC 5_7 performing by far the best. There are only two minor weaknesses here. First, two items have only a weak Point Biserial Correlation with the total score (PBisCorr) in Table 8, which, however, is still the best result of all combinations examined. Second, the significant p-value 0.09 occurs with Andersen's LRT with respect to the total score (LRT mean ts) in Table 9. Furthermore, this combination also achieves the best values of all combinations in Latent Trait Analysis (Chiquote) in Table 5, Cronbach's alpha and in the model comparison of 1F1P with 1F2P in Table 8. Finally, the ICCs of the Birnbaum model look almost perfect, they have about the same slope overall, see Figure 4. Overall, a sufficiently good model fit of 5_7 with 1F1P can be assumed here. In consequence, the item set AG 1, AG 2, AG 4, AG 5, and AG 6 might be used to define a single competency.

The second best combination is clearly 5_8. The values of the LTA result as well as of Cronbach's alpha are good, however, Table 8 shows minor problems with the Point Biserial Correlation with total score and with pairwise inter-item association. In Table 9 everything looks very good, while in Table 10 a significant value is found in the test Tmd. The Birnbaum ICCs also show very similar slope, although the (very difficult) item AG 7 is an outlier. Overall, one could therefore also use the item set AG 1, AG 2, AG 4, AG 5, and AG 7 as a definition of a competence.

All other itemsets of Table 5 show multiple serious problems in the model tests, therefore they seem not eligible for defining a competency. Most of these problems seem to be triggered by the item AG 3, which may create inhomogeneity in item demands. This hypothesis is supported by the item's moderate difficulty (see Table 6) combined with a weak correlation with the other items, see Table 7. More details on this reasoning can be found in Section 2.3. A look at the demand analysis of AG 3 (see Section 4.1) raises the suspicion that this inhomogeneity is generated by the implementation of a counter. In fact, when we examined a larger number of student solutions, we found very different implementations of this task, which we will discuss in more detail in a follow-up publication.

In summary, the items of the MP3 (19/20), with the exception of AG 3, appear suitable for defining a competency. However, our results only ensure that either AG 6 or AG 7 can be included. The 6-item combination AG 1, AG 2, AG 4, AG 5, AG 6, AG 7 could not yet be tested in the Latent Trait Analysis due to the insufficient number of entries in the response matrices. However, a comparison of the item demands of AG 6 and AG 7 in Table 3 and Figure 3 (see Section 4.1), shows that, at most, one could use the ability to implement *delete(BST, Element)* to distinguish between the competencies defined via AG 6 and AG 7, respectively, but this makes little sense in terms of a complete implementation of the ADT BST. Based on the current state of our research, we therefore formally define the competence C_1 as the following combination of item demands:
C_1 := CombItemDemands(AG 1, AG 2, AG 4, AG 5, AG 6, AG 7).

However, we keep in mind that for the test of C_1, we have so far only demonstrated the suitability of item sets 5_7 respectively 5_8 separately. The best description of C_1 would be in our eyes:
C_1: *The ability to solve everyday problems by implementing the ADT BST.*

The cognitive structure of the competency C_1 defined in the preceding Section 4.5 is essentially represented by Table 3 and the graph in Figure 3, see Section 4.1. However, in Table 3 the column AG 3 has to be removed as well as the nodes *AG 3* and *countElements(BST)* with all of their edges in Figure 3. Based on this structure and other information regarding the application context, our resulting competency C_1 can be represented in the Competency Structure Model, see Table 11.

Table 11. Concise overview on our competency C_1 including categorization in the Competency Structure Model of OOP, adapted from [27]

Competency Description: The ability to solve everyday problems by implementing the Abstract Data Type *Binary Search Tree*.	
Included tasks: Insert an entry, Find a single entry, Find set of entries, Change entries without resorting, Delete entries and resort tree, Change entries and resort tree.	
(Sub-)Dimension	**Manifestation**
1. CS knowledge and skills [37]	
1.1. Data structure	Primitive data types, array, *binary (search) tree*
1.2. Class & object structure	Classes, objects, associations, inheritance, polymorphism
1.3. Algorithmic structure	Sequences, loops, conditional statements, nesting, recursion
1.4. Notional machine [4, 54]	High-Level OO Machine, see [54]
2. Representation language [50]	
2.1. Semantic Paradigm	Object oriented/imperative programming language (Java)
2.2. Modality	Textual
2.3. Level of Formalisation	Programming language
3. Cognitive operations	
3.1. CS problem solving stage [49, 62]	Understanding the problem – determine how to solve the problem – translating the problem into a computer language program – testing and debugging the program
3.2. PISA problem solving stage [41]	Exploring and understanding – Representing and formulating
3.3. Cognitive process category [12]	Interpreting (Remember, Understand, Analyze) – Producing (Apply)

5 DISCUSSION

In this article, we have shown that among the subtasks of an exemplary mini-project (MP3) of our lecture, there is a subset that is suitable for empirically defining a competency (C_1) based on IRT according to [51]. This result was confirmed by the outcomes of a variety of test procedures for the items and their interrelatedness. The totality of these test procedures covers both classical and probabilistic test theory. In consequence, from a psychological point of view, it seems likely that our students' ability to solve the items under investigation depends primarily on a particular psychometric construct, precisely this (hypothesized) competency (C_1).

However, these results are subject to certain limitations of validity, reliability and transferability. First, there is the dependence of the results on the specific characteristics of the context in which all this research was conducted. Although there were certain variations over the different semesters, by limiting the exemplary research to winter term 2019/20, we still only considered a snapshot of it. Thus, the circumstances depend, among other things, on the conception of the lecture, the course of study, and possibly special peculiarities of the students in that university. One would therefore have to conduct similar studies on completely different lectures on the same subject area. We have deliberately included a detailed description of our methodology in this paper to enable fellow researchers to conduct this kind of research.

Second, we had only very weak control of the solution process of the mini-projects. The students solved the tasks at home, using partly unknown material, under unknown influences by special,

psychologically relevant events of the respective solution week. Repeating the same examinations in another semester with exactly the same lecture structure would mitigate this problem.

The influences of the selection of the considered solutions are not entirely clear. After all, we performed several filtering operations, such as reduction to a JPlag component or deletion of trivial solutions with only correct or incorrect answers. Improving the items, for example increasing the difficulty, or changing the framework of the lecture so that submitting a completely unedited program template no longer qualifies for the attestation could help here.

Further, the assignments have a stereotypical structure given all the possibilities to create programming assignments. Ruf et al. have presented a categorization for this [50], according to which we face only one of 11 possible categories (Type 2: "write code using the given code"). For a valid measurement of complicated competencies, one would have to vary this task structure, for example by having program code produced "from the scratch" without a template.

Finally, the results are based solely on the quantitative outputs of the analyses by JACK. The students' solutions must additionally be analyzed qualitatively, see 6.

6 CONCLUSION

In this article, we presented the methodology and results of our empirical research project, using the example of one mini-project, which reached the evidence-based definition of a competency in object oriented programming. We provided a detailed description of the rigorous methodology that enables fellow researchers to conduct similar research, elicit more evidence-based competency definitions, and validate existing results. Currently, we are working on the definition of competencies based on other mini-projects, for which we expect good results for at least two more cases.

We can now also approach the detailed investigation of the internal structure of the identified competency. In addition to the quantitative methods used so far, the students' solutions can also be analyzed qualitatively in order to gain insight into the variations of the possible solutions and thus delineate which of these variations can be considered externalizations of the defined competency. This evaluation is currently underway and will be published soon. We will also consider misconceptions revealed in selected student solutions and analyse interviews we conducted with 38 students.

Additional research findings will emerge from evaluations of student solutions on a larger scale. To this end, we have represented the program structure of all 48,000 solutions, comprising more than 288,000 program files, by specific syntax graphs called *TGraphs* [10] and stored them in a graph database (Neo4j) [25], on which we can now very efficiently perform queries for structural features [5, 20].

Finally, we plan to develop and pilot test instruments for the competencies based on the itemsets we used for their definition. The results can furthermore be used to establish automatic feedback strategies for programming tasks which are empirically-based and didactically sound. Those feedback strategies will be helpful for learning in lectures as well as in programming MOOCs.

ACKNOWLEDGMENTS

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Grant No. 412374068.

REFERENCES

[1] Erling B. Andersen. 1973. A goodness of fit test for the rasch model. *Psychometrika* 38, 1 (1973), 123–140. https://doi.org/10.1007/BF02291180

[2] David J. Bartholomew, Fiona Steel, Irini Moustaki, and Jane I. Galbrath. 2008. *Analysis of multivariate social science data* (2 ed.). CRC Press / Taylor & Francis, Boca Raton Fla.u.a.

[3] Ashok R. Basawapatna, Alexander Repenning, Kyu Han Koh, and Hilarie Nickerson. 2013. The zones of proximal flow: guiding students through a space of computational thinking skills and challenges. In *Proceedings of the ninth annual international ACM conference on International computing education research (ICER '13)*, Beth Simon, Allison Clear, and Quintin Cutts (Eds.). ACM, New York, NY and USA, 67–74. https://doi.org/10.1145/2493394.2493404

[4] Michael Berry and Michael Kölling. 2013. The design and implementation of a notional machine for teaching introductory programming. In *Proceedings of the 8th Workshop in Primary and Secondary Computing Education*, Michael E. Caspersen (Ed.). ACM, New York, NY, 25–28. https://doi.org/10.1145/2532748.2532765

[5] Daniel Bildhauer and Jürgen Ebert. 2008. Querying Software Abstraction Graphs. In *Working Session on Query Technologies and Applications for Program Comprehension (QTAPC 2008)*. Amsterdam, Netherlands.

[6] Allan Birnbaum. 1968. Some latent trait models and their use in inferring an examinee's ability. In *Statistical theories of mental test scores*, Frederic M. Lord, Melvin R. Novick, and Allan Birnbaum (Eds.). Addision-Wesley, Reading, MA, 395–479.

[7] J. S. Bruner, R. R. Olver, P. M. Greenfield, and et al. 1966. *Studies in cognitive growth*. Wiley, Boston, MA.

[8] Thomas H. Cormen, Charles Eric Leiserson, Ronald L. Rivest, and Clifford Stein. 2003. *Introduction to algorithms* (2. ed., 4. printing. ed.). MIT Press, Cambridge and Mass.

[9] Lee Joseph Cronbach. 1951. Coefficient alpha and the internal structure of tests. *Psychometrika* 16, 3 (1951), 297–334.

[10] Jürgen Ebert and Angelika Franzke. 1994. A Declarative Approach to Graph Based Modeling. In *Graph-Theoretic Concepts in Computer Science*, Vol. 903. Springer Berlin Heidelberg, Berlin, Heidelberg, 38–50. https://doi.org/10.1007/3-540-59071-4$\delimiter"026E30F$textunderscore

[11] Joakim Ekström. 2011. The Phi-coefficient, the Tetrachoric Correlation Coefficient, and the Pearson-Yule Debate. https://escholarship.org/uc/item/7qp4604r

[12] Ursula Fuller, Colin G. Johnson, Tuukka Ahoniemi, Diana Cukierman, Isidoro Hernán-Losada, Jana Jackova, Essi Lahtinen, Tracy L. Lewis, Donna McGee Thompson, Charles Riedesel, and Errol Thompson. 2007. Developing a Computer Science-specific Learning Taxonomy. *SIGCSE Bull* 39, 4 (2007), 152–170. https://doi.org/10.1145/1345375.1345438

[13] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1993. Design Patterns: Abstraction and Reuse of Object-Oriented Design. In *ECOOP' 93 — Object-Oriented Programming*, Oscar M. Nierstrasz (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 406–431.

[14] Cees A.W. Glas and Norman D. Verhelst. 1995. Testing the Rasch Model. In *Rasch Models*, Gerhard H. Fischer and Ivo W. Molenaar (Eds.). Springer, New York, 69–95.

[15] Michael Goedicke and Michael Striewe. 2017. 10 Jahre automatische Bewertung von Programmieraufgaben mit JACK – Rückblick und Ausblick. In *7,5. HDI-Workshop des GI-Fachbereichs Informatik und Ausbildung / Didaktik der Informatik*. https://doi.org/10.18420/in2017_21

[16] Sumit Gulwani, Ivan Radicek, and Florian Zuleger. 2016. Automated Clustering and Program Repair for Introductory Programming Assignments. *CoRR* abs/1603.03165 (2016). http://arxiv.org/abs/1603.03165

[17] Johannes Hartig. 2008. Psychometric Models for the Assessment of Competencies. In *Assessment of competencies in educational contexts*, Johannes Hartig, Eckhard Klieme, and Detlev Leutner (Eds.). Hogrefe & Huber Publishers, Toronto, 69–90.

[18] Johannes Hartig, Eckhard Klieme, and Detlev Leutner (Eds.). 2008. *Assessment of competencies in educational contexts*. Hogrefe & Huber Publishers, Toronto.

[19] Bastiaan Heeren and Johan Jeuring. 2014. Feedback Services for stepwise exercises. *Science of Computer Programming* (2014).

[20] Manfred Kamp. 1998. GReQL - eine Anfragesprache für das GUPRO-Repository. In *GUPRO – Generische Umgebung zum Programmverstehen*. Koblenz, 173–202.

[21] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education (TOCE)* 19, 1 (2018), 1–43. https://doi.org/10.1145/3231711

[22] Eckhard Klieme, Hermann Avenarius, Werner Blum, Peter Döbrich, Hans Gruber, Manfred Prenzel, Kristina Reiss, Kurt Riquarts, Jürgen Rost, Heinz-Elmar Tenorth, and Helmut J. Vollmer. 2004. *The Development of National Educational Standards: An Expertise*. Bundesministerium für BildungundForschung, Berlin.

[23] Eckhard Klieme, Johannes Hartig, and Dominique Rauch. 2008. The Concept of Competence in Educational Contexts. In *Assessment of competencies in educational contexts*, Johannes Hartig, Eckhard Klieme, and Detlev Leutner (Eds.). Hogrefe & Huber Publishers, Toronto, 3–22.

[24] Eckhard Klieme and Detlev Leutner. 2006. Kompetenzmodelle zur Erfassung individueller Lernergebnisse und zur Bilanzierung von Bildungsprozessen: Überarbeitete Fassung des Antrags an die DFG auf Einrichtung eines Schwerpunktprogramms. http://kompetenzmodelle.dipf.de/de/resolveuid/0ac39a9248b26a284565ccb03d4b887f

[25] Adrian Andreas Kögl, Peter Hubwieser, Mike Talbot, Johannes Krugel, Michael Striewe, and Michael Goedicke. 2022. Efficient Structural Analysis of Source Code for Large Scale Applications in Education. In *2022 IEEE Global Engineering Education Conference, EDUCON 2022*. IEEE.

[26] Ingrid Koller and Reinhold Hatzinger. 2013. Nonparametric tests for the Rasch model: explanation, development, and application of quasi-exact tests for small samples. *InterStat* 11 (2013), 1–16. http://interstat.statjournals.net/YEAR/2013/articles/1311002.pdf

[27] M. Kramer, P. Hubwieser, and T. Brinda. 2016. A Competency Structure Model of Object-Oriented Programming. In *International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*. IEEE Xplore Digital Library, 1–8. https://doi.org/10.1109/LaTiCE.2016.24

[28] Johannes Krugel and Peter Hubwieser. 2017. Computational thinking as springboard for learning object-oriented programming in an interactive MOOC. In *2017 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 1709–1712. https://doi.org/10.1109/EDUCON.2017.7943079

[29] Johannes Krugel and Peter Hubwieser. 2020. *Web-Based Learning in Computer Science: Insights into Progress and Problems of Learners in MOOCs*. Springer Singapore, Singapore, 51–79. https://doi.org/10.1007/978-981-15-6747-6_4

[30] Detlev Leutner, Johannes Hartig, and Nina Jude. 2008. Measuring Competencies: lntroduction to Concepts and Questions of Assessment in Education. In *Assessment of competencies in educational contexts*, Johannes Hartig, Eckhard Klieme, and Detlev Leutner (Eds.). Hogrefe & Huber Publishers, Toronto, 177–192.

[31] Patrick Mair and Reinhold Hatzinger. 2007. CML based estimation of extended Rasch models with the eRm package in R. *Psychology Science* 49, 1 (2007), 26–43.

[32] Patrick Mair and Reinhold Hatzinger. 2007. Extended Rasch Modeling: The eRm Package for the Application of IRT Models in R. *Journal of Statistical Software* 20, 9 (2007), 1–20. https://doi.org/10.18637/jss.v020.i09

[33] Patrick Mair, Reinhold Hatzinger, Marco J. Maier, Thomas Rusch, and Maintainer Patrick Mair. 2020. Package 'eRm'. https://cran.r-project.org/web/packages/eRm/eRm.pdf

[34] Patrick Mair, Reinhold Hatzinger, and Marco J. Maier. 2009. Extended Rasch Modeling: The R Package eRm. https://cran.r-project.org/web/packages/eRm/vignettes/eRm.pdf

[35] Per Martin-Löf. 1974. Exact tests, confidence regions and estimates. In *Proceedings of Conference on Foundational Questions in Statistical Inference*. Univ. Aarhus, Aarhus, 121–138.

[36] Ivo W. Molenaar. 1995. Estimation of Item Parameters. In *Rasch Models*, Gerhard H. Fischer and Ivo W. Molenaar (Eds.). Springer, New York, 39–51.

[37] Andreas Mühling, Peter Hubwieser, and Marc Berges. 2015. Dimensions of Programming Knowledge. In *Informatics in Schools. Curricula, Competences, and Competitions*, Andrej Brodnik and Jan Vahrenhold (Eds.). Lecture notes in computer science, Vol. 9378. Springer International Publishing, 32–44. https://doi.org/10.1007/978-3-319-25396-1{_}4

[38] Jonas Neugebauer, Peter Hubwieser, Johannes Magenheim, Laura Ohrndorf, Niclas Schaper, and Sigrid Schubert. 2014. Measuring Student Competences in German Upper Secondary Computer Science Education. In *Informatics in Schools.*, Yasemin Gülbahar and Erinc Karatas (Eds.). Springer, Heidelberg, New York u.a, 100–111.

[39] Jonas Neugebauer, Johannes Magenheim, Laura Ohrndorf, Niclas Schaper, and Sigrid Schubert. 2015. Defining Proficiency Levels of High School Students in Computer Science by an Empirical Task Analysis Results of the MoKoM Project: Informatics in Schools. Curricula, Competences, and Competitions: 8th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2015, Ljubljana, Slovenia, September 28 - October 1, 2015, Proceedings. Springer International Publishing, Cham, 45–56. https://doi.org/10.1007/978-3-319-25396-1{_}5

[40] Elizabeth Odekirk-Hash and Joseph L. Zachary. 2001. Automated Feedback on Programs Means Students Need Less Help from Teachers. In *Proceedings of the Thirty-second SIGCSE Technical Symposium on Computer Science Education* (Charlotte, North Carolina, USA) *(SIGCSE '01)*. ACM, New York, NY, USA, 55–59. https://doi.org/10.1145/364447.364537

[41] OECD. 2013. *PISA 2012 assessment and analytical framework: Mathematics, reading, science, problem solving and financial literacy*. OECD, Paris.

[42] OECD (Ed.). 2013. *PISA 2012 Mathematics Framework: Mathematics, Reading, Science, Problem Solving And Financial Literacy*. OECD Publishing, Paris. https://doi.org/10.1787/9789264190511-en

[43] Ivo Ponocny. 2001. Nonparametric goodness-of-fit tests for the rasch model. *Psychometrika* 66, 3 (2001), 437–460. https://doi.org/10.1007/BF02294444

[44] Lutz Prechelt and Guido Malpohl. 2003. Finding Plagiarisms among a Set of Programs with JPlag. *Journal of Universal Computer Science* 8 (2003), 1016–1038.

[45] Georg Rasch. 1960. *Probabilistic models for some intelligence and attainment tests*. Studies in mathematical psychology, Vol. 1. Danmarks pædagogiske Institut, Copenhagen.

[46] Tenko Raykov and George A. Marcoulides. 2011. *Introduction to psychometric theory*. Routledge, New York.

[47] Dimitris Rizopoulos. 2006. ltm: An R Package for Latent Variable Modeling and Item Response Analysis. *Journal of Statistical Software* 17, 5 (2006), 1–25. http://www.jstatsoft.org/v17/i05

[48] Jürgen Rost. 2001. The Growing Family of Rasch Models. In *Essays on Item Response Theory*, Anne Boomsma, Marijtje A. J. Duijn, and Tom A. B. Snijders (Eds.). Springer, New York, 25–42.
[49] Winston W. Royce. 1970. Managing the Development of Large Software Systems. In *Proceedings IEEE WESCON*. 1–9.
[50] Alexander Ruf, Marc Berges, and Peter Hubwieser. 2015. Classification of Programming Tasks According to Required Skills and Knowledge Representation. In *Informatics in Schools. Curricula, Competences, and Competitions*, Andrej Brodnik and Jan Vahrenhold (Eds.). Lecture notes in computer science, Vol. 9378. Springer International Publishing, 57–68. https://doi.org/10.1007/978-3-319-25396-1{_}6
[51] F. Schott and S. Azizi Ghanbari. 2009. Modellierung, Vermittlung und Diagnostik der Kompetenz kompetenzorientiert zu unterrichten - wissenschaftliche Herausforderung und ein praktischer Lösungsversuch. *Lehrerbildung auf dem Prüfstand* 2, 1 (2009), 10–27.
[52] Robert Sedgewick and Kevin Wayne. 2011. *Algorithms* (4. ed. ed.). Addison-Wesley, Upper Saddle River, NJ.
[53] Tina Seidel and Manfred Prenzel. 2008. Assessment in Large-Scale Studies. In *Assessment of competencies in educational contexts*, Johannes Hartig, Eckhard Klieme, and Detlev Leutner (Eds.). Hogrefe & Huber Publishers, Toronto, 279–304.
[54] Juha Sorva. 2013. Notional machines and introductory programming education. *ACM Transactions on Computing Education* 13, 2 (2013), 1–31. https://doi.org/10.1145/2483710.2483713
[55] Draylson M. Souza, Katie R. Felizardo, and Ellen F. Barbosa. 2016. A Systematic Literature Review of Assessment Tools for Programming Assignments. In *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*. 147–156. https://doi.org/10.1109/CSEET.2016.48
[56] Michael Striewe and Michael Goedicke. 2013. Trace Alignment for Automated Tutoring. In *Proceedings of International Computer Assisted Assessment (CAA) Conference 2013*. Southampton.
[57] Robert F. Tate. 1954. Correlation Between a Discrete and a Continuous Variable. Point-Biserial Correlation. *Ann. Math. Statist.* 25, 3 (1954), 603–607. https://doi.org/10.1214/aoms/1177728730
[58] Jürgen Uhl and Hans Albrecht Schmid. 1990. *A systematic catalogue of reusable abstract data types*. Lecture notes in computer science, Vol. 460. Springer, Berlin. https://doi.org/10.1007/BFb0016877
[59] Anne Venables and Liz Haywood. 2003. Programming students NEED instant feedback!. In *Proceedings of the fifth Australasian conference on Computing education - Volume 20* (Adelaide, Australia) *(ACE '03)*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 267–272. http://dl.acm.org/citation.cfm?id=858403.858436
[60] Norman D. Verhelst and Cees A.W. Glas. 1995. Dynamic Generalizations of the Rasch Model. In *Rasch Models*, Gerhard H. Fischer and Ivo W. Molenaar (Eds.). Springer, New York, 181–201.
[61] Franz E. Weinert. 2001. Concept of Competence: A conceptual clarification. In *Defining and Selecting Key Competencies*, Dominique Simone Rychen and Laura Salganik (Eds.). Hogrefe and Huber, Seattle, 45–65.
[62] L. E. Winslow. 1996. Programming pedagogy - a psychological overview. *ACM SIGCSE Bulletin* 28, 3 (1996), 17–22.

A APPENDIX

A.1 Original worksheet text of Miniproject 3 in 2019/20

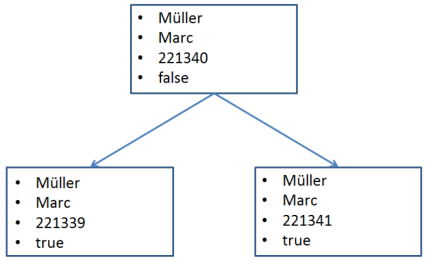
(Translated by the authors from the original language German.)

There are two classes in this miniproject, Phonebook and Person. The class Person declares a string lastName for the person's last name, a string firstName for the person's first name, an integer number for the person's phone number, and a boolean married that indicates whether the person is married. It also references a field leftSuccessor and a field rightSuccessor of type Person. The Phonebook class declares a field firstEntry of type Person to represent the root of a binary tree. All elements in the tree are of type Person. The Phonebook class also provides a main method.

As in a real phone book, the entries in our tree should always be sorted as follows:

1) The last name of the person
2) If the last name is equal: first name of the person
3) If the first and last name is equal: the person's telephone number.

In a binary tree this means: The left successor of an entry is always smaller in the sense of the sorting criterion than the parent node and the right successor is always larger in the sense of the sorting criterion than the parent node. A minimal example is displayed in the following picture:



The entry for Marc Müller with the phone number 221339 is the leftSuccessor of the entry with the phone number 221340 and the entry with the phone number 221341 is the rightSuccessor of the entry with the phone number 221340. The existing class, variable and method names must not be changed! On the other hand it is allowed to add further methods if necessary or to change the content of the main method for own tests. For all tasks, two methods are given, one declared public and one private. The method declared public should call the method declared private, in which the recursion then takes place. You can customize the methods with the modifier private as you wish, and even change the return and parameterization if that brings you closer to the solution. You can assume that each person always has their own phone line, i.e. for each phone number there is at most one person to whom this phone number is assigned.
Please note that the submission of a (not necessarily correct) solution to the mini-project is required for the respective test set admission!

Task 1: Insertion of an Entry. Implement the method insertPerson(String lastName, String firstName, int number), which creates a person with the given parameters and sorts it into the tree in the right place. You can use the method compareTwoPersons(Person1, Person2) to compare two persons in the sense of the sorting criterion.
Task 2: Find an Entry. Implement the method findPerson(String lastName, String firstName, int number) to decide whether a person with the given parameters can be found in the phone book.
Task 3: Counting all entries. Implement a function count() that returns the number of all phone book entries.
Task 4: Searching for persons. Implement the method findAllPersonsByFirstName(String firstName) in the class PhoneBook, which creates an array that includes all persons with the given first name. The array should also be sorted according to the sorting criteria of the phone book, i.e. the person "Marc Müller" with the phone number "221339" must be located in the array before of the person "Marc Müller" with the phone number "221340". If there are no entries for the given first name, an empty array should be returned. Thus the method should never return zero.
Task 5: Change of an Entry. Due to a change in the legal regulations, all persons with the surname "Wedding" must actually be married. Implement the method marryTheWedding() to set the value of married to true for all people with the last name "Wedding".
Task 6: Deletion of Entries. Of course, it must also be possible to delete entries from the phone book. Implement the method removePersonFromPhoneBook(String lastName, String firstName, int number), which performs just this operation. Please note that when you delete an entry, its right and left successors should remain in the tree and in addition, the tree must remain sorted correctly.
Task 7: Change Entries and Sort Again. When two persons get married, sometimes at least one of them changes his/her last name. Implement the method changePerson(String lastName, String firstName, int number, String newLastName), which renames the person with the given parameters correctly and keep the tree sorted correctly. Since the person has changed his/her name by marriage, this should also be read in their attribute married, i.e. this attribute should have the value true.