# Automated Measurement of Competencies and Generation of Feedback in Object-Oriented Programming Courses

Johannes Krugel*, Peter Hubwieser*, Michael Goedicke†, Michael Striewe†, Mike Talbot*,
Christoph Olbricht†, Melanie Schypula†, and Simon Zettler*
*School of Education, Technical University of Munich, Munich, Germany
Email: krugel@tum.de, peter.hubwieser@tum.de, mike.talbot@tum.de, simon.zettler@tum.de
†Paluno – The Ruhr Institute for Software Technology, University of Duisburg-Essen, Essen, Germany
Email: {michael.goedicke, michael.striewe, christoph.olbricht, melanie.schypula}@paluno.uni-due.de

***Abstract*—To overcome the shortage of computer specialists, there is an increased need for correspondent study and training offers, in particular for learning programming. The automated assessment of solutions to programming tasks could relieve teachers of time-consuming corrections and provide individual feedback even in online courses without any personal teacher. The e-assessment system JACK has been successfully applied for more than 12 years up to now, e.g., in a CS1 lecture. However, there are only few solid research results on competencies and competence models for object-oriented programming (OOP), which could be used as a foundation for high-quality feedback.**

**In a joint research project of research groups at two universities, we aim to empirically define competencies for OOP using a mixed-methods approach. In a first step, we performed a qualitative content analysis of source code (sample solutions and students' solutions) and as a result identified a set of suitable competency components that forms the core of further investigations. Semi-structured interviews with learners will be used to identify difficulties and misconceptions of the learners and to adapt the set of competency components. Based on that we will use Item Response Theory (IRT) to develop an automatically evaluable test instrument for the implementation of abstract data types. We will further develop empirically founded and competency-based feedback that can be used in e-assessment systems and MOOCs.**

***Index Terms*—computer science education, object oriented programming, educational technology, electronic learning**

## I. Introduction

To overcome the current shortage of computer specialists, there is an increased need for correspondent study and training offers, in particular for learning programming. An analogous lack of sufficiently competent teaching staff, however, limits the supply of face-to-face courses. The automated assessment of solutions to programming tasks could relieve teachers of time-consuming corrections in order to increase the capacities of classroom teaching. At the same time, it can provide complementary online services without any human tutoring, in particular Massive Open Online Courses (MOOCs).

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 412374068

The working group "Specification of Software Systems" at University of Duisburg-Essen has developed and maintained the e-assessment system JACK for more than 12 years up to now [1]. It has been successfully applied for automated assessment e. g. in the context of the CS1 lecture given by the working group at that university. During the first 12 years of service, approximately 93,000 solutions of attestations as well as approximately 133,000 solutions of preparatory practice tasks were automatically graded and provided with feedback.

In order to support the intended learning processes, feedback should address competency deficits, rather than simply list failed test cases or provide correct solutions. However, computer science education research has so far lacked solid research results on competencies and competence models for programming, which could be used as a foundation for the development of such high-quality feedback. Furthermore, the research on errors and misconceptions in programming is rather inconsistent, incomplete and hardly illuminated from the competency-oriented point of view.

The research group for computer science education at the Technical University of Munich is conducting intensive research on these fields. In addition, it has created a MOOC called LOOP for learning object-oriented programming on the edX platform [2, 3], which was attended by more than 7000 students up to now. The JACK system mentioned above was already used in the MOOC for real-time evaluation and feedback generation on the students' solutions of exemplary programming tasks.

Since 2015, both research groups combine their experience and expertise, aiming to link fundamental research on e-assessment with the extensive research on computer science competencies in order to address the research gaps mentioned above. Our collaboration has already produced relevant results. We were able to identify several potential competencies that are required to implement and operate 1- and 2- dim arrays, linked lists, and binary trees (see [4]). In recognition of these preliminary results, the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) has agreed to fund our collaboration project AKoFOOP, which links areas

of research in computer science and its didactics. The project started in April 2019 and will be terminated in March 2021. E-assessment systems enable surveys with large case numbers, which are necessary for our competency analyses based on the Item-Response Theory (see e.g. [5]). Our project aims to achieve the following main goals, using and improving the JACK system:

1) Empirically founded definition and structural analysis of competencies for OOP, in particular in the field of abstract data types (e.g. array, list, tree, graph),
2) Development and piloting of an automatically evaluable test instrument for the implementation of abstract data types,
3) Exploring the possibilities and impact of competency-based feedback in e-assessment systems and MOOCs.

In this paper, we first describe the theoretical background and related literature in this field (Section II). To give the context for our research, we describe the course under investigation and the data collection using the e-assessment system JACK (Section III). We present our mixed-methods-based approach and in particular the qualitative content analysis of source code that we used to identify possible components of competencies (Section V). We also describe the further project plan including the analysis of students solutions, semi-structured interviews, the measurement of possible competencies using Item Response Theory (IRT) and the adaption of the automated feedback (Section VI).

## II. Theoretical Background and Related Work

We start by defining competencies and give an overview of competence models in computer science. Afterwards we discuss related literature on errors and misconceptions for OOP and the automated assessment of programming solutions. We furthermore describe the IRT which forms the basis for the measurement of competencies.

### A. Competencies

In the context of this project, we denote by *competency* a context-specific cognitive performance disposition in the sense of Weinert [6, p. 27], who defined 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 the motivational, volitional and social readiness and capacity to use the solutions successfully and responsibly in variable situations.*"

To define competencies in our project, we make use of Schott and Azizi Ghanbari [7, p. 15]: "*A competence consists of certain amounts of tasks that can be carried out if you have the competence.*" In this sense, we can consider certain sets of "suitable" tasks as representations of a specific competency. Of course, there are certain conditions 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.

Within the framework of this project we are particularly interested in the internal structure of such competencies. The partial capabilities whose mastery is necessary, but not sufficient for having the competency, will be called *competency components* in the following (similar to *item demands* as described by Hartig, Klieme, and Leutner [8]).

### B. Competence Models for Computer Science

Instead of an orientation at the term *competency*, the works of the international computer science education communities are still dominated by the terms *knowledge* and *skills*. Correspondingly, 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 [9, p. 6-7] are of particular importance.

Some research in this direction has been carried out within the framework of the MoKoM project. This resulted in a first empirically and theoretically based competence structure model for two sub-areas of computer science (modelling and system comprehension) [10] as well as a test instrument for them. It turned out, however, that the dimensionality of the measured competencies was not sufficiently clear [11]. Above all, the model turned out to be far too coarsely granular in the area of OOP, which is relevant for us. The MoKoM model contains only a few competencies in this respect. For this reason, there are currently no sufficiently detailed and empirically founded competence structure models specifically for recording and modelling programming competencies, even though individual further attempts have already been made to do so, e. g. [12, 13].

Based on the above competency definition, Mühling, Hubwieser, and Berges [14] examined the potential cognitive facets by analyzing the structures from declarative knowledge to OOP through surveys of concept maps [14]. The similarities in the knowledge structures of first-year students with very different computer science education resulted in the subject-specific sub-dimensions of a first proposal for a competence model for OOP, which was developed on the basis of an extensive literature study, also on other subjects, as well as various additional qualitative data [15, 16]. The result are described in [17]; it contains among others the following sub-dimensions which are particularly relevant for our project:

1. OOP knowledge and skills:
   - 1.1 Data structure (graph, tree, array),
   - 1.2 Class & object structure (object, attribute, association),
   - 1.3 Algorithmic structure (loops, conditional statement).

### C. Errors and Misconceptions

We expect further indications of the internal structure of our competencies by the analysis of the errors or flaws found in the student solutions of the tasks. These deficits might be caused by competency deficits, individual misconceptions as well as by general learning barriers.

There are a number of publications on the study of errors and misconceptions in programming, see for example [18, 19]. Robins et al. also provide a literature overview of learning and teaching programming [20]. They also looked into the question of what specific deficits can be identified in ineffective programming beginners. Ragonis and Ben-Ari [21] performed a mixed-methods analysis and identified a set of misconceptions and difficulties when learning object-oriented programming. Sirkiä observes new programmers looking for hints which programming concepts are misunderstood by learners and lead to problems when learning a programming language. He summarized these concepts into misconceptions and divided them into five categories (see also [22], [23, p. 11]). Vahrenhold and Paul [24] develop test items for OOP focusing specifically on algorithms and data structures. Caceffo et al. [25, 26] recently developed a concept inventory for OOP including an overview of misconceptions in Java.

Zehetmeier et al. [27] made a first connection between misconceptions and competencies, proposing a classification of the relevant cognitive processes on the basis of the taxonomy of Anderson & Krathwohl [28]. Each error class was assigned to missing "competencies". However, the error categories are very general and the competencies are not sufficiently theoretically founded, for example by classification into a competence structure model.

Programming errors can also point to learning barriers, which the Danish learning theorist Knud Illeris divided into the areas of *fail learning*, *defence against learning*, *ambivalence* and *resistance against learning* [29]. We will also use this taxonomy as a starting point for our categorization.

### D. Automated Assessment and Feedback

Due to high numbers of participants in introductory courses, many universities use e-assessment systems for formative or summative assessment. For the subject of programming exercises, a wide range of e-assessment systems exists that detect errors by executing test cases and analyzing source code [30, 31]. In general, automated systems perform similarly well compared to human graders regarding scoring and the generation of feedback [32]. However, a lot of research is related to direct error feedback with ambivalent results: On the one hand, fine-grained feedback and grading schemes have been proven to be effective [33], but on the other hand detailed syntax error messages appear to be ineffectual [34]. There is also research on additional aspects like assessing code quality and style [35, 36] or estimating difficulty of exercises through the analysis of large amounts of solution data [37], while explicit research on competency measurement for programming through automated assessment is rare.

### E. Measurement of Competencies / Item Response Theory

According to Classical Test Theory (CTT), the construct of interest (e.g. student abilities) is considered to be measured directly by item scores, although this is 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, the Item Response Theory (IRT) treats the constructs of interest as latent psychometric constructs that cannot be measured directly. Yet, the probability $X_{i,k}$ of a correct answer of person $i$ for item $k$ depends on those constructs in a certain way:

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

where $\Theta_i$ is the parameter of person $i$, representing the manifestation of the psychometric construct, $\beta_k$ the parameter of item $k$, representing its difficulty, and $f(\Theta_i, \beta_k)$ a function that is determined by the psychometric model (e.g. the Rasch Model (RM), see below) that is assumed to fit the observations. In most cases these parameters have to be estimated by numeric calculations. Depending on the assumptions about the structure of the psychometric constructs that are to be measured, several different models may be considered, e.g. unidimensional models that cover only one single latent variable or alternatively multidimensional models. One of the simplest and most widely used ones is the basic unidimensional (monofactorial) RM with one parameter (1F1P) :

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

The graph of such a function called Item Characteristic Curve (ICC) and S-shaped (see below in Figure 1). 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. Yet, this model is applicable only if the ICCs have (at least nearly) the same slope. This slope is represented by an additional discrimination parameter $\delta_k$ in the 2-parametric RM (1F2P):

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

In the case that there is more than one psychometric construct to be measured, the multidimensional RM must be used [17]. In this case the probability of getting item $i$ right is determined by the sum of all abilities that are required for this task [16]. In all those cases, three general preconditions have to be met for the application of the RMs:

1) Homogeneity of items: all items are measuring the same psychometric construct.
2) Local stochastic independence: the psychometric construct is the only coupling factor between items.
3) Specific objectivity: for all samples from the population, the item parameters are independent of the specific sample; the same holds for all samples of questions and person parameters.

## III. Course Context and Data Collection

For the bachelor degree programs "Applied Computer Science" and "Business Informatics" at the University of Duisburg-Essen, the course "Computer Science 1" (CS1) is mandatory in the first term. In CS1 the students acquire the knowledge and skill of basic programming. The lecturer and the teaching staff offer a wide range of learning opportunities for the students including the lectures, recitation sessions, consultation hours, exercises, and mini-projects. Small exams (so called *attestations*) are held every two weeks. The following subsections describe those course elements, the differences between the course runs and the data extraction.

### A. Content and Organization of the Course

The course includes two lectures and one recitation session held by the lecturer or graduate teaching assistants per week. Each lecture and recitation session takes 90 minutes and the term has 14 weeks. The topics covered in CS1 are:

- Weeks 1 & 2: "Java crash course" enabling students to write first small Java programs without digging into conceptual details
- Weeks 3 & 4: Classes, objects, attributes, variables, primitive data types and control structures
- Weeks 5 & 6: Arrays and lists
- Weeks 7 & 8: Trees and recursion
- Weeks 9 & 10: Inheritance and interfaces
- Weeks 11 & 12: Generics and exceptions
- Weeks 13 & 14: Lambda expressions and graphs

An optional 2 hours consultation session every weekday is offered by undergraduate teaching assistants to support the students on programming with Java. The consultation session is held in a computer pool, where terminals are provided for the students to work on. The students may ask questions about every aspect of the course. In case a question cannot be answered by the undergraduate teaching assistants, they inform a graduate teaching assistant, who will help to solve the issue.

Throughout the term, 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 to 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, a larger exercise called mini-project (MP) is provided every two weeks in JACK. The mini-project is a mandatory assignment the students must solve in about a week's time. Each mini-project addresses a specific subject area and is split into five to ten subtasks of increasing difficulty. The mini-project aims to supply a comprehensive overview of the subject area. It offers the students a way to work on a coherent set of tasks, with the intent to provide a project-like environment contrary to the small and isolated exercises mentioned above. The students can submit any number of solutions to be evaluated by JACK. The system provides extensive feedback, so that the students can improve their solution with each submission. In general, each mini-project is designed to prepare the students for the corresponding following 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.

Finally, attestations (AT) are held every two weeks during the term, for a total of five attestations. These attestations are small admission exams, in which students can earn a maximum of 100 points each. Students are only allowed to take the end of term exam, if they reach a total score of at least 330 points in the attestations. In an attestation, students solve a given task within 45 minutes in a computer pool under exam conditions. Students must use Eclipse as an integrated development environment (IDE) and have no access to the internet. The

TABLE I
Mini-project (MP) and attestation (AT) tasks

| **Task & subject** | **Subtasks** |
|---|---|
| MP0: training | Implement simple methods with simple datatypes |
| AT0: training | Implement simple methods with simple datatypes |
| MP1: one- and two-dimensional arrays | Implement constructor, add single value to array, add array to existing array, get the minimum value in array, get all values above threshold in array, scalar multiplication of matrix, add two matrices, get column of matrix, transpose matrix |
| AT1: one- and two-dimensional arrays | Get maximum value in array, subtract two matrices |
| MP2: singly linked lists | Implement constructor, append element, get element at position, count elements, sum up values of attribute over all elements, change values of attribute in a specific way over all elements, insert element at position, remove element from list, split list at position, reverse order of list elements |
| AT2: singly linked lists | Mark specific elements in list, remove marked elements from list |
| MP3: recursion over tree | Insert element, find element, count all elements, find all elements with attribute value and save to array, change element without changing order, delete element, change element and update tree accordingly |
| AT3: recursion over tree | Insert element in binary tree, return array with tree elements in preorder |
| MP4: inheritance and interfaces | Inherit from class, implement constructor, overload method, override method |
| AT4: inheritance and interfaces | Inherit from class, implement method in parent, override method in child |
| MP5: generic list and map with custom comparator | Implement constructor to initialize calendar object, iterate over generic map, insert and get values from map, implement custom comparator, sort generic list with created comparator, insert element, insert list of elements, get all elements, get specific elements, get specific elements in order and count elements in generic list |
| AT5: generic list and map with custom comparator | Add element to generic list, add element at position to generic list, add values to map, search for key in map, replace value of all key-value-pairs in map |

TABLE II
NUMBER OF STUDENTS AND SUBMISSIONS PER TERM FOR MINI-PROJECTS

| | Number of students | | | | Number of submissions | | | |
|---|---|---|---|---|---|---|---|---|
| **No. and variant** | ***2015/16*** | ***2017/18*** | ***2018/19*** | ***Total*** | ***2015/16*** | ***2017/18*** | ***2018/19*** | ***Total*** |
| MP0 (variant 1) | 669 | | | 669 | 2413 | | | 2413 |
| MP0 (variant 2) | | 477 | | 477 | | 921 | | 921 |
| MP0 (variant 3) | | | 479 | 479 | | | 874 | 874 |
| MP1 | 580 | 663 | 662 | 1905 | 2362 | 2142 | 2235 | 6739 |
| MP2 | 483 | 579 | 607 | 1669 | 3643 | 3105 | 4655 | 11403 |
| MP3 (old version) | 399 | 434 | | 833 | 1630 | 1794 | | 3424 |
| MP3 (new version) | | | 496 | 496 | | | 3756 | 3756 |
| MP4 | 316 | 334 | 381 | 1031 | 1090 | 955 | 1482 | 3527 |
| MP5 | 260 | 248 | 311 | 819 | 1005 | 1070 | 1463 | 3538 |

students may submit several solutions and the number of points is calculated from the best submitted solution. An additional attestation that does not count for the total score is offered early in the term so that students can make themselves familiar with these special conditions. Since the largest computer pool of the university can only be used by about 200 students at the same time, up to four attestations sessions are offered and the attestation tasks vary slightly between these sessions to avoid exam fraud. The tasks are similar in difficulty and workload to single subtasks of the corresponding mini-project. A comprehensive overview for each subtask of each mini-project and the corresponding attestation tasks is presented in Table I. Submitted solutions will be evaluated by JACK after the attestation is finished. Scores and feedback are available to the students in the following week. Students may review their submitted solutions, scores and feedback under the supervision of an undergraduate teaching assistant in any consultation session. Attestations and mini-projects are a way of ensuring that students have to work continually on their programming skills.

### B. Differences Between the Course Runs

Relevant for our project are the course runs in winter terms 2015/16, 2017/18, and 2018/19 as well as future course runs until 2020/21. We ignore the run in winter term 2016/17, since the course was held by a different lecturer, who made large deviations from the course structure and attestation topics discussed above. Besides that, there were also some other notable changes between the three terms and there will be others for future terms.

All attestation tasks were refurbished or replaced by new ones before winter term 2017/18 without changing the main topics of each attestation. In winter term 2018/19, the tasks for mini-project 3 were also replaced, because the previous one was perceived as too difficult for the students. Tasks for mini-project 0 (as a preparation to the additional, non-graded attestation mentioned above) changed every term.

For the future starting from winter term 2019/20, the "Java crash course" will be turned into a regular part of the lecture but cover less Java constructs and more conceptual details on variables, primitive data types, control structures and arrays. Week three and four can then be reserved for object-oriented concepts and lists. These changes will actually have no effect on the topics of mini-projects and attestations.

### C. Usage Figures

Table II provides an overview on the number of submitting student and the total number of submissions per term for the mini-projects. The figures for the submissions of the attestations are very similar. The drop-out of students throughout each term is within the usual margins for that course. It is due to the fact that students who already gathered enough points to be admitted for the final exam tend to waive participation in further attestations. Similarly, students who cannot gather enough points in the remaining attestations usually tend to quit the course for that term and make another try in the next term.

Notably, the average number of submissions per student varies from term to term (e. g. between 3.23 and 4.07 for MP1 or between 5.36 and 7.67 for MP2). There is no obvious reason for that, i. e. students did not receive different instructions on whether to make more or less submissions in different terms.

### D. Data Extraction and Preparation

In order to allow for easier processing and analysis within the project, all submissions to the JACK system have been exported to a unified file system structure. Every year and exercise (including mini-projects and attestations) is represented as a separate folder within that structure. Each of these folders contains separate folders for each student and in them are separate folders for each submission of that student. Hence, each individual solution can be addressed by a unique path as *"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.

As a first preprocessing step, so-called "sequencing tables" have been produced for each exercise. Each line in these tables represents one submission. Each column represents 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 feedback,

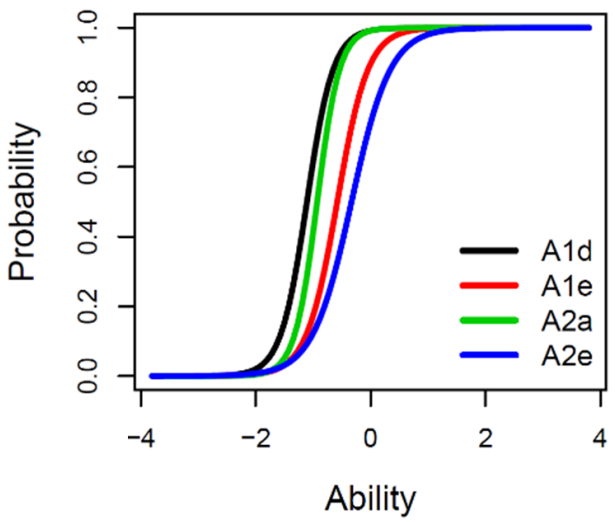


Fig. 1. Acceptable 2-parameter ICC of a 4-combination in MP1.

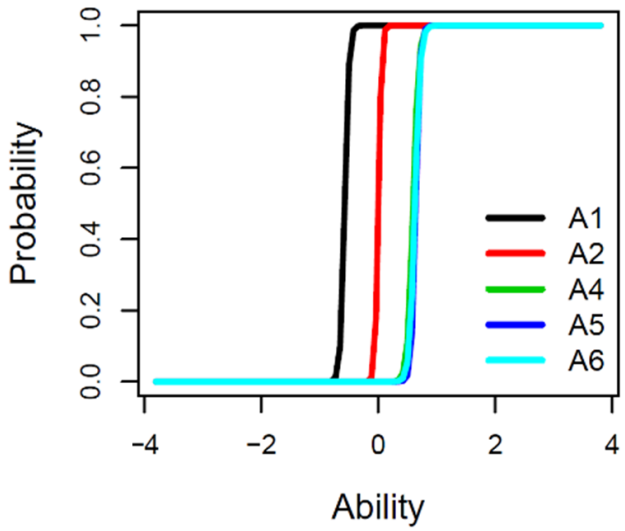


Fig. 2. 2-parameter ICC of nearly perfect item set in MP3.

the corresponding cell contains a ”0”. There is an ”n-to-one”-mapping from feedback messages to test cases and another ”n-to-one”-mapping from test cases to sub-tasks. Hence, one test case can fail with several different error messages and the correctness of a solution with respect to a sub-task can be checked by several test cases. The sequencing tables are produced automatically based on these mappings: The list of feedback for each submission is inspected and each feedback message is looked up in the mapping. The cells for the corresponding test cases and sub-tasks are set to ”0”, while all other cells for that submission are set to ”1”. Note that we do not use the score displayed to the students (a value in the range 0 to 100) in this study, as each exercise can have individual rules on how to calculate the score based on the outcome of the test cases.

## IV. Preliminary Results

In preparatory work for our project application [4], we had already analyzed the MPs in the course run of 2015/16 (the MPs in 2015/16 were numbered differently; we changed the numbering here to unify the exposition). We found several shorter item combinations that seemed to have acceptable variations in the discrimination parameters $\delta$ (see Figure 1).

The “best” item combination regarding its length, the variation of the discrimination parameter and the model fit was one of the two combinations of 5 items of MP3 (see Figure 2), called MP3-C5 in the following text. The only flaw of MP3-C5 was that three items A4, A5 and A6 have nearly the same item difficulty, causing a coincidence of their ICCs by this way. We had also calculated the item parameters of MP3-C5, resulting in the same difficulty ranking as produced by the classical item difficulty (portion of correct solutions). In addition, the resulting person parameters had turned out to follow a bimodal distribution.

## V. Methodology and Results

One of the main objectives of our project is the definition, delimitation and clarification of competencies for object-oriented programming. An important part is to identify potential competencies and their components. Since we make use of Schott and Azizi Ghanbari [7, p. 15] to define competencies through a set of homogeneous tasks that can be carried out when a student has the underlying competency, the link between competencies, their components and tasks (items) is essential to validate the competencies found through IRT.

Therefore we analyzed the structure of all tasks and sample programming solutions of the mini-projects and attestations from an expert’s perspective in order to determine the required skills from a subject-specific cognitive point of view. The analysis included only the skills required to program the version of the sample solution. In many cases there are alternative solutions (e.g. an iterative instead of a recursive solution or vice versa). The alternative solution paths are determined by analyzing the students’ solutions.

As methodology we used an inductive qualitative content analysis (QCA) based on Mayring [38] adapted to the analysis of source code errors as proposed by Shah [39]. We coded the programming solutions according to his guidelines (paraphrasing, generalization, categorizing, checks). There were 4 coders and all decisions taken were noted in a coding manual. The work was carried out in pairs and by individuals.

The paraphrasing step (similar to the methodology of Mayring [38]) eliminates non content-bearing code and transforms the remaining code into a unified form [39]; the program code of all the sample solutions is provided in a unified form. The generalization step [39] was done by transforming the abstract programming code into an abstract description in natural language to describe the main parts of the code’s solution approach. In the categorization step, we categorized the descriptions of the code according to the basic concepts,

describing the solution components or steps (e.g. *concatenate two arrays, change and re-sort elements in a binary tree*).

A central question of our research concerns the level of granularity at which relevant differences can be found among the students' solutions on certain tasks. At the most abstract level, almost all items of our tasks will measure more or less the same competencies, e. g. "*implement the abstract data type list with the most important operations*", while at the most detailed level, probably all students would master all requirements, e. g. "*implement basic algorithmic control structures in Java*". In between, however, there must be a level at which relevant differences can be found both in the item demands and in the individual programming skills. Our hypothesis is that these are at the level of specific patterns needed for the solutions, similar to design patterns in the field of software engineering (see e.g. [40]) or to computational thinking patterns (see [41]).

For completeness, the code of the first analyzed mini-project Java classes was categorized in very small parts (e.g. *class definition, variable declaration, conditional statement, ...*). We have decided to categorize these very detailed parts of the code, which are not complex enough to be an important part of a competency, only at their first appearance. In the following we limited ourselves to the identification and categorizing of programming patterns in solution components or solution steps with sufficient complexity.

From the analysis of the sample solutions of the mini-projects and the attestation we have derived a series of 46 potential competency components of reasonable complexity, which will form the core of all further investigations. Some examples are:

- *Concatenate two arrays*
- *Determine an extreme value of an array*
- *Filter array into new array*
- *Sort list with a comparator*
- *Filter elements of a list to another list*
- *Split a chained list by index*
- *Find elements in a binary tree*
- *Insert an element in a binary tree*
- *Change and re-sort elements in a binary tree*
- *Convert a binary tree to an array*
- *Calculate the average of all values in a set*
- *Add two two-dimensional arrays element by element*
- *Multiply a two-dimensional array with a scalar*
- *Define recursive methods*
- *Implement an interface*
- *Implement polymorphic methods*

To achieve reliability and objectivity Mayring [38], 15% of the total program code was inter-coded.

Since there are alternative solutions to the sample solutions we currently analyze student's solutions of the mini-projects and attestations in order to extend the category system of possible competencies and competency components to saturation. Furthermore, we examine the students' solutions for errors and misconceptions. The deficiencies in the code can then help to further structure and differentiate the competency candidates and, in particular, to improve the feedback of JACK. Errors and misconceptions are also categorized by a qualitative content analysis.

## VI. Further Project Plan

Following a mixed-methods approach, we are going to complement the results from different perspectives. In particular, we conduct interviews, develop questionnaires, and analyze protocols of the consultation sessions to refine and extend the set of potential competency components. The further project plan is described in the following paragraphs.

### A. Further data sources

To get insights into the approaches and difficulties of the students regarding the assignments, we developed guidelines for semi-structured interviews. The students are interviewed after each attestation and before the results of the attestation is published. The selection of students is random and no personal information is recorded. The students are asked to give detailed feedback about the most recent mini-project and the attestation. They grade the difficulty of both assignments and explain which subtasks were easy or difficult to solve. They supply information on the external sources they might have used to solve the problems and whether they worked with other students, or preferred to work alone. Finally, they evaluate the feedback given by JACK and how they prepared themselves for the attestation.

The answers will also serve as basis to develop an online questionnaire consisting mainly of closed questions. This questionnaire is going to be distributed in the following course run to all students of the course to get a representative picture of the participants. The results of the interviews will furthermore be used to extend our category system of competency components.

Another method to get insights into the learners' perspectives are protocols of the consultation sessions, which the undergraduate teaching assistants write for each session. The teaching assistants record the number of attending students, whether the students work in groups, the occurred topics with frequently asked questions and the average time they interacted with each student. These protocols are another promising source to evaluate key problems the students might have with each course topic. They can also help to adapt and extend the category system of potential competency components.

### B. Analysis Based on Item Response Theory

All results obtained at this point will be compiled together in order to obtain detailed information on combinations of competency components that can be regarded as candidates for empirically founded competencies.

The sequencing tables are then examined for combinations of columns that could be homogeneous in the psychometric sense, i.e. whose solution or application essentially depends exclusively on the development of a common competency (component). For this purpose, the following evaluations are performed with all column combinations: latent trait analysis

(LTA) according to [42], non-parametric analysis according to [43], Rasch tests according to [44], qualitative analysis of the item-characteristic curves (ICCs), multi Rasch analyses for potentially multi-factorial item sets and ANOVA with different psychometric models. The results of all these methods are then compared, also with classical test theory such as Cronbach-Alpha and point biserial correlations. This results in a selection of column combinations that could be relatively homogeneous. These combinations are then regarded as definitions of "competency candidates" in the sense of [7], whose subject-matter structure can in turn be derived from the competency components involved.

In the next step, the psychometric personal parameters for the column combinations of the candidate competencies are calculated for the students and their distribution are determined. The candidate competencies are validated by a comparison with the results of all previous results (interviews, questionnaires, consultation session protocols etc.). This comparisons is expected to result in a detailed description of the individual candidate competencies from several perspectives, including the corresponding item demands, learning barriers and solution patterns. On this basis, the candidate competencies will then be classified into the existing competence models [10, 17].

### C. Adaption of the Items

Depending on the results of the preceding research steps, the items used in the mini-projects and attestations will be modified or adapted. A change in the item difficulty could become necessary if the variance is too low. An adjustment of the item selectivity, i.e. the slope of the ICC in the Birnbaum model, can also become necessary if the variance is too low. An adjustment of the item selectivity (i.e. the slope of the ICC in the Birnbaum model) can also become necessary, e.g. if the ICCs within an item set overlap due to a high variance of the slopes (meaning the specific objectivity is violated). For all the mentioned adaptations, the knowledge gained up to this point, e.g. about problem areas, difficulty of the competency candidates or the frequency of certain solution patterns, will most likely prove to be very helpful. The result is a complete, ready-to-use set of items for all mini-projects and attestations, whose quality as a testing instrument from a psychometric point of view should increase from version to version.

### D. Development of Feedback

With these results, we strive to improve the feedback of JACK. Using the gained understanding of programming competencies, we can provide more support for the students. If we identify which missing concepts lead to a specific error by the students, we can inform them about the reason for this error. Instead of a simple error message or short hint what the error might be, we will be able to inform the student very specifically about the topics and concepts, which have to be revised in order to solve the given task. This might significantly improve the way a student is able to learn the skills needed to become a programmer.

## VII. Conclusion

From the extensive data of e-assessment systems, conclusions can be drawn about knowledge gaps and misconceptions, but also about competencies and their components. This data can be combined with further data sources (like interviews with the learners) to get a more complete picture.

The empirically-based definition of competencies further serves as basis for the generation of high-quality feedback.

The approach of defining and measuring competencies of our project is not limited to object-oriented programming, but could also be generalized to other topics and even disciplines.


## Acknowledgment

We thank Ronja Billenstein for her support with the qualitative content analysis.


## References


[1] M. Goedicke and M. Striewe. "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*. 2017. DOI: https://doi.org/10.18420/in2017_21.

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

[3] J. Krugel and P. Hubwieser. "Strictly Objects First: A Multipurpose Course on Computational Thinking". In: *Computational Thinking in the STEM Disciplines: Foundations and Research Highlights*. Ed. by M. S. Khine. Cham: Springer International Publishing, 2018, pp. 73–98. DOI: https://doi.org/10.1007/978-3-319-93566-9_5.

[4] P. Hubwieser, M. Striewe, M. Berges, and M. Goedicke. "Towards Competency Based Testing and Feedback". In: *Proceedings of IEEE Global Engineering Education Conference (EDUCON)*. 2017. DOI: https://doi.org/10.1109/EDUCON.2017.7942896.

[5] T. Raykov and G. A. Marcoulides. *Introduction to Psychometric Theory*. Routledge, 2011.

[6] F. E. Weinert, ed. *Leistungsmessungen in Schulen*. Weinheim und Basel: Belz Verlag, 2001.

[7] F. Schott and S. Azizi Ghanbari. "Modellierung, Vermittlung und Diagnostik der Kompetenz kompetenzorientiert zu unterrichten - wissenschaftliche Herausforderung und ein praktischer Lösungsversuch". In: *Lehrerbildung auf dem Prüfstand* 2.1 (2009), pp. 10–27.

[8] J. Hartig, E. Klieme, and D. Leutner, eds. *Assessment of competencies in educational contexts*. Toronto: Hogrefe & Huber Publishers, 2008. ISBN: 0889372977.

[9] E. Klieme and D. Leutner. *Kompetenzmodelle zur Erfassung individueller Lernergebnisse und zur Bilanzierung von Bildungsprozessen: Überarbeitete Fassung des Antrags an die DFG auf Einrichtung eines Schwerpunktprogramms*. Frankfurt a. Main, Duisburg, Essen, 2006.

[10] J. Neugebauer, J. Magenheim, L. Ohrndorf, N. Schaper, and S. Schubert. "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". In: Cham: Springer International Publishing, 2015, pp. 45–56. DOI: https://doi.org/10.1007/978-3-319-25396-1_5.

[11] J. Neugebauer, P. Hubwieser, J. Magenheim, L. Ohrndorf, N. Schaper, and S. Schubert. "Measuring Student Competences in German Upper Secondary Computer Science Education". In: *Informatics in Schools.* Ed. by Y. Gülbahar and E. Karatas. Heidelberg, New York: Springer, 2014, pp. 100–111. DOI: https://doi.org/10.1007/978-3-319-09958-3_10.

[12] J. Bennedsen and C. Schulte. "Object Interaction Competence Model v. 2.0". In: *Learning and Teaching in Computing and Enginering (LaTiCE)* (2013), pp. 9–16. DOI: https://doi.org/10.1109/LaTiCE.2013.43.

[13] K. Bröker, U. Kastens, and J. Magenheim. "Competences of Undergraduate Computer Science Students". In: *KEYCIT 2014 - Key Competencies in Informatics and ICT* 7 (2015), pp. 77–96.

[14] A. Mühling, P. Hubwieser, and M. Berges. "Dimensions of Programming Knowledge". In: *Informatics in Schools. Curricula, Competences, and Competitions*. Vol. 9378. LNCS. 2015, pp. 32–44. DOI: https://doi.org/10.1007/978-3-319-25396-1_4.

[15] P. Hubwieser et al. "Computer science/informatics in secondary education". In: *Proceedings of the 16th annual conference reports on Innovation and technology in computer science education - working group reports*. Ed. by L. Adams and J. J. Jurgens. ITiCSE-WGR '11. New York, NY and USA: ACM, 2011, pp. 19–38. DOI: https://doi.org/10.1145/2078856.2078859.

[16] P. Hubwieser et al. "A Global Snapshot of Computer Science Education in K-12 Schools". In: *Proceedings of the 2015 ITiCSE on Working Group Reports*. ITICSE-WGR '15. New York, NY, USA: ACM, 2015, pp. 65–83. DOI: https://doi.org/10.1145/2858796.2858799.

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

[18] A. Robins, N. Rountree, and J. Rountree. "My Program Is Correct but It Doesn't Run: A Review of Novice Programming and a Study of an Introductory Programming Paper". Technical Report. Otago, NZ: University of Otago, 2001.

[19] A. Robins, P. Haden, and S. Garner. "Problem Distributions in a CS1 Course". In: *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*. ACE '06. Darlinghurst, Australia, Australia: Australian Computer Society, Inc, 2006, pp. 165–173. ISBN: 1-920682-34-1.

[20] A. Robins, J. Rountree, and N. Rountree. "Learning and teaching programming: A review and discussion". In: *Computer Science Education* 13 (2003), pp. 137–172. ISSN: 0899-3408. DOI: https://doi.org/10.1076/csed.13.2.137.14200.

[21] N. Ragonis and M. Ben-Ari. "A long-term investigation of the comprehension of OOP concepts by novices". In: *Computer Science Education* 15.3 (2005), pp. 203–221. DOI: https://doi.org/10.1080/08993400500224310.

[22] T. Sirkiä and J. Sorva. "Exploring Programming Misconceptions: An Analysis of Student Mistakes in Visual Program Simulation Exercises". In: *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*. New York, NY, USA: ACM, 2012, pp. 19–28. DOI: https://doi.org/10.1145/2401796.2401799.

[23] T. Sirkiä. "Recognizing Programming Misconceptions: An analysis of the data collected from the UUhistle program simulation tool". Master's Thesis. Aalto University, 2012. DOI: https://doi.org/10.1145/2401796.2401799.

[24] J. Vahrenhold and W. Paul. "Developing and validating test items for first-year computer science courses". In: *Computer Science Education* 24.4 (2014), pp. 304–333. DOI: https://doi.org/10.1080/08993408.2014.970782.

[25] R. Caceffo, S. Wolfman, K. S. Booth, and R. Azevedo. "Developing a Computer Science Concept Inventory for Introductory Programming". In: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. SIGCSE '16. Memphis, Tennessee, USA: ACM, 2016, pp. 364–369. DOI: https://doi.org/10.1145/2839509.2844559.

[26] R. Caceffo, P. Frank-Bolton, R. Souza, and R. Azevedo. "Identifying and Validating Java Misconceptions Toward a CS1 Concept Inventory". In: *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE '19. Aberdeen, Scotland Uk: ACM, 2019, pp. 23–29. DOI: https://doi.org/10.1145/3304221.3319771.

[27] D. Zehetmeier, A. Bottcher, A. Brüggemann, and T. Veronika. *Development of a Classification Scheme for Errors Observed in the Process of Computer Programming Education*. 2015. DOI: https://doi.org/10.4995/HEAd15.2015.356.

[28] L. W. Anderson and D. R. Krathwohl. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*. Abridged ed., 4. print. New York: Longman, 2001. ISBN: 080131903X.

[29] K. Illeris. *Lernen verstehen: Bedingungen erfolgreichen Lernens*. Klinkhardt, 2010. ISBN: 9783781517639.

[30] D. M. Souza, K. R. Felizardo, and E. F. Barbosa. "A Systematic Literature Review of Assessment Tools for Programming Assignments". In: *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*. Apr. 2016, pp. 147–156. DOI: https://doi.org/10.1109/CSEET.2016.48.

[31] H. Keuning, J. Jeuring, and B. Heeren. *Towards a Systematic Review of Automated Feedback Generation for Programming Exercises - Extended Version*. Tech. rep. Technical Report UU-CS-2016-001. Utrecht University, Mar. 2016. DOI: https://doi.org/10.1145/2899415.2899422.

[32] M. Gaudencio, A. Dantas, and D. D. S. Guerrero. "Can Computers Compare Student Code Solutions as Well as Teachers?" In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education, SIGCSE'14*. Atlanta, Georgia, USA, Mar. 2014. DOI: https://doi.org/10.1145/2538862.2538973.

[33] N. Falkner, R. Vivian, D. Piper, and K. Falkner. "Increasing the Effectiveness of Automated Assessment by Increasing Marking Granularity and Feedback Units". In: *Proceedings of the 45th ACM Technical Symposium on Computer Science Education, SIGCSE'14*. Atlanta, Georgia, USA, 2014, pp. 9–14. DOI: https://doi.org/10.1145/2538862.2538896.

[34] P. Denny, A. Luxton-Reilly, and D. Carpenter. "Enhancing Syntax Error Messages Appears Ineffectual". In: *Proceedings of the 2014 Conference on Innovation and Technology in Computer Science Education*. ITiCSE '14. Uppsala, Sweden: ACM, 2014, pp. 273–278. DOI: https://doi.org/10.1145/2591708.2591748.

[35] K. Ala-Mutka, T. Uimonen, and H.-M. Jävinen. "Supporting students in C++ Programming Courses with Automatic Program Style Assessment". In: *Journal of Information Technology Education* 3 (2004), pp. 245–262.

[36] E. Araujo, D. Serey, and J. Figueiredo. “Qualitative aspects of students’ programs: Can we make them measurable?” In: *2016 IEEE Frontiers in Education Conference (FIE)*. Oct. 2016, pp. 1–8. DOI: https://doi.org/10.1109/FIE.2016.7757725.

[37] O. Seppälä, P. Ihantola, E. Isohanni, J. Sorva, and A. Vihavainen. “Do We Know How Difficult the Rainfall Problem is?” In: *Koli Calling 2015*. Nov. 2015, pp. 87–95. DOI: https://doi.org/10.1145/2828959.2828963.

[38] P. Mayring. *Qualitative content analysis: theoretical foundation, basic procedures and software solution*. 2014. URL: https://nbn-resolving.org/urn:nbn:de:0168-ssoar-395173.

[39] P. Shah, M. Berges, and P. Hubwieser. “Qualitative Content Analysis of Programming Errors”. In: *Proceedings of the 5th International Conference on Information and Education Technology*. ICIET ’17. New York, NY, USA: ACM, 2017, pp. 161–166. DOI: https://doi.org/10.1145/3029387.3029399.

[40] E. Gamma. *Design patterns: Elements of reusable object-oriented software*. 40th ed. Addison-Wesley professional computing series. Boston: Addison-Wesley, 2012. ISBN: 0-201-63361-2.

[41] A. R. Basawapatna, A. Repenning, K. H. Koh, and H. Nickerson. “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*. Ed. by B. Simon, A. Clear, and Q. Cutts. ICER ’13. New York, NY and USA: ACM, 2013, pp. 67–74. DOI: https://doi.org/10.1145/2493394.2493404.

[42] D. J. Bartholomew, F. Steel, I. Moustaki, and J. I. Galbrath. *Analysis of multivariate social science data*. 2nd ed. Chapman & Hall/CRC statistics in the social and behavioral sciences series. Boca Raton Fla.: CRC Press / Taylor & Francis, 2008. ISBN: 978-1-58488-960-1.

[43] I. Koller and R. Hatzinger. “Nonparametric tests for the Rasch model: explanation, development, and application of quasi-exact tests for small samples”. In: *InterStat* 11 (2013), pp. 1–16.

[44] I. Koller, R. W. A. Alexandrowicz, and R. Hatzinger. *Das Rasch-Modell in der Praxis: Eine Einführung mit eRm*. Vol. 3786. UTB Psychologie, Wirtschaftswissenschaften. Wien: Facultas, 2012.