Researchers identified different reasoning levels to be one source of error in the communication between instructor and student in programming education. How a student traverses between those different levels of reasoning was studied extensively. However, the results from this research are not practical for the instructor. This paper deals with this problem and proposes a categorization method, as well as an exercise catalog for the instructor, to let her determine the reasoning level of students through the use of written tests only. This approach is investigated with in a case study. It is shown that written tests, which are designed based on the proposed approach, do not always lead to the correct determination of reasoning level. The reasons for this, namely students creative strategies to gain the most points on exercises, are discussed and improvements to this approach are proposed.

Preface

This thesis is the final product of the master education in Information Sciences at the Radboud University Nijmegen. I would like to thank my head supervisor Erik Barendsen and my co-supervisor Sjaak Smetsers for their support, time and valuable feedback. Furthermore I want to thank all participants, who volunteered to be a part of this research. I am also grateful to the Institute for Computing and Information Sciences for their free-of-charge coffee vending machine, which helped me lot during the writing process. Finally I would like to thank my family and friends for their support during my time studying Information Sciences at the Radboud University.

Introduction

Studies show that many novice programmers struggle during their first exposure to the field of programming. Most often they perform below their instructors’ expectations (McCracken et al., 2001). A multi-national study conducted by Watson and Li (2014) shows that the pass rate for introductory programming courses is only 67.7%. It is also concluded in this study, that programming paradigm and language have little to no effect on the learning success of novice programmers. This conclusion shows, that the debate about best suited programming paradigms and languages for introductory programming courses will likely not have an impact on the learning success of novice programmers.

If language and paradigm play little to no role, the root problem causing the low passing rates has to lie somewhere else. Lister (2011) took an approach, which is language and paradigm agnostic. While he studied the role of abstraction in programming education, he recognized that the capability of reasoning abstractly is fundamentally different between programming experts and novices. Translated to the domain of education, he concluded that the main problem stems from instructors explaining and guiding students using a language, which represents an experts level of abstraction. The problem arises when instructors are faced with students, who can not (yet) grasp this level of abstraction, “we (the instructors) may as well be lecturing in a foreign language, and in a sense we are.” (Lister, 2011).

Thus Lister identified a form of miscommunication between students and teachers, which arises from different levels of reasoning. According to Lister (2016), the unawareness of teachers about those different reasoning levels could be the root problem, which causes novice programmers to fail. In his research about these different levels of reasoning, Lister (2011, 2016) proposed a model, based on neo-Piagetian theory, which describes the different levels of reasoning (called neo-Piagetian stages) a novice programmer traverses. Each of those levels is accompanied by a different set of characteristics and skill sets, which the novice exhibits. According to Lister (2011), instructors should confront their students only with exercises and explanations, which match their respective reasoning levels.

Lister’s model is supported by empirical evidence found by Teague (2015). Among others Corney, Teague, Ahadi, and Lister (2012) found empirical evidence that preoperativeal students perform poorly on exercises, which require abstract reasoning. In order to find this evidence Teague (2015) used a mixed methods approach by incorporating both qualitative and quantitative research elements. She argues that by only conducting written tests, “(...) there is actually very little evidence of the thought processes, problem solving and reasoning skills the students employed.” Concerning this
problem she rather used think-aloud-sessions in her research to categorize students into their respective stage in Lister’s Model (Teague, Corney, Ahadi, & Lister, 2013).

In summary, Lister (2011) proposed a model which can help instructors recognize their students reasoning level and Teague (2015) found empirical evidence, which backs his model.

However, Lister and Teague did not formulate how their findings can be applied in a classroom setting. This problem is covered in this paper and in particular, how an instructor can utilize their findings. One practical problem arises when Teague et al. (2013) argued that the determination of the stage should not be conducted only through the use of written tests. This is because an instructor has most likely limited resources, so that she or he might not be able to use think-aloud-sessions (like Teague) with each of his students. While Teague is correct to assume that less information is gained about a student by conducting written tests opposed to think-aloud-sessions, it could still be enough information to determine the neo-Piagetian stage of a novice. In order to investigate the practicality of Lister’s model it is thus important to investigate the suitability of written tests to determine the reasoning level of a novice.

**Background**

This section describes the underlying concepts of Lister’s Neo-Piagetian Theory For The Novice Programmer and its stages, as well as Teague’s approach of evaluating the stage of a novice.

**Reasoning vs. Tracing**

According to Lister, the core skills on which to deduce the stage of a novice are tracing and reasoning. Simply put tracing is the ability of a student to execute a piece of code in their mind. For example to determine the values of some variables after a series of variable assignments. Reasoning about code is the ability to state abstractly what a some piece of code does.

Where exactly is tracing then different from reasoning? Take for example the code example from Figure 1. A student who is able to trace but not to reason about code, could rightfully give the end state of the program given the start state. For example, given the start state \{y1=3, y2=4, y3=5\} he could rightfully deduce the end state \{y1=5, y2=4, y3=3\}. This is achieved by plugging the values into the variables and by executing every line of code in isolation. But faced with the question, on what the code does on an abstract level (in this case, sorting three values in decreasing order), the student who can trace but not reason would not be able to answer correctly.

Figure 2 shows a simplified overview of the stages of reasoning respective to the novices ability to trace and reason at that stage. A student begins at the sensorimotor reasoning level, in which he is not able to trace code reliably let alone to reason about code. Then a novice traverses to the preoperational stage, in which he is able to trace but not reason. After that the novice reaches the concrete operational stage, where he has developed the ability to reason abstractly about code.

**The Overlapping Waves Model**

Is a programmer at one specific stage of reasoning in any given moment, for any given skill? Lister (2016) assumes that this is not the case. He accepts the so called “overlapping
waves" model, seen in Figure 3 (taken from Lister (2016)). A person can reason at a range of different levels. However, she or he will reason mostly at the level their predominant stage determines. For example, when a person starts to learn a new skill, she or he will reason predominantly at the sensorimotor stage. After they learn about this new skill, they will reason less at the sensorimotor and more at the preoperational stage and traverse in that way. Thus different ways of reasoning can coexist but at any moment one specific stage is predominant.

Accommodation vs. Assimilation

It may seem that traversing between the stages would happen in a linearly manner, but this is not the case. The introduction of new knowledge might shatter old ways of thinking. For example, a novice programmer who reasons at the preoperational level might not be able to trace code which includes new concepts (like loop, arrays or recursion). For those new concepts the novice programmer would reason at the sensorimotor level. This process is known as accommodation and is described in Figure 4 (taken from Lister (2016)).

Assimilation denotes the opposite, when a programmer is introduced to a new concept and his mix of stages does not change in response to it. For example, knowing integer values, the concept of floating point numbers might not affect the reasoning of the novice (Lister, 2016).

Neo-Piagetian Stages

This section gives a more detailed view on the different stages of Lister’s model. The lines of code in Figure 1 are used as a progressive example at the beginning of each stage description. The task of the student in each example is to trace, given the inputs \( \{y_1=3, y_2=4, y_3=5\} \), and to explain what the code does.

Sensorimotor Stage. The student answers the tracing question wrongly with \( \{y_1=3, y_2=5, y_3=4\} \) and is not able to formulate an answer to the second question.

Lister (2016) describes a novice programmer in the sensorimotor stage to have an incoherent understanding of program execution. A sensorimotor novice programmer can be distinguished from a programmer in the preoperational stage by analyzing their ability to trace code. That is, sensorimotor novice programmers can only solve tracing exercises less than a 50% success rate (Lister, 2011). Teague (2015) elaborates further on distinguishing behavior of a novice programmer in the sensorimotor stage opposed to other stages. In this stage tracing attempts take extensive cognitive effort and the main strategy for solving problems is trial and error. The sensorimotor novice programmer is just developing his or her language skills programming, therefore he or she has a lot of misconceptions about code as well as problems to distinguish between different parts of code (e.g. built-in names and reserved words in the programming language). This fragile understanding of programming concepts (right or wrong) is applied inconsistently. Teague concludes that the novice programmer is not able to trace code accurately when he has not a coherent understanding of the programming syntax and semantics.

Concrete Operational Stage. The student examines the code and finds its abstract meaning: It orders the three numbers in a decreasing manner. Knowing this, he deduces the answers to the tracing question: \( \{y_1=5, y_2=4, y_3=3\} \). He then answers the second question correctly.

Concrete operational reasoning distinguishes itself from preoperational reasoning through the use of abstract thinking. A student at the concrete operational stage is able to
reason about conservation, reversible processes and transitive inference (Lister, 2011).

In the domain of programming one aspect of the concept of conservation is the preservation of a specification across variations in the implementation. That is, the novice is able to change its implementation while the specification of it is unchanged. For example if the specification is to provide an algorithm to sort a list, the novice would be able to create or reason about more than one implementation of this algorithm (preserve the specification across variations in the implementation).

The ability to find the abstract meaning of code, enables the novice to write code which reverses the effect of previous code. This is called implementation of reversible processes. For instance, when the novice is given a method which mutates an object, he is able to write a second method, which undoes the effect of the first.

The novice is able to apply transitive inference at this stage. He can deduce the relationship between \(A\) and \(C\) by observing the relationship between \(A\) and \(B\) along with the relationship between \(B\) and \(C\). Even though the novice at the concrete operational stage is able to reason in an abstract manner, this abstract thinking is restricted to only familiar and real situations.

**Formal Operational Stage.** Formal operational reasoning is the most advanced and abstract form of reasoning, where programmers are able to apply abstract thinking to unfamiliar and theoretic situations. Even though this level of reasoning should be the goal for every novice programmer, this paper does not focus on this level, since most students need guidance to traverse between the preceding stages (Lister, 2016).

**Practical Usage Of The Neo-Piagetian Stages**

Lister (2011) states, that in order to help a student to traverse between the stages, he should be confronted with exercises, which require the same abstract reasoning that the student is capable of. After the student has done a certain amount of those exercises, he will need less cognitive effort to solve them. Since the student needs less cognitive effort for this type of exercises, he can use the freed-up cognitive ability to reason more abstractly then before (he traverses to the next neo-Piagetian stage). Thus an instructor should first determine the neo-Piagetian stage of her students and after that confront them with stage appropriate exercises and explanations.

As an example, assume an instructor is teaching two students (\(A\) and \(B\)), which are in different neo-Piagetian stages (\(A: preoperational\) and \(B: concrete operational\)), about an algorithm.

Without the knowledge of neo-Piagetian stages, the instructor would present this algorithm to both students in the same way. She would show an implementation of the algorithm and would assume that both students would understand how the algorithm works. Student \(B\) would in fact understand the algorithm just by looking at the code, since he is able to reason about code abstractly. Student \(A\) would not be able to understand the algorithm. He would try to trace the shown code with some exemplary values and would try to derive meaning from the in and output of the code, but he would not see the purpose of it.

With the knowledge of neo-Piagetian stages, the instructor would confront Student \(A\) with tracing exercises until he would show that he is capable of reasoning at a concrete operational level. Only then the instructor would confront the student with exercises that require abstract reasoning about code.

**Teague’s Stage Evaluation**

In her research Teague et al. (2013) used think-aloud sessions to investigate the reasoning for each neo-Piagetian stage. In those sessions the novice was given a set of programming exercises and was asked to think aloud. The set of programming exercises contained out of a set of code tracing and code explaining exercises. During the think-aloud-sessions emphasis was laid on minimizing the cognitive effort of the novice to produce those verbalizations. The novice should simply articulate her or his thoughts, rather than formulate an explanation for the interviewer. Those sessions were recorded and those recording were later analyzed. Teague studied the shown reasoning behavior and in particular the relation between performance on tracing and explaining tasks. From comparing the displayed behavior with the neo-Piagetian stage typical behavior (found by Lister (2011)), Teague determined the neo-Piagetian stage of the participant. One interesting result from her research is the collection of a set of distinguishing programming behaviors for each neo-Piagetian stage (Teague, 2015). As an example, a few of those programming behaviors can be seen in Figure 5.

**Aim Of The Study**

The aim of this paper is to give insights about a practical way to determine the neo-Piagetian stage of students. The research described in this paper is divided into two parts. In the first part, a practical way to apply Lister’s model is proposed. This proposal emphasizes the use of written tests. In this proposal already existing exercises from literature for the determination of the neo-Piagetian stage are analyzed and categorized. With the help of this categorization an exercise catalog is presented. Then a manner is proposed on how this exercise catalog can be used by instructors.

The second part of this research concerns the validity and applicability of the proposed approach. In a case study, the behavior of eight students is examined, while written exercises as well as interviews are conducted.
Figure 5. Distinguishing Behavior

<table>
<thead>
<tr>
<th>Programming Behavior</th>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Struggles to trace code</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Tracing attempts take considerable</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>cognitive effort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relies on specific values to trace</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>code</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Traces without shortcuts (because can</td>
<td>-</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>not reason about code</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tracing is a mechanic process, without</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>construing purpose</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Can trace reliably</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Can work with the concept of</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Conservation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Can short-cut trace</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

The research question reads as follows:

*To what extent can the neo-Piagetian stage of a student be determined using written tests and interviews?*

This research question is further divided into the following research questions:

**RQ1** Are the results of written tests on neo-Piagetian stages in the case study conform to the expected results described in literature?

**RQ2** Is the behavior observed in the interview conform to neo-Piagetian theory?

**RQ3** Do the neo-Piagetian stages determined by the written tests correspond to the neo-Piagetian stages determined by the interviews?

The described case study was conducted in a computer programming course during a computer practicum. Students of this course could choose, whether they wanted to work on an assignment for the course or participate in this study. Since the participation was voluntarily, it was told to the students that they would benefit from taking the test seriously. This is because first it could help them to identify gaps in their programming knowledge and second it would help the teaching staff to give them more appropriate advice. All students of this computer programming course have had at least six months of programming experience and completed at least two prior programming courses successfully.

**Method**

The first part of this section contains a proposal on how to categorize existing and new exercises for the determination of the neo-Piagetian stage. In this part the reasoning behind a set of categorized exercises is discussed and then a way to apply those exercises in the classroom is proposed. In the second part, a case study is described, which is concerned with the proposed application.

**Categorization Of Exercises**

**Topic Dependency.** When trying to estimate the level of reasoning of a novice, it is important to recognize that those levels are topic dependent. For example a novice may be able to reason concrete operational for lines of code, which contain only variable assignments, but she would only be able to reason preoperational on lines of code, which contain lists. In the literature this shifting of reasoning levels is explained through the process of *accommodation* (Lister, 2016). Another way to explain these shifts of abstract reasoning can be found in neo-Piagetian literature, where the level of reasoning is seen as skill dependent (Teague, 2015). One can argue that each concept in programming can be seen as a separate skill to the novice. Thus, the reasoning level of a novice programmer can not be described by merely one stage but by many, where each stage corresponds to a programming concept the novice programmer was introduced to.

This paper covers the most basic topics: *Variable Assignment*, *if-Statement*, *while-Statement*, *Lists* and *for-Statement*. Surely there are more topics in programming, like for example inheritance. Thus this list of topics is not complete, but nor is this the goal of this paper. Using the following categorization and application method, one could create and categorize new exercises of those unmentioned topics, to determine the stage of a novice programmer.

Every exercise is categorized by the concepts it covers. For example the exercise in Figure 6, covers only one topic, which is *Variable Assignment*.

**Exercise Level.** Exercises are further categorized by their level. The level of an exercise denotes whether it can be used to determine the sensorimotor, preoperational or concrete operational stage of a novice. Exercises for sensorimotor programmers (*exercise level one*) are different from ex-
Figure 7. Conclusions From Answers

<table>
<thead>
<tr>
<th>Level 1</th>
<th>Level 2</th>
<th>Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Sensorimotor</td>
</tr>
<tr>
<td>✓</td>
<td></td>
<td>Preoperational</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
<td>Concrete Operational</td>
</tr>
</tbody>
</table>

eercises for preoperational and concrete operational programmers (exercise level two). This is because for sensorimotor programmers, misconceptions about the syntax and basic concepts are predominant.

According to Lister (2011), a test which determines whether a novice programmer is in the sensorimotor stage should identify if the participant is able to successfully answer tracing exercises at least 50% of the time. The code tracing exercises should uncover the novice programmers misconceptions. The level one exercises, which are covered in this paper, are targeting common misconceptions of novices. The benefits of targeting common misconceptions isolated in tracing exercises are twofold. First those tracing exercises are covering a wide variety of code and provide therefore an accurate depiction of the programmers ability. Second, if a question is answered wrongly, then the student probably has the underlying misconception, which is targeted in the exercise. Through the identification, misconceptions can be corrected by explicit explanations from the instructor. The example exercise in Figure 6 has level one and covers the misconception: Expression is not evaluated when assigned. A description of all exercises and explanations to the misconceptions they cover can be found in section Exercise Catalog.

Given it is established, that the novice is not sensorimotor, then if he is able to answer questions that require abstract reasoning, his stage is concrete operational, else his stage is preoperational (Lister, 2011). This is summarized by Figure 7. Note that a novice, who can not answer level one questions correctly will most likely not be able to answer level two questions correctly. Her or his inability to answer level one questions correctly alone, categorizes her or him as sensorimotor (Lister, 2011).

Types Of Concrete Operational Reasoning. Lister (2011) discussed three types of abstract reasoning, which separate preoperational novices from concrete operational ones: conservation, transitive inference and reversible processes (see Background - Neo-Piagetian Stages - Concrete Operational Stage). Level two exercises are further categorized by these different themes. When examining the novice, she should be able to answer questions from all three different themes of one topic to categorize her as concrete operational in this topic.

The underlying information structure of the categorization of exercises can be seen in Figure 8.

Exercise Catalog

This section contains a discussion of a set of exercises divided by topic. Those exercises are partly based on or taken from studies about misconceptions or neo-Piagetian stages. To function better as an accessible reference for teaching material, the exercises are bundled in the Appendix of this paper. In order to follow this section, please review the Appendix. The programming language of all exercises is Java. Some misconceptions may be Java specific, however most misconceptions and exercises should be easily transferable to other programming languages.

Variable Assignments. VA1M001. In this exercise, the following misconception is covered: The novice programmer thinks that a variable assignment can be written either way, from left to right and from right to left. This is because lines of code look similar to mathematical equations, in which it
TOWARDS A PRACTICAL APPLICATION OF THE NEO-PIAGETIAN THEORY FOR NOVICE PROGRAMMERS

7

does not matter whether a statement is on the left or right side. Therefore, a student might believe, that the same rule applies to programming code (Du Boulay, 1986). Those students would answer VA1M001 with c instead of the right answer b.

VA1M002. Covers the misconception, that novices might not know that an expression (in Java) is evaluated during variable assignments. They could think that a variable could contain the expression rather than the outcome of an expression (Du Boulay, 1986). Those students would answer VA1M002 with 7 + 3 instead of the right answer 10.

VA1M003. The misconception that a primitive type variable can contain more than one value, is covered in this exercise. This misconception might occur, because it was taught that a variable would behave like a box. The novice reasons that a box can contain multiple items, so why should a variable not (Du Boulay, 1986). Those students would answer Q.1.03 with b instead of the right answer c.

VA1M004. Box analogies might also lead students to believe, that assigning the value of one variable f to another variable a will empty the value of f (Du Boulay, 1986). This misconception is covered in this exercise. Those students would answer VA1M004 with b instead of the right answer a.

VA1M005. The concept of the empty string is hard to grasp for some students, since it does not occur in natural language (Du Boulay, 1986). The misconception, that the empty string is not a value is covered in this exercise. Those students might answer VA1M005 with statements like f has no value or f is "Hello World" instead of the right statement f is "".  

VA1M006. After assigning the primitive value of one variable to another, novices might have the misconception, that those variables now are linked together (Du Boulay, 1986). Those students would assume that after the code in VA1M006 is executed, b = a; would link a to b, so that the following line a = 4; would also change b. Those students would answer that a is 4 and b is 4 instead of the right answer: a is 4 and b is 3.

VA1M007. Some students might not recognize the need for variable initialization, thus have the misconception that variables do not need to be initialized explicitly before their value is used. They would wrongly reason, that the value of an uninitialized integer is zero because it seems intuitive (Du Boulay, 1986). Those students would answer VA1M007 with c instead of the right answer b.

VA1M008. The misconception, that subsequent lines of code can have an effect on previous code is covered in this exercise. This misconception is coined the "parallelism bug" by Pea (1986). Students who have not grasped that code is executed in a sequential manner, might reason that code is executed some way in parallel before the program outputs something. A novice with this misconception will likely answer VA1M008 with b, because they would assume that the change to the values of the variables width and height would have an effect on the previously assigned variable area.

VA1M009. The code in this exercise denotes a series of variable assignment. This exercise covers a collection of previously examined misconceptions. It can be used to determine whether the novice is able to handle a series of variable assignments. The variable assignments in this exercise cover the assignments of a value, an expression, another variables value and an expression which contains another variable. Should the novice fail to answer this question correctly, then it is likely that she has misconceptions about one of those variable assignments or the complexity of this exercise demanded to much cognitive effort.

VA2R001. In this exercise the novice needs to write code, which reverses the effect of the given code. In order to do so she has to be able to reason abstractly on the meaning of the given code. If this question is answered correctly the novice is likely to reason at a concrete operational level (Lister, 2011).

VA2C001. In this exercise the novice needs to switch the values of two variables. This can be accomplished in two different ways, because it does not matter which value is stored in a temporary variable. The novice needs to remember for which implementation he chose, in order to pick the correct subsequent lines of code (conversion) (Lister, 2011). The two correct solutions are: c = a; a = b; b = c; and c = b; b = a; a = c; If this question is answered correctly the novice is likely to reason at a concrete operational level (Lister, 2011).

If-Statements. CI1M001. The misconception that an if statement gets executed whenever the condition applies is covered in this exercise. The reason for this misconception might be, that in natural language the word if can have similar meaning as the word whenever, whereas in code the keyword if does not. Novices with this misconception would assume that after the code of CI1M001 is executed, the following is printed: second first . This is because at the last line of code a is assigned 10 which would in the mind of the student "trigger" the execution of the if statement.

CI1M002. This exercise covers the misconception, that the computer is able to understand the intention of a piece of code and will behave accordingly. This fallacy was coined the intentionality bug (Pea, 1986) and follows from the superbug, which denotes the misconception, that "[...] there is a hidden mind somewhere in the programming language that has intelligent, interpretive powers." (Pea, 1986). The code in CI1M002 could be interpreted as a program which determines, whether a given integer from a user input is bigger than 5. Students who assume that the computer understands the intention of the program will likely trace the code wrongly. It is likely, that they would miss that the computer would print out Small Number! regardless of whether the
number given by the user was small or big.

CI2R001. In this exercise the novice needs to write code, which reverses the effect of the given code. The novice needs to recognize, that this code sorts two variables in decreasing order and then needs to write code, which brings those variables in their respective start state. If this question is answered correctly the novice is likely to reason at a concrete operational level (Lister, 2011).

CI2T001. A variant of this exercise was used by Lister (2011) to determine whether a novice programmer is able to reason with transitive inference. After the code in CI2T001 is executed the novice needs to deduce that y1 ≥ y2 ≥ y3 or that y1, y2 and y3 are ordered from big to small. She needs to use transitive inference, because in the code there is no direct relationship between y1 and y3, this relationship needs to be inferred.

CI2T002. This exercise, was used by Teague and Lister (2014) in order to investigate the novices ability to reason with transitive inference. The novice needs to deduce, that the purpose of the code is to print the smallest value. Because only two variables are compared to each other, the novice needs to infer the relationship between all of the three variables.

While-Loops. CW1M001. In this exercise the misconception, that a while loop is immediately stopped when the condition no longer resolves to true, is covered. The reason for this misconception might be that the word while in natural language might have the meaning, that an activity should be immediately stopped when the condition no longer holds. This might lead some students to reason that a while condition is evaluated after every line of code inside a while loop (Du Boulay, 1986). In the exercise those students would assume, that the while loop with the condition counter < 2 would translate in natural language to: As long as the value of counter is smaller than 2, which would lead the student to believe that immediately after the value of counter would be changed, the while condition would be tested again. Those students would answer CW1M001 with start end start end instead of the right answer start end start.

CW1M002. This exercise covers the misconception, that in a do-while loop, the condition is evaluated before its first iteration. Novices, which have this misconceptions, are thinking that count will not be incremented by this code.

CW2T001. A variant of this exercise was used by Whalley et al. (2006) and Lister et al. (2006), in which the for-loop is substituted with a while-loop. Lister (2011) explains, that because the novice needs to reason about the meaning of the comparison of two array elements for the global purpose of the code, she has to be able to use transitive inference to answer this question correctly.

CW2C001. This exercise targets the ability to make transformations between implementations, while conserving the specification. Only novices on a concrete operational reasoning level are able to solve this kind of problems according to Lister (2011). The code in this exercise shows a complicated manner of implementation for a method, which greets the user for a given amount. The novice needs to deduce the purpose of the code and is then able give an alternative implementation.

Lists & For-Loops. LI1M001. The misconception, that the index-count for arrays and lists starts at one, is covered in this exercise. This misconception might occur because of the natural manner of counting items in a list. In the natural language, if one goes through all items in a list, he or she usually numbers the first item as one instead of zero. The phenomenon of arising misconceptions, when using metaphors from natural language, is covered by Du Boulay (1986).

LI1O001. This exercise does not cover any specific misconceptions. However, a novice has to be able to understand the basic workings and syntax of for-loops and lists to answer this question correctly.

LI1O002. A variant of this exercise was used by Lister, Fidge, and Teague (2009) in order to investigate, whether a novice is able to trace code reliably which includes lists and for-loops.

LIZR001. This exercise was used by Lister (2011) in order to determine whether a novice is able to implement reversible processes.

LIZC001. Lister (2011) used this exercise, to determine whether a novice is able to conserve a specific way of implementation in subsequent lines of code.

LIZT001. In the code of this exercise, the items of one list are sorted into three lists, based on the items value. The novice needs to infer the relationship between the values of two of the resulting lists. In the code, this relationship is not particular stated in the code and thus has to be inferred by the novice. Thus, novices who answer this question correctly, need to use transitive inference.

LIZT002. This exercise was used by Whalley et al. (2006) and Lister et al. (2006) (A variant of this exercise is CW2T001). The novice needs to reason about the meaning of the comparison of two array elements for the global purpose of the code. Thus she has to be able to use transitive inference to answer this question correctly (Lister, 2011).

Application Of The Exercise Catalog

A test can be created from those proposed exercises. For each relevant topic for the tested novices, exercises of both levels should be selected. A set of level one questions can be used to determine whether the novice reasons at a sensorimotor or preoperational level and a set of level two questions can be used to determine whether the novice reasons at a concrete operational level.
The analysis of level one questions includes the percentage of correctly answered questions. Lister found out, that novices, who can not solve code tracing exercises reliably (at least 50% of the time), are not able to reason about code and thus are in the sensorimotor stage (Lister, 2011). Thus, if a novice answers less than 50% of the level 1 questions for a given topic they can be classified as sensorimotor for this topic. According to Lister, those novices are not able to reason about code abstractly (Lister, 2011). Therefore it can be assumed that they are not able to answer level two questions correctly.

For Level 2 Lister makes no assumption about a specific percentage of rightly answered questions and their implication for a given stage. However, he states, that if a novice is able to answer a question, which involves Reversing, Conservation or Transitive Inference, he is likely to reason at a concrete operational level (Lister, 2011). Thus as a rule of thumb, the more correctly answered Level 2 exercises, the more the student is likely to reason at a concrete operational level.

Case Study Of This Approach

The validity of written tests, which are composed out of the proposed exercise catalog is investigated in this study. A written test is composed out of a set exercises for each topic discussed in this paper. The participants take this written test and after that are interviewed about questions for each topic from the test. In those interviews, the neo-Piagetian stage of each subject is determined through the classification of the observed behavior. The relationship between this behavior and the respective neo-Piagetian stage is defined by Teague (2015) (see Figure 5 as an example). The outcome of the written test will also indicate a neo-Piagetian stage for each topic. Those outcomes will be compared, in order to find how close the results from the written tests resemble those of the interview.

Written Test. For each topic a set of level one and level two exercises were chosen for the written test. The amount of exercises was limited, to guarantee a maximum examination time of two hours. In further research without this limitation the amount of exercises could be increased and the influence could be studied. The exercises were chosen arbitrarily from the exercise catalog, only with the time limitation and equal distribution between level one and level two exercises per topic in mind. The following exercises per topic were chosen for the written test in this study:

- Variable Assignments: VA1M007, VA1M008, VA1M009, VA2C001, VA2R001
- If-Statements: CI1M001, CI1M002, CI2R001, CI2T002
- While-Loops: CW1M001, CW1M002, CW2T001, CW2C001
- For-Loops & Lists: LI1M001, LI1O001, LI10002, LI2R001, LI2C001

For the sake of simplicity the answer to those questions are graded either as correct or incorrect. In further research it could be investigated whether different results could be obtained when a continuous grading system is applied. Lister and Teague made no assumptions on whether a continuous or binary grading system is superior.

For each topic the percentage of correctly answered level one questions is calculated. If this percentage is below the threshold of 50% (Lister, 2011), the participant is classified for this topic as sensorimotor. Given that the participant is not classified as sensorimotor, the percentage of the correctly answered level two questions is examined. If the percentage is close to zero, the participant is classified as preoperational. In case that the percentage is close to 50, then the participant is classified to lie in between preoperational and concrete operational reasoning behavior. When the percentage is close to 100, the participant is classified as concrete operational. According to literature, if a student is classified as sensorimotor, he should not be able to answer a question that requires abstract reasoning (level two questions).

It will be investigated whether a student in this study, who was categorized as sensorimotor (by not answering level one questions correctly), was still able to answer level two questions, which require abstract reasoning. In addition to that it will be analyzed how the different neo-Piagetian stages are distributed among the participants.

Interview. One level two exercise for each topic was covered in the interview. All of these exercises were also part of the written test. These exercises were covered in the interview: VA2C001, CI2T002, CW2C001, LI2C001. Like in the written test, the amount of exercises to be covered in the interview was limited by the maximum examination time. It was chosen for level two questions, because level one questions only determine whether a novice is sensorimotor. Note, that since in the interview only level two questions were examined, it could not be determined whether a student is sensorimotor or preoperational when she exhibited non concrete operational behavior. If it was shown from previous tracing exercises, that a participant was sensorimotor, the interview would identify him or her thus as preoperational.

During the interview the participant was asked to redo and verbalize his reasoning process for the respective exercises. The interviewer only asked questions to clarify which of the distinguishing behavior (found by Teague (2015)) the novice exhibits. This procedure differs from the approach of Teague et al. (2013), since the interviewer can direct the participant on the specific aspect of stage dependent behavior. This difference of method occurs because the goals of Teague’s and this study are slightly different. While Teague was not only trying to determine the neo-Piagetian stage of the participant, she was also searching for patterns in the behavior of the par-
participants. Therefore in her research, she did not use a semi-directive interview, but undirected think-aloud-sessions.

In the interview, the stage of the participant was determined by identifying typical reasoning behavior, which are stated by Teague (2015). For example, two of those identified behaviors for the preoperational stage are relying on specific values to reason about code and tracing without shortcuts (because it is not reasoned about code; see Figure 5 as an example for distinguishing behavior). The interviewer also acted as the behavior-observing classifier for the participants.

During the interviews the interviewer made sure that insights about the following often occurring distinguishing behaviors of the participant were recorded: "Relies on specific values to trace code (preoperational)" vs. "Sees the purpose of the code (concrete operational)" and "Sees every line of code as separate entities (preoperational)" vs. "Sees and understands the relationship between code (concrete operational)" (Teague, 2015).

For each exercise covered in the interview, the student was given the exercise on paper (without his or her previous answer) and then was asked to share his or her thoughts on the shown exercise. During the time when the student shared his or her thoughts on an exercise, the interviewer analyzed whether the student showed or described distinguishing programming behavior that was found by Teague (2015). After the student stopped talking, the interviewer decided whether the student showed or described the distinguishing programming behavior, if not a directive question was asked to ensure that enough information was given to determine his or her neo-Piagetian stage (for example: Did you try this out with concrete values or derived this meaning only from the code?). The transcripts of those interviews were then analyzed and compared to the distinguishing programing behavior found by (Teague, 2015), in order to determine the neo-Piagetian stage of the interviewee. It is then analyzed whether this method is suitable to determine the neo-Piagetian stages and if the observed behavior was contradictory to the behavior described in the literature.

**Comparison.** The validity of the written test compared to the interview is investigated by reviewing the claim of Lister (2011), that participants who can not trace code, are not able to reason about it. Transferred to this setup, this means that no participant should be classified as concrete operational in the interview, while he was classified as sensori-motor from the written test. It will be investigated, whether the data adheres to that statement.

The results of the written test will be compared to the results of the interview. It will be analyzed to what extend the results of the written test predict the result of the interview. This will be done for the exact prediction, that is whenever the written test results indicate the exactly same neo-Piagetian stage as the interview. But also for the prediction within the range of half a stage. As Lister (2011)

---

**Table 1: Individual Results of the Test**

<table>
<thead>
<tr>
<th>Participant</th>
<th>Topic</th>
<th>For &amp; Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p/c</td>
<td>c</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>p</td>
</tr>
<tr>
<td>4</td>
<td>p/c</td>
<td>p/c</td>
</tr>
<tr>
<td>5</td>
<td>c</td>
<td>p/c</td>
</tr>
<tr>
<td>6</td>
<td>c</td>
<td>p/c</td>
</tr>
<tr>
<td>7</td>
<td>c</td>
<td>p/c</td>
</tr>
<tr>
<td>8</td>
<td>p/c</td>
<td>p/c</td>
</tr>
</tbody>
</table>

---

**Figure 9. Individual Results of the Test**

<table>
<thead>
<tr>
<th>Participant</th>
<th>Topic</th>
<th>For &amp; Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p/c</td>
<td>c</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>p</td>
</tr>
<tr>
<td>4</td>
<td>p/c</td>
<td>p/c</td>
</tr>
<tr>
<td>5</td>
<td>c</td>
<td>p/c</td>
</tr>
<tr>
<td>6</td>
<td>c</td>
<td>p/c</td>
</tr>
<tr>
<td>7</td>
<td>c</td>
<td>p/c</td>
</tr>
<tr>
<td>8</td>
<td>p/c</td>
<td>p/c</td>
</tr>
</tbody>
</table>

---

**Figure 10. Stage Results Of The Written Test**

<table>
<thead>
<tr>
<th>Participant</th>
<th>VA</th>
<th>IF</th>
<th>While</th>
<th>For &amp; Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>p/c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>2</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>p</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>c</td>
<td>p/c</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>p/c</td>
<td>p/c</td>
<td>c</td>
</tr>
<tr>
<td>5</td>
<td>c</td>
<td>p/c</td>
<td>p/c</td>
<td>c</td>
</tr>
<tr>
<td>6</td>
<td>c</td>
<td>p/c</td>
<td>p/c</td>
<td>c</td>
</tr>
<tr>
<td>7</td>
<td>c</td>
<td>c</td>
<td>p/c</td>
<td>s</td>
</tr>
<tr>
<td>8</td>
<td>c</td>
<td>p/c</td>
<td>p/c</td>
<td>p</td>
</tr>
</tbody>
</table>

---

**Results**

**Written Test**

The individual results of the test for each participant and each exercise are shown in Figure 9. Where a check mark denotes a correct answer.

The topic related stages which were extracted from the re-
sults of the test are shown in Figure 10. s denotes the sensorimotor stage, p the preoperational stage and c the concrete operational stage.

Interview

All transcripts for all participants can be found in the Appendix D. In this section the implications of the observed behavior for the neo-Piagetian stage of the participant is described for each participant.

Participant 1. VA2C001. The participant recognized immediately the relationship between the lines of code by recognizing c to be the temporary variable. Then she deduced the meaning of the variable assignments without the use of concrete values. Seeing relationship between subsequent lines of code and not relying on concrete values are both indicators for concrete operational reasoning (Teague, 2015).

CI2T002. The participant understands the relationship between subsequent lines of code, by recognizing that the code determines the smallest value. In order to see that, the participant needed to be able to use transitive inference (see Exercise Catalog: CI2T002), which requires abstract reasoning on concrete operational level. In addition to that, the participant did not use concrete values but deduced the purpose from the code only by looking at the code, which is also an indicator for concrete operational reasoning (Teague, 2015).

CW2C001. The participant could not see the purpose of the code. In CW2C001 the student needs to understand the relationship between the depicted lines of code in order to determine the purpose of the code. When the purpose of the code is found, the student can create a different implementation. The participant was not able to see this purpose, however she still tried to answer the question by using a strategy, in which it is not required to understand the code. The participant just changed the while-loop into a for-loop, without looking at the purpose of the code. Not being able to see the purpose of the code and not being able to understand the relationship between subsequent lines of code are indicators for preoperational reasoning (Teague, 2015).

LI2C001. The participant chose for one implementation and understood the purpose that each of the lines of code had on the other lines of code. Understanding the relationship of subsequent lines of code and finding out its purpose are a indicators for concrete operational reasoning (Teague, 2015).

Participant 2. VA2C001. The participant saw the relationship between the lines of code by recognizing c to be a ‘helping’ variable. The meaning of the variable assignments were deduced, without the use of concrete values. Seeing relationship between subsequent lines of code and not relying on concrete values are both indicators for concrete operational reasoning (Teague, 2015).

CI2T002. The participant recognizes that the purpose of the code is to print the lowest value. In addition to that she also did not use any concrete value, but deduced the purpose from the code itself. This behavior indicates that she reasons at concrete operational level (Teague, 2015).

CW2C001. The participant could see the purpose of the code partially. She recognized that a name is printed based on the value in amount. However, she did not use this knowledge to create a new implementation. She rather used small adjustments, which do not influence subsequent lines of code, to change the implementation. One could argue, that figuring out the relationship between subsequent lines of code was cognitively so demanding for the participant, that she chose not to do so, although she would be capable of it. This behavior can be interpreted to lie between preoperational and concrete operational reasoning. Given more practice the participant would need less cognitive effort and would be able to understand the relationship between subsequent lines of code easier. So that she would reason at a concrete operational level (Teague, 2015).

LI2C001. The participant chose for one implementation and understood the influence from certain lines of code to other lines of code. Understanding the relationship of subsequent lines of code and finding out its purpose are indicators for concrete operational reasoning (Teague, 2015).

Participant 3. VA2C001. The participant recognized the relationship between the lines of code by recognizing c to be the temporary variable. She deduced the meaning of the variable assignments without the use of concrete values. Seeing relationship between subsequent lines of code and not relying on concrete values are both indicators for concrete operational reasoning (Teague, 2015).

CI2T002. The participant described why the purpose of the code is to print the lowest value. She did not use any concrete value but rather used only the code to come to this conclusion. This behavior indicates that she reasons at concrete operational level (Teague, 2015).

CW2C001. The participant failed to understand the relationship between the subsequent lines of code correctly. Because of that, she was not able to deduce the meaning of the code and could not create a different implementation. Not being able to understand the relationship between subsequent lines of code and not being able to find the purpose of code are indicators for preoperational reasoning (Teague, 2015).

LI2C001. The participant chose for one implementation, but did not understand the relationship between the code choices. Therefore she made code choices, which were not consistent with her chosen implementation. Not being able to see the relationship between subsequent lines of code is an indicator for preoperational reasoning (Teague, 2015).

Participant 4. VA2C001. The participant recognized immediately the relationship between the lines of code by explaining the purpose of each variable. Seeing the relation-
ship between subsequent lines of code and not relying on concrete values are both indicators for concrete operational reasoning (Teague, 2015).

CI2T002. The participant understands how subsequent lines of code influence each other and concludes that the purpose of the code is to determine the smallest value. He needs to be able to use transitive inference in this exercise to come to this conclusion, which requires abstract reasoning on concrete operational level (Teague, 2015).

CW2C001. The participant did not investigate the purpose of the code. He recognized that the while-loop can be substituted with a for-loop and also that both if statements can be condensed in a if-else statement. Since the participant showed a partial understanding on how subsequent lines of code can be transformed to change the implementation, it is concluded that the participant lies between the levels of preoperational and concrete operational reasoning. It can be argued that the participant chose a solution in which he would only need to reason abstractly about the relationship between a few lines of code, because reasoning about more subsequent lines of code would demand a lot cognitive effort.

LI2C001. The participant chose one implementation and explained how this decision influenced his other choices. In order to do so correctly he would need to be able to understand the effects subsequent lines of code have on each other. Understanding the relationship of subsequent lines of code is an indicator for concrete operational reasoning (Teague, 2015).

Participant 5. VA2C001. The participant explained the purpose of each variable, as well as its influence on other variables. Recognizing the relationship between subsequent lines of code and seeing the purpose of code just by looking at it are indicators for concrete operational reasoning (Teague, 2015).

CI2T002. The participant deduced the purpose of the code without needing to pluck in concrete values. This is an indicator for concrete operational reasoning (Teague, 2015).

CW2C001. The participant determined the purpose of the given code. Then he created a different implementation that would fit the determined purpose. In order to do so he needs to be able to apply the concept of conversion as well as recognize the relationship between subsequent lines of code, which are both indicators for concrete operational reasoning (Teague, 2015).

LI2C001. The participant did not understand the purpose of the first statement and its influence on the subsequent lines of code. Not understanding how subsequent lines of code influence each other is an indicator for preoperational reasoning (Teague, 2015).

Participant 6. VA2C001. The participant’s strategy was to first think of a way to implement a switch himself. After that he chose for a solution that resembled his implementation. He also recognized that there are two solutions and gave the other. Being able to construct small programs, without trial and error, is an indicator for concrete operational reasoning (Teague, 2015).

CI2T002. The participant deduced the meaning of the code correctly, just by looking at the code, which is an indicator for concrete operational reasoning (Teague, 2015).

CW2C001. The participant recognized that the while loop can be substituted with the for loop and also which influence that would have on the purpose, that a name is printed for a the amount specified. Recognizing the relationship between lines of code and being able to give a different implementation for a given purpose are considered indicators for concrete operational reasoning (Teague, 2015).

LI2C001. The participant recognized all implications that the chosen implication had for all the given code choices. In order to do so he had to be able to see the relationship between subsequent lines of code, which is an indicator for concrete operational reasoning (Teague, 2015).

Participant 7. VA2C001. The participant recognized the role of variable c to be the temporary variable, that was needed for a switch statement. In addition to that he recognized the purpose behind the assignment of a, b and c, without the use of concrete values. Being able to see the purpose of code without relying on specific values is an indicator for concrete operational reasoning (Teague, 2015).

CI2T002. The participant determined the meaning of the code by examining the effects the subsequent lines of code had on each other. He did not use concrete values to determine this meaning. Both of those programming behaviors are indicators for concrete operational reasoning (Teague, 2015).

CW2C001. The participant did not investigate the purpose of the code, but rather used a strategy, in which he can change an implementation without needing to understand the meaning between subsequent lines of code. He thus only centered his focus on specific lines of code rather than seeing the big picture. This behavior, known as spatial centration is an indicator for preoperational reasoning (Teague, 2015).

LI2C001. The participant failed to recognize the influence of the first statement on the subsequent lines of code. Not understanding how subsequent lines of code influence each other can be seen as indicator for preoperational reasoning (Teague, 2015).

Participant 8. VA2C001. When asked about his reasoning, the participant recognized which variable implemented which functionality. He recognized that c denotes the temp variable: "So c = a, you want to switch a and b, so then I did a = b and then b gets the value of a which was in c, so c = b". In this small problem the participant showed he could work with the concept of conservation "[...]
You can choose both of them but you need to be consistent". He could also see the relationship between code, by identifying the correct order and need for a temporary variable.
This behavior indicates that the participant is for code, which only includes variable assignments, capable to reason at a concrete operational level (Teague, 2015).

CI2T002. The participant showed, that he could not see a clear meaning in the code and in relationships between lines of code: "Most of the time something is checked, that one is bigger than the other". From only looking at specific statements he made a guess about the global purpose of the code: "[...] So I had the idea that it had something to do with the order, so to order those variables. [...] it could be something different. But I think the general idea has something to do with order. Not seeing meaning in code and relationship between lines of code is typical behavior for students who reason at a preoperational level (Teague, 2015).

CW2C001. The participant did not try to understand the purpose of the code. In this exercise the participant was asked to write a different implementation for a program. His strategy was to go through every line and see if he could substitute something in the code, without understanding what the code does. This strategy might indicate that for the participant it takes a lot of cognitive effort to process the meaning of code. It also shows that the participant focuses on each line of code separately without seeing the bigger picture. Those ways of reasoning are predominantly applied at a preoperational level (Teague, 2015).

LI2C001. The participant could not find the meaning of the relationship between the given lines of code and used a similar strategy to the one he used for CW2C001. He decided to use a strategy in which he would only need to look at one line of code at a time: "You had to be consistent, well I thought in one of those options you always use something with get and with the other ones you don’t. [...]So I always chose for those which used the get. [...]for example for those two (options), this line has more gets than this one so I chose the one with more gets". Focusing on each line of code separately without seeing the bigger picture and not seeing the purpose of relationships between code, is behavior which is mostly applied at a preoperational level (Teague, 2015).

Summary. The determined stages from the interview for all participants are shown in Figure 11. In this figure, preoperational reasoning for a given topic is denoted with a p. Concrete Operational reasoning is denoted by c. If the participant showed types of preoperational together with concrete operational reasoning, he or she lies between those stages. This is denoted in the figure with p / c.

Comparison

A comparison between the categorizations of the test and the interview are shown per topic in the Figures 12 - 15. The similarity is underlined through the use of background colors. The color green indicates the exact same stage result, yellow indicates, that the different of stages is not greater than a half stage and red indicates that the stages determined through written test and interview differ by minimum one stage.
**Analysis**

**Written Test**

Participant 3 was not able to answer at least half of the level one exercises for Variable Assignment correctly, but could answer the level two exercise VA2C001. This should not be possible, since VA2C001 requires that the student is able to handle the concept of conservation, which novices at the sensorimotor stage should not be able to do. One reason for this result, could be that the categorization of the novice as sensorimotor was wrong and his low success rate for tracing exercises depended on the specific set of level one exercises. It could be, that if the novice had done more exercises he or she would eventually finish more than half of them correctly and therefore would be classified as non-sensorimotor. Another explanation is, that the novice is in fact sensorimotor and guessed the correct solution to the exercise. The chance to guess the correct answer of VA2C001, lies relatively high at 25%. Instructors should be aware of this high chance when using this particular exercise in their test.

The other observed case of sensorimotor behavior was conform to the literature. Participant 7 was classified as sensorimotor for the topic For & Lists and was thus not able to answer a single level two exercise of this topic correctly.

Analyzing the distribution of the observed neo-Piagetian stages, it stands out that only two participants were classified as sensorimotor from the test. Most participants showed concrete operational behavior for various topics (see Figure 10). On first view, this contradicts the observation, that programming students in their first year are having problems with concepts of programming (McCracken et al., 2001). The observations in this case study can be explained by the set of tested participant. Each of those participant had already successfully passed to programming courses. It shows that a requirement for successfully passing these courses was apparently to be able to understand programming concepts at least at a preoperational level.

**Interview**

By analyzing the observed behavior during the interviews it appears that even the preoperational students, did not use tracing in order to infer the purpose of code. One strategy of preoperational novices to determine the purpose of code, is by choosing a set of input values and trace the code mechanically. After that the novice tries to deduce the purpose of code by looking how the input values are transformed into the output of the program (Teague, 2015).
other strategies, which are based on tracing were not observed in this case study. If the code was too complex to deduce its meaning only from it, the participants described strategies to answer the question, in which they did not need to understand all lines of code, but rather small parts of code. This behavior to only focus on some lines of code rather than on the whole picture, is called spatial centration (Teague, 2015). Students can cope with spatial centration by using code traces to infer the meaning of code. As described, the novices in this study did not choose this tracing strategy but found other creative strategies in order to cope with spatial centration. For example students created a different implementation for a given specification (concept of conservation) through the use of simple transformations of existing code (changing a while-loop into a for-loop), in which they did not need to understand the relationship between subsequent lines of code.

Since the traversal between stages happens in a continuous manner (Overlapping Waves Model, see Figure 3), it was difficult to determine a discrete neo-Piagetian stage for the shown behavior. For example Participant 2 showed that he could in fact determine the purpose of the code correctly, but did also signs of facing spatial centration. In this case study, this observed behavior was seen as an indicator for lying between the preoperational and concrete operational stage, which was encoded as the discreet value p/c. It can not be exactly determined where the student lies exactly between those stages. Is he more close to preoperational or to concrete operational reasoning? Since the results are encoded into discreet values, it can not be differentiated between a student who is more close to preoperational and a student who is more close to concrete operational reasoning behavior (both are classified as p/c). However, in both cases the implications for the instructor are the same. The student should be confronted with exercises, which require preoperational reasoning behavior, in order to traverse to concrete operational reasoning behavior.

Comparison

With the help of Figure 16 it can be investigated, whether a participant, who did not answer level one questions reliably for one topic, was able to answer level two questions. A check mark in the row of level one indicates, that the participant did answer at least 50% of the level one exercises for this topic correctly. On the other hand a check mark in the row of level two indicates, that the participant did answer at least one level two question correctly. If the relation between the answers of the level one and level two questions, adheres to the theory their marks are colored green, else they are colored red.

The claim of Lister (2011), which is that students need to be able to trace code reliably in order to reason about code, holds for the collected data with one exception. Participant three could not trace reliably for the topic of Variable Assignment in the written test, but was able to show a form of concrete operational reasoning for this topic in the interview. One explanation for this, could be that the novice is not sensorimotor and his or her low success rate for tracing exercises is based on the specific set of questions. It could be, that if the novice had done more exercises he or she would eventually finish more than half of them correctly and therefore would be classified as non-sensorimotor. Another explanation is, that the participant is in fact sensorimotor and just retold the explanation of a switch statement that he heard prior, without understanding it.

The accuracy of the written test resulting in the exact stages as the interview is shown in Figure 17 per topic. The proposed method does not determine the exact level of reasoning precisely (accuracy: 62,5%). However, with the proposed method the level of reasoning can be approximated most of the time (accuracy: 90,6%).

The low accuracy obtained when testing for the topic while-loops can be explained as follows. The level two assignment CW2C001 could be completed by novices, which would reason at a preoperational level, when they made use of a certain strategy. This strategy was to make small changes, which did not require them to understand the purpose of the code. In this cases the transformation from a while-loop to a for-loop and the transformation from an if-else structure to an if-structure.

Through this creative strategy they were able answer correctly, whereas they could not name the purpose of the code in the interviews. Comparable strategies are often used by novices at the preoperational level, see for example participant 8’s attempt to answer LI2C001 in the section Results - Written Test - Participant 8. The application of those strategies makes it harder to determine the abilities of a novice, since they have learned through the years of their education, to guess answers which would score them the most points. Teague (2015) also discovered instances of this problem and thus concluded, that one should always let the novices verbalize their thought processes while doing an exercise. While this might be the most straightforward solution, in most contexts the instructor does not have the resources to do so. Another way of dealing with this problem, is to be aware, that some exercises are prone to those kind of strategies (like CW2C001) and to discard those exercises or use them with caution.

Conclusion

RQ1. Are the results of written tests on neo-Piagetian stages in the case study conform to the expected results described in literature?

The participants of the case study performed on average better than it was expected from literature for first year programming students. One reason for this might be that all
participants had successfully passed two prior programming courses. The relationship between correctly answered level one exercises and level two exercises are conform with the expected results from literature with one exception. Reasons for this exception could be connected to the specific small set of level one exercises (VA1M007, VA1M008, VA1M009) or that the specific level two exercise (VA2C001) is not suited to classify concrete operational reasoning behavior.

RQ2. Is the behavior observed in the interview conform to neo-Piagetian theory?

The preoperational participants did not try to deduce the purpose of code through the use of tracing, as described in the literature (Teague, 2015). However, they showed behavior, which indicated that the understanding of multiple lines of code took them a lot of cognitive effort. The coping with this problem, namely finding a strategy in which multiple lines of code need not to be understood, was observed in is conform to the literature (Teague, 2015).

RQ3. Do the neo-Piagetian stages determined by the written tests correspond to the neo-Piagetian stages determined by the interviews?

The neo-Piagetian from the written test resembles the exact neo-Piagetian stage from the interview with an accuracy of 63%. One explanation for this low percentage is that it could have factored in, that a novice can have attributes of more than one stage at a time (see Overlapping Waves Model, Figure 3). This would have made it harder to determine his or her exact stage. Taking this into account and allowing a leeway of half a stage, the accuracy increases to 91%.

Discussion

Implications

In the case study, it was found that students could use strategies which would allow them to answer the level two exercise CW2C001 correctly on paper even though they would reason on a preoperational level. In order to assure a higher accuracy in predicting the neo-Piagetian stage through a written test, this particular exercise should be removed from the exercise catalog. In addition to that, instructors should investigate whether exercises can be answered through the application of those strategies, when creating written tests to determine the neo-Piagetian stage of a student. If they would find, that a particular exercise is prone to those kind of strategies, they should remove them from the test.

It was also found, that preoperational students do not necessarily use tracing in order to find the purpose of code. They might also apply other strategies. The strategies observed in the case study, always had to do with answering a question without seeing the relationship between subsequent lines of code. To avoid the possible application of those strategies, one should create level 2 exercises in a way, that the correct solution can only be found, when the relation of subsequent lines of code has occurred correctly.

Further it was found that the construction of written tests from the exercise catalog, results in a distribution of neo-Piagetian stages, which are conform to the literature (level 2 exercises can not be answered by preoperational students).

In summary this case study shows one example on how Lister’s theory can be utilized in order to determine the neo-Piagetian stage of a novice. The described method implies the following: First existing or newly written exercises are categorized using the proposed categorization method. Then a written test is created out of those exercises for the relevant topics for the students. The results of this written test are then analyzed and categorized in the proposed manner in order to receive the neo-Piagetian stage of the students. With that information, students can be helped individually with exercises and explanations corresponding to their reasoning level (see Background - Practical Usage Of The Neo-Piagetian Stages).

Limitations Of The Study

In this study, only a small amount of questions was used per topic, in order to keep the maximum examination time at two hours. It is open to question whether the increase of the amount of questions per topic would influence the results.

Another limitation of this study is the low number of participants (N = 8). Since this study is qualitative, this number is sufficient to gain new insights (see Discussion - Implications) about the proposed approach. However, this low number of participants harms the generalizability of the findings.

In this case study, the results from the written test and interviews where compared. In this comparison it was assumed, that the “correct” neo-Piagetian stage of a participant was determined through observation of stage specific behavior during the interview, which was previously defined by Teague (2015). Thus the applicability of the use of solely written tests to determine the neo-Piagetian stage, was determined by investigating to what extent the result of the written test reflect the result of the interview. One limitation of this research is, that it is not certain, that the stage result from the interview is in fact the actual neo-Piagetian stage of the observed participant. This method was chosen, because in literature it is stated, that novices of each stage show different programming behavior. The observation of this behavior was used in literature to determine their neo-Piagetian stage. However, there is no guaranteed way of knowing and checking, whether the concluded neo-Piagetian stage is in fact the stage of the novice. It could also be the case, that the observant made a mistake in categorizing the shown behavior, to be the described distinguishing behavior by (Teague, 2015).

In addition to that, only one observer was used in the case
study, to classify the behavior into the corresponding neo-Piagetian stage. It was not validated, whether those decisions might be subjective to the observer and would vary for different observers. In the literature there is no indication, that those decisions might differ between observers. However, it is a possibility and should be investigated in future research.

Future Research

One compelling question, which could be covered in future research is: Does this method hold up when different programming paradigms are used? According to Lister (2011), the different levels of reasoning are programming paradigm independent. The categorization method, proposed in this paper should also be paradigm independent, since an exercise for a functional programming language can also be categorized into two levels. It would be interesting to find out whether those questions could also approximate the neo-Piagetian stage in a similar way.

Another way this method could be used by future research is by testing novices with different exercises. In order to keep it practical for the novice, the time necessary to complete the test in this research should not be longer than one hour. It would be interesting to see, if a higher accuracy could be obtained when more exercises are used.

This research uncovered the problem of preoperational strategies which make it seem like the novice reasons at a higher level than he actually is. One could try to find a way to predict, which exercises are prone to those strategies.

It would also be fascinating to research whether the learning results of students would actually improve when their instructors use the proposed method.

References


A Appendix: Exercise Catalog
# 1 Variable Assignment

## 1.1 Level 1

<table>
<thead>
<tr>
<th>ID: VA1M001</th>
<th>Topic: Variable Assignment</th>
<th>Level 1</th>
<th>Misconception: Assignment from left to right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1) int f;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f = 2;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2) int f;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 = f;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Question:</td>
<td></td>
<td></td>
<td>After the execution of which of the above parts of code will the variable ( f ) contain the value 2?</td>
</tr>
<tr>
<td>Choices:</td>
<td></td>
<td></td>
<td>a) (1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>b) (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>c) (1) and (2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID: VA1M002</th>
<th>Topic: Variable Assignment</th>
<th>Level 1</th>
<th>Misconception: Expression is not evaluated when assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code:</td>
<td></td>
<td></td>
<td>int f = 7 + 3;</td>
</tr>
<tr>
<td>Question:</td>
<td></td>
<td></td>
<td>Give the value of the variable ( f ) after the above code is executed.</td>
</tr>
<tr>
<td>ID: VA1M003</td>
<td>ID: VA1M004</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Topic: Variable Assignment</td>
<td>Topic: Variable Assignment</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level 1</td>
<td>Level 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Misconception: Assignment over-rides previous value</td>
<td>Misconception: Assignment of value of variable $A$ to $B$, empties $A$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Code:</strong></td>
<td><strong>Code:</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| int $f = 2$;  
$f = 3$; | int $f = 5$;  
int $a = f$; |
| **Question:** Give the value of the variable $f$ after the above code is executed. | **Question:** Give the value of the variables $f$ and $a$ after the above code is executed. |
| **Choices:**  
a) $f$ is 2;  
b) $f$ is [2,3];  
c) $f$ is 3; | **Choices:**  
a) $a$ is 5; $f$ is 5  
b) $a$ is 5; $f$ is 0;  
c) $a$ is 'f'; $f$ is 5; |

<table>
<thead>
<tr>
<th>ID: VA1M005</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic: Variable Assignment</td>
</tr>
<tr>
<td>Level 1</td>
</tr>
<tr>
<td>Misconception: The <em>empty string</em> is not a value</td>
</tr>
</tbody>
</table>
| **Code:**  
String $f = "Hello World!"$;  
f = ""; |
<p>| <strong>Question:</strong> After the above code was executed, what is the value of $f$? |</p>
<table>
<thead>
<tr>
<th>ID: VA1M006</th>
<th>ID: VA1M007</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Topic:</strong> Variable Assignment</td>
<td><strong>Topic:</strong> Variable Assignment</td>
</tr>
<tr>
<td><strong>Level:</strong> 1</td>
<td><strong>Level:</strong> 1</td>
</tr>
<tr>
<td>Misconception: Assignment of value of variable $A$ to $B$, links $A$ to $B$</td>
<td>Misconception: Variables need no initialization</td>
</tr>
<tr>
<td><strong>Code:</strong></td>
<td><strong>Code:</strong></td>
</tr>
<tr>
<td>int $a$; int $b$; $a = 3$; $b = a$; $a = 4$;</td>
<td>int $f$; $f = f + 4$;</td>
</tr>
<tr>
<td><strong>Question:</strong> After the above code was executed, what are the values of $a$ and $b$?</td>
<td><strong>Question:</strong> After which piece of code will the variable $f$ contain the value 4?</td>
</tr>
<tr>
<td><strong>Choices:</strong> a) (1) b) (2) c) (1) and (2)</td>
<td><strong>Choices:</strong> a) (1) b) (2) c) (1) and (2)</td>
</tr>
</tbody>
</table>
Misconception: Assignments are carried out in parallel

Code:

```java
int width = 0;
int height = 0;
int area = width * height;
width = 4;
height = 4;
System.out.println(area);
```

Question:
After the above code is executed, what is shown in the console?

---

ID: VA1M009
Topic: Variable Assignment
Level 1

Code:

```java
int a = 0;
int b;
int c = 10 + 5;
int d = 23;
int e = 4;
b = c;
c = a;
a = b;
e = c + 3;
d = e;
c = d;
```

Question:
After the above code was executed, what are the values of a, b, c, d and e?
1.2 Level 2

<table>
<thead>
<tr>
<th>ID: VA2R001</th>
<th>ID: VA2C001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic: Variable Assignment</td>
<td>Topic: Variable Assignment</td>
</tr>
<tr>
<td>Level 2</td>
<td>Level 2</td>
</tr>
<tr>
<td>Theme: Reversing</td>
<td>Theme: Conservation</td>
</tr>
</tbody>
</table>

Code:

| a = a + b + c; |
| b = b + c + d; |
| c = c + d + a; |
| d = d + a + b; |

Question:

a, b, c and d are integers and initialized properly. Write code that undoes the effect of the above code.

Code:

| □ c = a; or □ c = b; |
| □ a = b; or □ b = a; |
| □ a = c; or □ b = c; |

Question:

a, b and c are integers and initialized properly. The program is normally executed from top to bottom. You have to make a choice for each line of code, whether the left or the right statement should be executed (tick the box). The goal is to switch the values of a and b.
2  IF Statement

2.1  Level 1

ID: CI1M001
Topic: IF-Statement, Variable Assignment
Level 1
Misconception: If triggers "whenever" condition applies.

Code:
```java
int a = 9;
if (a == 10) {
    System.out.print("first ");
}
System.out.print("second ");
a = 10;
```

Question:
After the above code is executed, what is shown in the console?

ID: CI1M002
Topic: IF-Statement, Variable Assignment
Level 1
Misconception: Computer understands intention of code.

Code:
```java
Scanner scanner = new Scanner(System.in);
System.out.print("Enter an integer: ");
int userInput = scanner.nextInt();
if (userInput > 5) {
    System.out.print("Big Number! ");
}
System.out.print("Small Number! ");
```

Question:
In the above code, scanner.nextInt() reads the next Integer from the console. What is shown in the console after the above code is executed and the user gave 6 as input?
2.2 Level 2

ID: CI2R001
Topic: IF-Statement, Variable Assignment
Level 2
Theme: Reversing

Code:
```plaintext
if (a > b) {
    e = e + 1;
} else if (a < b) {
    e = e - 1;
}
if (b < c) {
    e = e - 1;
} else if (b > c) {
    e = e + 1;
}
```

Question:
a, b, c and e are well initialized Integers. Write code that undoes the above effect for the variable e. Thus if your code gets executed after the above code, e should contain the same values as it had before the above code was executed.

ID: CI2T001
Topic: IF-Statement, Variable Assignment
Level 2
Theme: Transitive Inference

Code:
```plaintext
if (y1 < y2) {
    t = y1;
    y1 = y2;
    y2 = t;
}
if (y2 < y3) {
    t = y2;
    y2 = y3;
    y3 = t;
}
if (y1 < y2) {
    t = y1;
    y1 = y2;
    y2 = t;
}
```

Question:
Assume that the variables y1, y2 and y3 contain integer values and t is initialized properly. In one sentence, describe the purpose of the above code.
ID: CI2T002
Topic: IF-Statement
Level 2
Theme: Transitive Inference

Code:
```java
if (a > b) {
    if (b > c) {
        System.out.print(c);
    } else {
        System.out.print(b);
    }
} else if (a > c) {
    System.out.print(c);
} else {
    System.out.print(a);
}
```

Question:
In one sentence that you should write in the box below, describe the purpose of the above code. Do NOT give a line-by-line description of what the code does. Instead, tell the purpose of the code.
## 3 WHILE-Loops

### 3.1 Level 1

<table>
<thead>
<tr>
<th>ID: CW1M001</th>
<th>Topic: WHILE-Loop, Variable Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td></td>
</tr>
<tr>
<td>Misconception: While loops stop immediately when condition no longer applies.</td>
<td></td>
</tr>
</tbody>
</table>

**Code:**

```java
int counter = 0;
while (counter < 2) {
    System.out.print("start ");
    counter = counter + 1;
    System.out.print("end ");
}
```

**Question:** After the above code is executed, what is shown in the console?

<table>
<thead>
<tr>
<th>ID: CW1M002</th>
<th>Topic: WHILE-Loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td></td>
</tr>
<tr>
<td>Misconception: <code>do-while</code> evaluates its expression before the first run.</td>
<td></td>
</tr>
</tbody>
</table>

**Code:**

```java
int count = 4;
do {
    count = count + 1;
} while (count < 4)
System.out.println(String.valueOf(count));
```

**Question:** After the above code is executed, what is shown in the console?
ID: CW2T001
Topic: WHILE-loop, Lists, Variable Assignment
Level 2
Theme: Transitive Inference

Code:
```java
bool bValid = true;
int counter = 0;
while (counter < iNumbers.size()) {
    if (iNumbers.get(counter) > iNumbers.get(counter+1)) {
        bValid = false;
    }
    counter = counter + 1;
}
```

Question:
iNumbers denotes a List. Explain the purpose of the given code.
<table>
<thead>
<tr>
<th>ID: CW2C001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic: WHILE-loop, Variable Assignment</td>
</tr>
<tr>
<td>Level 2</td>
</tr>
<tr>
<td>Theme: Conservation</td>
</tr>
</tbody>
</table>

**Code:**

```java
void greeting(int amount, String name) {
    int i = 0;
    amount = amount * 2;
    while (i < amount) {
        if (i % 2 == 0) {
            System.out.print("Hello ");
        }
        if (i % 2 != 0) {
            System.out.println(name);
        }
        i = i + 1;
    }
}
```

**Question:**

Write a method, which generates the same output on the console as the given method but has a different implementation.
4 Lists & FOR-Loops

4.1 Level 1

<table>
<thead>
<tr>
<th>ID: LI1M001</th>
<th>Topic: List</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Misconception: Array-Index-Count starts at one.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code:</th>
</tr>
</thead>
<tbody>
<tr>
<td>List&lt;Integer&gt; numbers = Arrays.asList(32, 25, 7, 9, 5);</td>
</tr>
<tr>
<td>System.out.println(String.valueOf(numbers.get(1)));</td>
</tr>
</tbody>
</table>

| Question: |
| After the above code is executed, what is shown in the console? |

<table>
<thead>
<tr>
<th>ID: LI1O001</th>
<th>Topic: List, for-loop, Variable Assignment</th>
<th>Level 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>List&lt;Integer&gt; losses = Arrays.asList(1, 25, 4, 9, 16);</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int net = 100;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>for (Integer loss : losses) {</td>
<td></td>
<td></td>
</tr>
<tr>
<td>net = net - loss;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System.out.println(String.valueOf(net));</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Question: |
| After the above code is executed, what is shown in the console? |
### Level 1

**Code:**

```java
List<Integer> a = Arrays.asList(1, 1, 2);
List<Integer> b = Arrays.asList(4, 2, 1);
int last = -1;
for (int i = 0; i < a.size(); i++){
    if (a.get(i) < b.get(i)) {
        last = i;
    }
}
System.out.println(last);
```

**Question:**
After the above code is executed, what is shown in the console?

---

### Level 2

**Code:**

```java
int temp = x.get(x.size()-1);
for (int i = x.size()-2; i >= 0; --i) {
    x.set(i+1, x.get(i));
}
x.set(0, temp);
```

**Question:**
Assume x is an ArrayList. Write code that undoes the effect of the above code. That is, write code to move all elements of the ArrayList x one place to the left, with the leftmost element being moved to the rightmost position, using only the get, set and size method of ArrayList.
The code shows a method which returns the smallest value in the ArrayList \( x \). The code scans across the array, using the variable "minsofar" to remember the smallest value seen thus far. There are two ways to implement remembering the smallest value: remember the actual value, or remember the values position in the array. In the code you have to make four choices, which should all implement the first or the second way of implementation (it does not matter which implementation you choose, the task is to be consistent with the chosen implementation for all of your choices).
<table>
<thead>
<tr>
<th>ID: LI2T001</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic: List, FOR-loop, Variable Assignment</td>
</tr>
<tr>
<td>Level 2</td>
</tr>
<tr>
<td>Theme: Transitive Inference</td>
</tr>
</tbody>
</table>

**Code:**

```java
List<Integer> c = new ArrayList<Integer>();
List<Integer> d = new ArrayList<Integer>();
List<Integer> e = new ArrayList<Integer>();
for (int i = 0; i < a.size(); i++) {
    if (a.get(i) == b) {
        c.add(a.get(i));
    } else if (a.get(i) > b) {
        d.add(a.get(i));
    } else {
        e.add(a.get(i));
    }
}
```

**Question:**
Assume a is well initialized ArrayList and b is a well initialized Integer. Describe the relationship between the values of c and e.
ID: LI2T002
Topic: List, FOR-loop, Variable Assignment
Level 2
Theme: Transitive Inference

Code:
    bool bValid = true;
    for (int i = 0; i < iNumbers.size() - 1; i++)
    {
        if (iNumbers.get(i) > iNumbers.get(i+1)) {
            bValid = false;
        }
    }

Question:
iNumbers denotes a List. Explain the purpose of the given code.
B Appendix: Experiment Exercises
Practical Neo-Piagetian Theory QUIZ

Thank you for taking this quiz :) Taking this quiz will help you to identify gaps in your programming knowledge and will help the teaching staff to enhance their effort of giving the best possible instructions to help struggling novice programmers.
Your email address will only be used to send you the results of this quiz. For more information about how the data and personal information is used, please feel free to contact me: tim.kutscha@student.ru.nl.

email:

s-number:

I allow that my quiz results will be compared with the grade I obtain in this course:
□ Yes
□ No

Gender:
□ male
□ female
□ other

Age:
1 Part I

Code:

(1)
```java
int f;
f = f + 4;
```

(2)
```java
int f = 0;
f = f + 4;
```

Question:
After which piece of code will the variable `f` contain the value 4?

Choices:
a) (1)
b) (2)
c) (1) and (2)

Code:
```java
int width = 0;
int height = 0;
int area = width * height;
width = 4;
height = 4;
System.out.println(String.valueOf(area));
```

Question:
After the above code is executed, what is shown in the console?

Answer:
Code:
```c
int a = 0;
int b;
int c = 10 + 5;
int d = 23;
int e = 4;
b = c;
c = a;
a = b;
e = c + 3;
d = c;
c = d;
```

Question:
After the above code was executed, what are the values of $a$, $b$, $c$, $d$ and $e$?

Answer:

Code:
```c
□ c = a; or □ c = b;
□ a = b; or □ b = a;
□ a = c; or □ b = c;
```

Question:
a, $b$ and $c$ are integers and initialized properly. The program is normally executed from top to bottom. You have to make a choice for each line of code, whether the left or the right statement should be executed (tick the box). The goal is to switch the values of $a$ and $b$. 

3
Code:
\[
a = a + b + c; \\
b = b + c + d; \\
c = c + d + a; \\
d = d + a + b;
\]

Question:
a, b, c and d are integers and initialized properly. Write code that undoes the effect of the above code. That is, after your code is executed the values of a, b, c and d are the same as before the above code was executed.

Answer:

2 Part II

Code:
```java
int a = 9;
if (a == 10) {
    System.out.print("first ");
}
System.out.print("second ");
a = 10;
```

Question:
After the above code is executed, what is shown in the console?

Answer:
Code:

Scanner scanner = new Scanner(System.in);
System.out.print("Enter an integer: ");
int userInput = scanner.nextInt();
if (userInput > 5) {
    System.out.print("Big Number!");
}
System.out.print("Small Number!");

Question:
In the above code, scanner.nextInt() reads the next Integer from the console. What is shown in the console after the above code is executed and the user gave 6 as input?

Answer:
Code:
if (a > b) {
    e = e + 1;
} else if (a < b) {
    e = e - 1;
}
if (b < c) {
    e = e - 1;
} else if (b > c) {
    e = e + 1;
}

Question:
a, b, c and e are well initialized Integers. Write code that undoes the above effect for the variable e. Thus if your code gets executed after the above code, e should contain the same values as it had before the above code was executed.

Answer:
Code:

```java
if (a > b) {
    if (b > c) {
        System.out.println(String.valueOf(c));
    } else {
        System.out.println(String.valueOf(b));
    }
} else if (a > c) {
    System.out.println(String.valueOf(c));
} else {
    System.out.println(String.valueOf(a));
}
```

**Question:**
In one sentence that you should write in the box below, describe the purpose of the above code. Do NOT give a line-by-line description of what the code does. Instead, tell the purpose of the code.

**Answer:**
3 Part III

<table>
<thead>
<tr>
<th>Code:</th>
</tr>
</thead>
</table>
| int counter = 0;
while (counter < 2) {
   System.out.print(" start ");
   counter = counter + 1;
   System.out.print(" end ");
}

<table>
<thead>
<tr>
<th>Question:</th>
</tr>
</thead>
<tbody>
<tr>
<td>After the above code is executed, what is shown in the console?</td>
</tr>
<tr>
<td>Answer:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code:</th>
</tr>
</thead>
</table>
| int count = 4;
do {
   count = count + 1;
} while (count < 4)
System.out.println(String.valueOf(count));

| Question: | After the above code is executed, what is shown in the console? |
| Answer: |
Code:
bool bValid = true;
int counter = 0;
while (counter < iNumbers.size()) {
    if (iNumbers.get(counter) > iNumbers.get(counter + 1)) {
        bValid = false;
    }
    counter = counter + 1;
}

Question:
iNumbers denotes a List. Explain the purpose of the given code.

Answer:
Code:

```java
void greeting(int amount, String name) {
    int i = 0;
    amount = amount * 2;
    while (i < amount) {
        if ((i%2)== 0) {
            System.out.print("Hello ");
        } else {
            System.out.println(name);
        }
        i = i + 1;
    }
}
```

Question:
Write a method, which generates the same output on the console as the given method but has a different implementation.

Answer:
4 PART IV

<table>
<thead>
<tr>
<th>Code:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>List&lt;Integer&gt; numbers = Arrays.asList(32, 25, 7, 9, 5); System.out.println(String.valueOf(numbers.get(1)));</code></td>
</tr>
<tr>
<td>Question:</td>
</tr>
<tr>
<td>After the above code is executed, what is shown in the console?</td>
</tr>
<tr>
<td>Answer:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>List&lt;Integer&gt; losses = Arrays.asList(1, 25, 4, 9, 16); int net = 100; for (Integer loss : losses) { net = net - loss; } System.out.println(String.valueOf(net));</code></td>
</tr>
<tr>
<td>Question:</td>
</tr>
<tr>
<td>After the above code is executed, what is shown in the console?</td>
</tr>
<tr>
<td>Answer:</td>
</tr>
</tbody>
</table>

11
Question:
After the above code is executed, what is shown in the console?

Answer:

```
int temp = x.get(x.size() - 1);
for (int i = x.size() - 2; i >= 0; --i) {
    x.set(i + 1, x.get(i));
}
x.set(0, temp);
```

Question:
Assume x is an ArrayList. Write code that undoes the effect of the above code. That is, write code to move all elements of the ArrayList x one place to the left, with the leftmost element being moved to the rightmost position, using only the get, set and size method of ArrayList.

Answer:
Question:
The code shows a method which returns the smallest value in the ArrayList `x`. The code scans across the array, using the variable "minsofar" to remember the smallest value seen thus far.
There are two ways to implement remembering the smallest value: remember the actual value, or remember the values position in the array. In the code you have to make four choices, which should all implement the first or the second way of implementation (It does not matter which implementation you choose, the task is to be consistent with the chosen implementation for all of your choices).
Appendix: Interview Exercises
1  Part I

Code:
□ c = a; or □ c = b;
□ a = b; or □ b = a;
□ a = c; or □ b = c;

Question:
a, b and c are integers and initialized properly. The program is normally executed from top to bottom. You have to make a choice for each line of code, whether the left or the right statement should be executed (tick the box). The goal is to switch the values of a and b.

2  Part II

Code:
if (a > b) {
  if (b > c) {
    System.out.println(String.valueOf(c));
  } else {
    System.out.println(String.valueOf(b));
  }
} else if (a > c) {
  System.out.println(String.valueOf(c));
} else {
  System.out.println(String.valueOf(a));
}

Question:
In one sentence that you should write in the box below, describe the purpose of the above code. Do NOT give a line-by-line description of what the code does. Instead, tell the purpose of the code.

Answer:
3 Part III

Code:

```java
void greeting(int amount, String name) {
    int i = 0;
    amount = amount * 2;
    while (i < amount) {
        if ((i%2)== 0) {
            System.out.println("Hello ");
        }
        if ((i%2)!= 0) {
            System.out.println(name);
        }
        i = i + 1;
    }
}
```

Question:
Write a method, which generates the same output on the console as the given method but has a different implementation.

Answer:
4 PART IV

Code:

```java
class
public int min(ArrayList<Integer> x) {
    int minsofar = 0;
    int minsofar = x.get(0);
    for (int i = 1; i < x.size(); i++) {
        if (x.get(i) < minsofar) {
            if (x.get(i) < x.get(minsofar)) {
                minsofar = i;
                minsofar = x.get(i);
            }
        }
    }
    return minsofar;
    return x.get(minsofar);
}
```

Question:
The code shows a method which returns the smallest value in the ArrayList x. The code scans across the array, using the variable "minsofar" to remember the smallest value seen thus far.

There are two ways to implement remembering the smallest value: remember the actual value, or remember the values position in the array. In the code you have to make four choices, which should all implement the first or the second way of implementation (It does not matter which implementation you choose, the task is to be consistent with the chosen implementation for all of your choices).
D Appendix: Interview Transcripts
1 Subject 1

1.1 Exercise VA2C001

Thank you for taking the time to answer the question. Here are some of the
questions you just answered. I now would like to know your reasoning and
your plan of action as you were confronted with this exercise.
You had to switch a and b and you had c as a temporary variable. Therefore
I began, with c = a, so that a = b and later b = c, because c contains the
value of a.

1.2 Exercise CI2T002

What were your thoughts on this exercise?
Here the code determines the smallest value, from all three variables. For
example you see here that b < a and if b < c then b is logged. And here the
same thing happened (points at lower part of code)
Did you try this out with concrete values or did you only use the code to
determine what it does?
I saw it just by looking at the code.

1.3 Exercise CW2C001

What were your thoughts on this exercise?
Here I didn’t really knew what was going on. So I just replaced the while
loop with a for loop. And replaced the second if statement with an else
statement. Because if the first if statement is false, the second if statement
is true.
What is the purpose of the function? So the first time ”hello” is printed.
Then it prints the name. It depends on what amount ..wait I guess it doesn’t
even print ”hello”, because amount does not change.
Amount times two mean it becomes an even number, so I guess this statement
holds true. Because the name is not printed but I would do something with
the name so that hello gets printed.. If I look at it now, I think the name is
not printed.
1.4 Exercise LI2C001

What were your thoughts on this exercise?
I have chosen the way that you first do: int is get 0, so you get the value of the smallest till now. So in the next one you need to compare the values, which is the first if, because this if compares values with I guess, the value you have. And if it is smaller, then you would have to use the x.get so that the right value is received. And then you can just return the value.

2 Subject 2

2.1 Exercise VA2C001

Thank you for taking the time to answer the question. Here are some of the questions you just answered. I now would like to know your reasoning and your plan of action as you were confronted with this exercise.
This is a common swap function, as we have seen in a previous programming course. In which c as a helping variable is used. I have chosen to store a in it. So that you then can store b in a. And I can use C to store the old value of a in b.

2.2 Exercise CI2T002

What were your thoughts on this exercise?
Here, I spend some time to figure this out. But at the end my conclusion was, that the lowest value is printed. And how did you come to this conclusion? I don’t know, I just looked at the function and then I thought ”Wait it just prints the lowest value!”. Did you try this out with concrete values or did you only use the code to determine what it does?
I just looked at the code.

2.3 Exercise CW2C001

What were your thoughts on this exercise?
I substituted the while-loop for a for-loop. And the second if, through an else, because if the first is not even then it has to be odd. What is the purpose of the function?
It prints a name, and how much depends on which value in amount is. So
you saw what the function did, and then you tried to implement it yourself, or did you look at the implementation and then just did one or two things differently?

I just looked at the implementation and saw that a while loop can be best substituted with a for loop. So I did this.

2.4 Exercise LI2C001

What were your thoughts on this exercise?

I chose the first way of implementation. And then when I was at the second option, I was a little bit confused, but I guess at the end I did it correctly. Which option did you choose?

For the first one I chose, is zero. What is the purpose of this initialization?

This is the position of the smallest value and then you have to, for the second option, compare it to the x.get because you want to compare the value. And then you want to save the new position, so you have to choose for i and at the end you want to return the lowest value, so then you have to use the x.get again.

3 Subject 3

3.1 Exercise VA2C001

Thank you for taking the time to answer the question. Here are some of the questions you just answered. I now would like to know your reasoning and your plan of action as you were confronted with this exercise.

Here you store the value of a in c, change the value of a, give it value b. And then you choose this one and you get the previous value in b.

What is the purpose of the variable c? This is the temporary slot, to store the value.

3.2 Exercise CI2T002

What were your thoughts on this exercise?

I just walked through it and tried to understand what it does. It looks to me like it just looks at and prints the lowest value of these three. Thus a > b and b > c then c is the smallest and it prints this one. If c is not smaller
than b it prints b, because b was already smaller than a. And if that’s not true, it checks whether a is bigger than c and prints the smallest.

*Did you try this out with concrete values or did you only use the code to determine what it does?*

I did not try this one out. (Just looked at the code)

### 3.3 Exercise CW2C001

*What were your thoughts on this exercise?*

I thought a long time about it and I think, that you don’t need to know this in order to determine in which order hello is printed. You know that amount is even, so this is not needed (confuses variable i with variable amount). Its always gonna switch because it goes one up. So instead of having this code with amount times two, then you can just.. oh no that’s wrong, you don’t know if its equal or not. Well I got it wrong anyway. What I did is that first I did a for-loop, and then I said amount times two minus one and then I just printed name and hello. Cause it started with name. I think I changed it and took this away and then it doesn’t work.

*Did you understand exactly what the purpose of the code was?*

Yes, I guess if it is two for amount, we get four, we get "name", "hello", "name", "hello".

### 3.4 Exercise LI2C001

*What were your thoughts on this exercise?*

I just took this one started with minsofar zero. Then I checked this one, this one takes from the array, see if this is smaller then store it in minsofar. And choose this one, I think I did a mistake here, because I think I chose this one but it is not needed.

*So you store the value?*

Yes, so I store this value in minsofar.

*With which option did you start exactly?*

With the first option. (This option stores the position)
4 Subject 4

4.1 Exercise VA2C001

Thank you for taking the time to answer the question. Here are some of the questions you just answered. I now would like to know your reasoning and your plan of action as you were confronted with this exercise.

So you start with the storing of a or b, because you want to switch those. And then it depends on which one you stored first. So you can start with \( c = b \) but I chose for \( c = a \) so after that you can change a, so that it is b. Then you can say that \( b = c \), which holds the value of a.

4.2 Exercise CI2T002

What were your thoughts on this exercise?

You see that he checks if a is bigger than b and then if b is bigger then c and if a is not bigger than b and if a is not bigger than c, then he is going to print the lowest value. So I thought that the purpose of this piece of code was to print the lowest value of a, b or c.

4.3 Exercise CW2C001

What were your thoughts on this exercise?

Think of some things to substitute, for example a while loop is used, so here you can also use a for-loop. Or here the if-if can be substituted with a if-else. So that is what I did.

Did you understand exactly what the purpose of the code was?

No, I did not look at this.

4.4 Exercise LI2C001

What were your thoughts on this exercise?

You had to be consistent. So if you use minsofar here for the value itself then here at the following you shouldn’t use the implementation for the position. So you have to choose for \( x\text{.get} \), that you store the right value for the chosen implementation. And not for once store the position instead of the value for
the position. I chose for the first option, then the get and then the return minsofar.

5 Subject 5

5.1 Exercise VA2C001

Thank you for taking the time to answer the question. Here are some of the questions you just answered. I now would like to know your reasoning and your plan of action as you were confronted with this exercise. Logically I looked at what values they at the end should have. I know that it in this case c should have the value of a, so then a can have value of b. So I already switched one part. And if you give b the value of c, then you can give b the value of a, then you have switched them.

What is the purpose of the variable c?
You used c to store the value of a, so that you can store the value of b in a.

5.2 Exercise CI2T002

What were your thoughts on this exercise?
I looked at it stepwise on what it did exactly and what it would return. At the end I came to the conclusion, that it would give the smallest value of a, b and c back. Did you try this out with concrete values or did you only use the code to determine what it does?
I just looked at the code, no specific values. Then you could reason that the code would return the smallest value.

5.3 Exercise CW2C001

What were your thoughts on this exercise?
Here I looked at what the code would do exactly. I came to the conclusion, that it would print hello and then the name for the amount you chose. And here to use the modulo the amount is multiplied by 2. I wrote the same function but then with a for loop, personally I thought that would be more logical. And then I also saw that the second if was not needed and can be substituted with an else statement.
5.4 Exercise LI2C001

*What were your thoughts on this exercise?*
First I closely looked at the question to find out what was expected of me. Then I chose this option, with the `x.get`, because the code is about the smallest value in the list and I thought that you would set the `minsofar` to zero, so that you would get the smallest value of the list in the first iteration. I thought this would be the fastest way of implementation. (misinterprets the first statement)

6 Subject 6

6.1 Exercise VA2C001

*Thank you for taking the time to answer the question. Here are some of the questions you just answered. I now would like to know your reasoning and your plan of action as you were confronted with this exercise.*
First I thought about, how I would do it myself and then choose for the right code. I chose for `c = a`, `a = b` and `b = c` and the other three would have also been right.

6.2 Exercise CI2T002

*What were your thoughts on this exercise? For example, did you try this out with concrete values or did you only use the code to determine what it does?*
No, just looked at the code, and it prints the smallest values.

6.3 Exercise CW2C001

*What were your thoughts on this exercise?*
This is easy to do in a for-loop, because you have stop criteria. The only thing I was not sure about, here you have `print` and then here you have `println` so I was not sure if he would print everything on one line or if he would everything in subsequent lines. So I still reused both of those prints, so that the name is printed a certain amount.
6.4 Exercise LI2C001

What were your thoughts on this exercise?
In my opinion here there were also two correct answers. So it depends on how you define your minsofar, do you define it as the real minimum or do you define it at the position of the minimum. And because of this the choices for the rest of the code were given. I chose for the minsofar is the real value. So the second choice for the first options.

7 Subject 7

7.1 Exercise VA2C001

Thank you for taking the time to answer the question. Here are some of the questions you just answered. I now would like to know your reasoning and your plan of action as you were confronted with this exercise.
So in a previous programming course we were already told, that for a switch need an extra variable. So that you can store the value which you assign last, which is c in this case. And then you can swap.

7.2 Exercise CI2T002

What were your thoughts on this exercise?
That was simple, because you checked if b <a and then you check if b or c is smaller, so you know which one is smaller from the first comparison and then you check which one of those potentially smallest values is smaller. The one that is smallest you print. So at the end I figured out that you would print the smallest value. Did you try this out with concrete values or did you only use the code to determine what it does?
Did only look at the code.

7.3 Exercise CW2C001

What were your thoughts on this exercise?
First I looked at amount, and I saw amount is multiplied by two but amount is not incremented in the loop, so I thought to myself: "What is going on?". Then I saw even or non even number, then I also thought this is an odd way to do this. Then I read the question, to give a different implementation, so
then it was for me clear that you should use a for-loop. Then you don’t need to check whether i is even or odd, cause if you increment i then it is even, odd, even, odd... anyway.

7.4 Exercise LI2C001

*What were your thoughts on this exercise?*

Oh this was very easy, you had to implement it by value or by reference. So I did it by value, so to use minsofar to store the value, so you had to choose the first one (misinterprets first statement). And so forth.

8 Subject 8

8.1 Exercise VA2C001

*Thank you for taking the time to answer the question. Here are some of the questions you just answered. I now would like to know your reasoning and your plan of action as you were confronted with this exercise.*

First I saw you want to switch a and b and there you need a temporary variable which in this case is c. I first thought what does it matter if I chose for a or b in the first option, so I just chose for c is a. You can choose both of them but you need to be consistent. So c = a, you want to switch a and b, so then I did a = b and then b gets the value of a which was in c, so c = b.

8.2 Exercise CI2T002

*What were your thoughts on this exercise?*

This was about the global idea on what the code does. I looked at it, and most of the time something is checked, that one is bigger than the other. Then I went through the code, so I checked the statement a > b. If that’s true you check if b > c and then I thought that it would be something about the order and the size of the values of the variables. So I had the idea that it had something to do with the order, so to order those variables. My conclusion was, that the order of a, b and c was determined but as I look now at it, it could be something different. But I think the general idea has something to do with order.
8.3 Exercise CW2C001

What were your thoughts on this exercise?
Here I saw a while loop and then i+1 so I immediately thought, here you could better use a for-loop. And that is what I did.
Do you understand the purpose of this code?
Honestly, I did not look into that. In my opinion that was not relevant.

8.4 Exercise LI2C001

What were your thoughts on this exercise?
You had to be consistent, well I thought in one of those options you always use something with get and with the other ones you don’t. So you have the option if you will use get or not. So I always chose for those which used the get.
So you looked at the lines of code and chose the one were a get was included?
Yes, for example for those two this line has more gets than this one so I chose the one with more gets.