Investigating Novice Programming Abilities with the Help of Psychometric Assessment

Marc Berges
Technische Universität München
Germany
berges@tum.de

Abstract: In 2008, a working group in the computer science department at Technische Universität München School of Education piloted an optional 2 ½ day experience that took place just before the first undergraduate lecture in computer science. This experimental project was a unique attempt to assess prior knowledge and programming skills for new CS students. The research team used item response theory on source code for 21 fundamental coding concepts in an attempt to assess novice programming skills. The responses were transferred to a Rasch Model which allows personal parameters to be assessed. Now that a metric has been validated for the coding ability, we are seeking to further our work by attempting to evaluate other core programming competencies. Originally, the complexity of the programming task was of central interest, we are seeking to further our work by creating a multidimensional assessment of problem solving abilities as well as coding abilities represents the different facets of these skills. How many dimensions are there and what do they look like?

Introduction

A few years ago, we have introduced a preliminary programming course for the freshmen of computer science at our university. These courses offered a broad research field for the investigation of source code that was written by students with well-known levels of programming experience in a closely controlled setting. By this way, we have gathered a substantial amount of source code during the last years.

Based on these results, we investigated the programming abilities of novice programmers at different knowledge levels. Now, these results are put together to find programming competences. For that reason, programming is seen as a latent construct that is not observable in a direct way. Psychometric methodologies are applied to find and assess the abilities of novice programmers.

After the investigation of the source code, the dimensions and characteristics of the programming abilities are getting into focus. The notions of computational thinking and problem solving are of special interest. So, what is the role of these two facets for the programming ability and how can it be assessed?

Assessment of Coding Ability

Although the direct evaluation of source code is difficult, several methodologies for assessing programming abilities from code have been presented, e.g. qualitative analysis of students' solutions (McCracken et al. 2001), measures of code quality (Hanks et al. 2004) or investigations about misconceptions (Sanders et al. 2007).

Yet, due to its complexity, we do not assume that any programming ability could be measured in a direct way. Nevertheless, we could regard certain attributes of the source code as manifestations of latent psychometric constructs according to the principles of item response theory. More detailed, we could treat the application of certain structural elements like loops, conditional statements or inheritance operators as positive responses on certain items (e.g. “existence of loops”). In consequence, the probability of such positive responses in dependence from the item difficulty and the estimated person abilities could be described by certain psychometric models, e.g. the Rasch Model (see Rasch 1980).

Data Gathering

Based on a concept extraction from the materials of the underlying course, a list of 21 concepts was formed according to the method described in (Berges et al. 2012): access modifier (AM), array (AR), assignment (AG), association (AC), attribute (AT), class (CL), conditional statement (CS), constructor (CO), data encapsulation (DE), datatype (DT), inheritance (IN), initialization...
(IS), instance (IT), loop statement (LO), method (ME), object (OB), object orientation (OO), operator (OP), overloading (OV), parameter (PA), and state (ST). For the final concept list, four of them are eliminated because of different reasons. OO is eliminated because it is provided by design of Java. CL and DT are excluded because it cannot be distinguished between “implementation by the students” and “implementation forced by the IDE”. Finally, IT is the same as OB and because of that only OB is included into the list.

For the calculation of a model in the item-response theory, a set of items is needed. In (Berges et al. 2012) the concepts mentioned above, were split up into observable items in the code. These items cover all observable aspects that are related to a specific concept. To gather the needed data, the code was either rated with 1 if the item is contained in the coder, or 0-rated otherwise.

During the investigation, the students were asked to implement a small project on the basis of an assignment that did not include explicit questions on programming. Nevertheless, the resulting programming code contains the responses on these questions. This is why we can assume the code-items to be assignments posed to the participants although they were not. Although the items do not cover all concepts that would be assigned to programming ability, we concentrate on the ones above because they are based on the concepts we extracted from the worksheets which are again the basis for the programming tasks.

In total 321 datasets from a preliminary programming course (see Hubwieser et al. 2011) are included in this research project. Each dataset consists of the personal data and a vector with the responses on all code items.

Results and Interpretation

Starting with the 33 items after the exclusion process, all items that violate the homogeneity criterion are eliminated. This is done by applying the non-parametric test proposed by Ponocy (2001) and Koller et al. (2013). Here the test on homogeneity of the items (T1m) is applied first. The test statistic is applied iteratively. In the end the remaining items are homogeneous. Now, as the item set has been reduced to six items, the exact tests can be applied for justifying the nonparametric tests. The Martin-Lof test (see Verguts 2000) is conducted on the resulting item set. Here, the p-value is 0.66. Thus, the items are assumed to be homogeneous and locally stochastically independent. After that, the two test statistics presented in (Bartholomew 2008) are calculated. For the general goodness-of-fit test statistic G², a value of 62.1 is resulting. Additionally, the X² test statistic compares in a value of 139.4. Both are not significant for 13 degrees of freedom to a level of 0.05 in the X²-distribution. Again, the H₀-hypothesis is rejected and the items are assumed to be homogeneous.

The final itemset includes the items OP3 (Are there any other operators used, apart from the assignment or logical operators?), CO2 (Is there a call of a constructor?), AT2 (Are attributes of other classes accessed?), OV2 (Is there an overloaded method used in the code?), and IN2 (Is the code using a manually created inheritance hierarchy?).

In Figure 1 the item characteristic curves for all items that are included in the model are shown. According to the definition of the Rasch model, they only differ in their level of difficulty. This is expressed in the figure by a shift on the x-axis, which shows the latent parameter on a scale of -10 to 10. All curves are parallel and only differ in the value of the latent parameter at the probability of 50% for rating a code item with “yes” (1). The probability that an individual with a specific value of the latent parameter has solved a specific item is drawn on the y-axis.

By definition, the item parameters sum up to 0. The items OP3 and IN2 have values of -5.4 and 5.0, respectively. The item that is closest to the average of 0 for the investigated population is AT2 (0.84). The use of attributes of other classes, either direct or by using a method, indicate participants with an average ability in coding, concerning the investigated items. Interestingly, all items except AT2 and OV2 have the same distance between each other. In general, Figure 1 presents a ranking of the items. The simplest item is OP3, which represents the use of arithmetic operators. The underlying concept is simple to code and all projects need calculations. As a result, the position within the items is not surprising. The next concept in the ranking is CO2, which indicates the use of a constructor or an initialization of an object. Again, the underlying concept is easy, but the basic object-oriented notions have to be implemented as well. Next, the items AT2 and OV2 indicate the use of interrelations between classes. As mentioned above, the first one represents the use of foreign methods and attributes. The second one represents the use of overloaded methods. Regarding the last item IN2 (use of an own class hierarchy), these two items represent more advanced concepts of object orientation. Thus, the item set contains representatives of simple coding concepts that can be related to the procedural paradigm, as well as representatives of advanced object-oriented notions.
Actually, there is a medium correlation (0.42) between the self-assessment of the students’ previous knowledge and their person parameters (p-value $\ll 0.01$). Regarding gender of the participants, females (-0.13) have a lower -but not significant- average person parameter than male students (0.26). On the other hand, the self-assessment has a significant difference (p-value $\ll 0.01$) in the person parameters. The students with previous knowledge have a mean value of 1.06, while those without any previous knowledge have a mean value of -0.93.

In addition to students’ previous knowledge and their gender, lines of code are another measurement that can be conducted on the source code. In particular, the projects differed a lot in their complexity. There were projects with only a few lines of code (min. 6 LOC) and some with more than one thousand lines of code (max. 1330 LOC) containing a GUI and other features. The mean value of project size regarding the lines of code is 212.7 LOC. Figure 2 shows the distribution of the project sizes in box plots for all participants, with and without previous knowledge.
For all participants, the median is 129 LOC, while the first quartile is 73 LOC and the third is 275 LOC. Furthermore, the projects developed by those with previous knowledge have significantly (p-value \( \ll 0.01 \)) more lines of code. The mean value for those with pre-knowledge is 253.2 LOC versus 160.7 LOC for those without pre-knowledge. Regarding the person parameters of the Rasch model, there is no correlation (0.07) to the lines of code.

Figure 2: Box plots of the lines of code for the different previous-knowledge groups

Validation of the Rasch model is quite difficult. In contrast to the formal validation that is described above, validating whether the items really measure the programming ability as the latent dimension is hard to proof. In fact, the items only cover a part of programming ability as some concepts are missing that are not mentioned in the material for the preliminary course. Additionally, the facet of problem solving that is a huge part of the programming ability cannot be assessed by a simple structural analysis.

Although the model is fitting the data, there still remain some problems. Obviously, there are different kind of difficulty concerning the code items. On the one hand, there are concepts that are difficult in a common understanding and there are concepts that force the programmer to recognize its use for a better programming code. So, this distinction implies two different kind of programming abilities and because of that more than one dimension. The investigation of the relationship of these dimensions has to content of future work.

**Computational Thinking and Programming Ability**

As our investigation on the development of an assessment tool to find computational thinking aspects in programming tasks is in its first steps, here, a brief literature review is given.
In (Barr et al. 2011) the core concepts of computational thinking -defined in (Wing 2008)- are presented: data collection, data analysis, data representation, problem decomposition, abstraction, algorithm and procedures, automation, parallelization, and simulation. Furthermore, applications in different subjects are presented. For computer science most of them are related to programming. For our purpose of investigation the programming abilities problem decomposition, abstraction, and data representation are important.

According to Wing (2006) “computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system. It is separation of concerns. It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable.”

After clarifying the concepts underlying computational thinking, two studies of the relationship of computational thinking and programming are presented.

Thuné et al. (2010) interviewed students of a mandatory programming course on their attitudes toward programming. More precisely, the experiences of the participants were enlightened in a phenomenographic study. Interesting for our research is the resulting category in which programming is related to problem solving. Obviously, students relate those two topics if they are asked about programming. Asking in a more detailed way could provide facets of the abilities needed for the programming tasks.

Another study on the relationship of computational thinking and programming is conducted by Kazimoglu et al. (2012). They investigated computational thinking skills in a programming course based on serious games. The development of a framework enhancing computational thinking skills during programming could be a serious step in the characterization of the relationship between the two concepts.

For the investigation of the role of computational thinking for the programming ability, in a first step, observable items fostering computational thinking have to be generated. Afterwards, these items have to be tested and correlated with the code analysis provided above. In a final step a test instrument has to be created.

Conclusions
The application of the Rasch model on source code provided interesting insights into the results of an preliminary programming course. First, it is possible to provide implicit questions in a more complex task. This is, especially, useful for programming, as the simple reproduction of small knowledge elements (e.g., loops or conditional statements) is not very difficult.

Nevertheless, the simple code analysis only reveals clues for possible competencies. To gather these competencies, in a first step, the characteristics have to be examined. For this purpose, computational thinking and problem solving are investigated on their role in programming.

References


