Mental Models of Recursion: A Secondary Analysis of Novice Learners' Steps and Errors in Java Exercises

Natalie Kiesler
DIPF Leibniz Institute for Research and Information in Education
Frankfurt, Germany
kiesler@dipf.de

Abstract

Teaching and learning recursive programming has been the subject of numerous research projects and studies. However, few research publications focus on learners' steps while solving recursive tasks and the corresponding identification of mental models. It is the goal of this secondary analysis to identify the challenges novice learners of Java encounter in recursive problem solving and to map them to the mental models and conceptions from the literature. The investigated dataset was collected via thinking aloud experiments with eleven first-year-students of computer science in a professional usability laboratory. Students had to recursively compute the factorial of n and the Fibonacci sequence in a learning environment. By using deductive categories from the literature, the students' performance was evaluated in terms of their programming steps, their challenges/errors, and thus their ability to generate a recursive function. The results show that mental models can partially be identified via the analysis of students' problem solving steps and errors. Moreover, recursive tasks with more than one recursive call are more challenging for novice learners. The passive flow of control along with the end of the recursion chain also seem to be counterintuitive for learners. The lack of viable, complete mental models implies the need for further educational research on instructional methods addressing these challenges of first-year students. Learning to program may be easy, but novices require fine-grained, step-by-step scaffolding and instruction, as well as time to understand and apply the more abstract concepts, which include recursion.

1. Introduction

Novice learners of programming experience a variety of challenges in the first year of their studies. Among them are the new circumstances due to the start of a new stage in life, organizational structures, high and unrealistic expectations during the study entry phase (Heublein et al., 2017; Große-Bölting, Schneider, & Mühling, 2019; M. McCracken et al., 2001; Luxton-Reilly, 2016). In the context of computing, students are further confronted with high performance requirements and possibly deficits. A great body of research is dedicated to the investigation of common obstacles for students in basic programming education (Spohrer & Soloway, 1986; Winslow, 1996; M. McCracken et al., 2001; Robins, Rountree, & Rountree, 2003; Luxton-Reilly et al., 2018). Among them is the lack of prior knowledge and expertise (Heublein, Richter, & Schmelzer, 2020), the cognitive gap between understanding and applying a concept (Ala-Mutka, 2004), a high level of abstraction (Kay & Wong, 2018), and the complexity of structuring and constructing algorithms, programs, and mental models (Kahney, 1983; Spohrer & Soloway, 1986; Winslow, 1996; Xinogalos, 2014). Programming primarily comprises cognitively complex tasks (Kiesler, 2020b, 2020a). Recursion is considered as one of them, which is due to its abstract nature, and the lack of everyday analogies; but also its introduction after loops, and the minor pedagogical emphasis towards it (Kurland & Pea, 1985; Leron, 1988; Levy & Lapidot, 2000; Pirolli & Anderson, 1985; Troy & Early, 1992; Wiedenbeck, 1988).

Despite this seemingly consensus on the difficulty of programming and the time and effort it takes to become an expert (Winslow, 1996), Luxton-Reilly summarizes that "learning to program is easy" (Luxton-Reilly, 2016); even children can do it. His arguments are compelling in the face of overwhelming, and repeated evidence of students' challenges in introductory programming (Tew, McCracken, & Guzdial, 2005; Whalley & Lister, 2009; Luxton-Reilly et al., 2018). Despite the great body of research, learners' challenges seem to continue. In the case of recursion, for example, learners still seem to experience more difficulties related to the passive flow of execution control, especially if two recursive calls are involved. Moreover, students can hardly predict when a recursive program terminates (Scholtz & Sanders, 2010).

In cognitive psychology, the concept of mental models refers to the individual, cognitive representations of a students' knowledge (Johnson-Laird, 1989; Norman, 2014; Schwamb, 1990), whereas the models can be characterized by their viability. In the context of computing, several models have been developed to categorize students' representation of knowledge on recursion (Kahney, 1983; Close & Dicheva, 1997; Bhuiyan, Greer, & McCalla, 1994). Helping students to develop adequate mental models of recursion is a crucial step towards improving their understanding and application of this cognitively complex concept. Educators therefore need a better understanding of students thinking, their problems and above all, their mental models, in order to help support them via conscious pedagogical decisions and instructions. However, educators need indicators to detect non viable mental models early.

In order to eventually help foster students' construction of mental models of recursion, this paper analyzes first-year Computer Science (CS) students' steps and challenges while recursively solving two programming tasks in Java. In addition, the aim is to investigate the extent of which students' problem solving steps can be aligned with the known mental models (Götschi, Sanders, & Galpin, 2003; Scholtz & Sanders, 2010) and reflect the associated viability. The research questions of the present work are: *(1) How do first-year CS students write recursive functions in Java? (2) Which challenges do first-year CS students encounter while recursively solving programming tasks in Java?* and *(3) To what extent are mental models from the literature reflected in students' problem-solving steps and errors?*

The contribution of this research is a secondary, qualitative analysis of novice programmers steps and the identification of their challenges related to recursive problem solving in Java. It will contribute to the greater generalization of students' representations of knowledge due to the analysis of their problem solving processes in an actual programming language. The results further provide implications on the recognition of non viable models, as well as the development of instructional methods and interventions to foster students' construction of viable mental models of recursion.

The structure of the paper is as follows. Section 2 summarizes related research on mental models, and mental models of recursion. In section 3, the methodology of this secondary analysis will be outlined by introducing the utilized dataset, and the method for data analysis. Next, the results with regard to students' steps (RQ1), and challenges (RQ2) in the recursive problem solving process are presented. These are then mapped with the mental models of recursion and discussed to answer RQ3. An overview of the limitations follows in section 5. Conclusions and future work wrap up this secondary research.

2. Related Research on Learners' Mental Models of Recursion

Mental models are a concept that is rooted in cognitive psychology (Johnson-Laird, 1989; Norman, 2014). They aim at the description of individual, cognitive representations of knowledge (Schwamb, 1990; Wu, Dale, & Bethel, 1998) through a constructivist lens. However, there is not a single constructivist theory. It is rather the sum of ideas on learning via the personal construction of meaning and corresponding instruction (Stone & Goodyear, 1995): "There are many ways to structure the world and there are many meanings or perspectives for any event or concept. Thus, there is not a correct meaning that we are striving for (Duffy & Jonassen, 1991)." There is thus not an ultimate reality everyone agrees upon (Duffy & Jonassen, 1991). Accordingly, learning is defined as the active, subjective process that highly depends on individual interpretation, experiences, and perspective (Stone & Goodyear, 1995). Therefore, knowledge is described as viable, and not as "true" or "correct". For this reason, this work does refer to the challenges novice learners experience in recursive problem solving, and not to the so-called "misconceptions" mentioned in other related work (Close & Dicheva, 1997; Dicheva & Close, 1993). The more radical constructivist theories even neglect the idea of any objective reality (Von Glasersfeld, 1996).

Mental models have also been investigated in the context of computing, and to describe student's individually constructed knowledge of recursion (Bhuiyan et al., 1994; Kahney, 1983; Close & Dicheva, 1997; Dicheva & Close, 1993, 1996; Scholtz & Sanders, 2010). It was found that novice learners' models deviate from the expert model of recursion, making them unable to predict how recursive programs behave (Kahney, 1983; Wu et al., 1998). While novices tend to have constructed the incorrect loop model, experts share the copies model (at least in SOLO, LISP and Logo) (Close & Dicheva, 1997). With more experience and practice, however, novices can change their construction and representation of knowledge, which is at the heart of constructivist learning (Duffy & Jonassen, 1991; Von Glasersfeld, 1996). Norman (2014) concludes that mental models may not be stable, i.e., they can be forgotten or confused with other systems to reduce mental complexity.

Kahney (1983) defines recursion as "a process that is capable of triggering new instantiations of itself, with control passing forward to successive instantiations and back from terminated ones (Kahney, 1983)." Similarly, George (2000a) defines recursion via the "(active) 'flow of control' to a new invocation/copy of the subprogram called", and the passive flow of control "back from terminated ones" (George, 2000a). According to Kahney (1983), and Kessler & Anderson (1986), novices do experience challenges in understanding tail recursion with functions and embedded recursion, which is related to their lack of understanding the control mechanisms, especially the passive flow of control. This in turn implies them having unfavorable mental models of recursion. Scholtz & Sanders (2010) conclude that students often develop inadequate mental models when two recursive calls are part of a procedure or function. They also observed that students can hardly predict when a recursive algorithm terminates, as they tend to associate recursion with the (previously constructed) loop model. The passive flow of control back from the terminated instantiations is thus a common problem among novice learners of programming. Velázquez-Iturbide (2000) adds that recursion seems to be more difficult in imperative programming languages, as educators tend to confuse recursion and imperative recursion, where other mechanisms (e.g., parameter passing, control stack, etc.) have to be mastered simultaneously.

The following mental models of recursion are summarized in related research investigating and classifying students' constructs (Kahney, 1983; Götschi et al., 2003; Sanders, Galpin, & Götschi, 2006):

- The *Copies Model* is the viable model that reveals the active flow of control, and the switch to the passive flow once the base case is reached. The passive flow of control is made explicit.
- The *Loop Model* views recursion as a kind of iteration that halts once the base case is reached. It ignores both the active and passive flow of control. The base case is considered as stopping condition of the loop.
- The *Active Model* only reflects the active flow of control, but the indication of a passive flow is absent. Students evaluate the solution at the base case. The model can be viable in some cases.
- The *Step Model* is nonviable, as the students lacks understanding of recursion. Either the recursive condition, or the recursive condition and the base case is executed once.
- The *Return Value Model* describes the view that values are generated by each instantiation, which are then stored and combined to calculate a solution.
- The *"Syntactic", "Magic" Model* reveals that students have no idea of recursion and how it works. Nonetheless, they can match syntactic elements. The active flow, base case, and passive flow can be traced. Due to their errors, a lack of understanding is assumed, which requires further teaching/learning activities.
- The *Algebraic Model* describes students who treat the program as algebraic problem.
- The *Odd Model* encompasses different misunderstandings, which lead to the student not being able to predict the program's behavior.

In addition to developing viable mental models, learners need time and practice to construct mental models to understand and apply program structures, whereas even varying mental models can lead to identical and correct problem solutions (Kahney, 1983). To conclude, it is difficult for learners to develop adequate schemata. Likewise, educators face challenges in assessing individual mental models and constructs required for the successful planning of algorithms and programs. Due to these reasons, it is important to investigate mental models of novice learners of programming further by examining their ability to recursively solve problems in a commonly used programming language, such as Java. The results and implications on students' mental models will help address learners' challenges by developing new forms of instructions and pedagogical concepts.

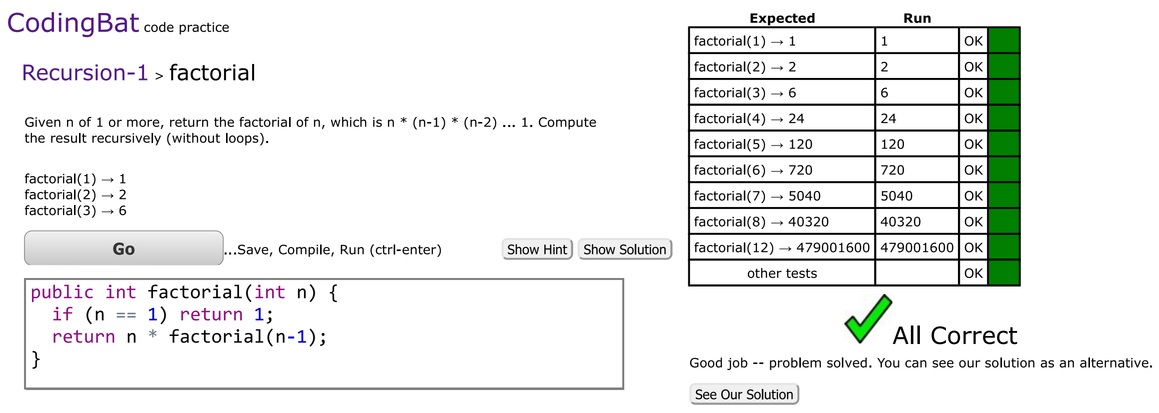
3. Methodology

For this secondary analysis of research data, a publicly available, qualitative dataset with students' problem solving steps was utilized to identify their challenges, and to what extend their mental models of recursion are reflected. This sections briefly introduces the context of the primary dataset, its structure, and tasks. Moreover, the analysis of the data is presented. The goal is to investigate students' steps and errors, and to test the mental models further, as implied by prior research (Close & Dicheva, 1997).

3.1. Data Collection

In educational technology research, identifying research data for secondary research is challenging due to several reasons (e.g., lack of recognition for researchers, lack of quality, data provenance etc. (Kiesler & Schiffner, 2022)). In the context of programming education, few datasets are available that allow for the analysis of novice learners' steps while recursively solving programming exercises in Java. One of them is provided by a German research data center (Kiesler, 2022a, 2022b). It was part of a recent doctoral dissertation in the context of introductory programming education (Kiesler, 2022d).

The data had originally been gathered to investigate the effects of informative feedback offered by (online) self-learning tools (Kiesler, 2022a), such as CodingBat (Parlante, 2022a) (see Figure 1). Two recursive tasks in Java were selected for the conduction of two thinking aloud experiments (test series A and B) (Heine, 2005; Knorr, 2013; Konrad, 2017) where students had to recursively solve problems in Java. In sum, eleven students of the Department of Applied Computer Science participated as test subjects in series A and B. All of them had successfully participated and completed the course "Programming 1" (i.e., the basic programming course) at Fulda University of Applied Sciences in Germany, which also addresses the concept of recursion as part of both the lecture and exercise session.



Given n of 1 or more, return the factorial of n, which is n * (n-1) * (n-2) ... 1. Compute the result recursively (without loops).

factorial(1) → 1
factorial(2) → 2
factorial(3) → 6

```
public int factorial(int n) {
  if (n == 1) return 1;
  return n * factorial(n-1);
}
```

Expected	Run		
factorial(1) → 1	1	OK	🟩
factorial(2) → 2	2	OK	🟩
factorial(3) → 6	6	OK	🟩
factorial(4) → 24	24	OK	🟩
factorial(5) → 120	120	OK	🟩
factorial(6) → 720	720	OK	🟩
factorial(7) → 5040	5040	OK	🟩
factorial(8) → 40320	40320	OK	🟩
factorial(12) → 479001600	479001600	OK	🟩
other tests		OK	🟩

Good job -- problem solved. You can see our solution as an alternative.

Figure 1 – Condensed screenshot of a CodingBat exercise on recursion.

Students had to solve two tasks, which served as test instruments. Both test series A and B included the computation of the factorial of n, resulting in a sample size of 11 students for the first task (students A01 to A06, and B01 to B05). In addition, the Fibonacci sequence was computed by the five students of test series B (students B01 to B05). A total of 16 transcripts from 11 students are thus available. In the primary research, another tool/task (Kiesler, 2016a, 2016b) was tested in series A, resulting in a lower N in the Fibonacci task. Computing the factorial of n as a task in both series A and B requires students to write a base case and one recursive call. Computing the Fibonacci sequence as second task in series B demands two base cases, and two recursive calls. Both tasks require students to apply *tail recursion*, which refers to the position of the recursive call as last action of the algorithms.

1. N=11: Given n of 1 or more, return the factorial of n, which is n * (n-1) * (n-2) ... 1. Compute the result recursively (without loops) (Parlante, 2022b).

2. N=5: The fibonacci sequence is a famous bit of mathematics, and it happens to have a recursive definition. The first two values in the sequence are 0 and 1 (essentially 2 base cases). Each subsequent value is the sum of the previous two values, so the whole sequence is: 0, 1, 1, 2, 3, 5, 8, 13, 21 and so on. Define a recursive fibonacci(n) method that returns the nth fibonacci number, with n=0 representing the start of the sequence (Parlante, 2022c).

The dataset comprises the manually transcribed students steps in the problem solving process in form of a table, so that students' input to the available CodingBat working space can be traced. Every change in the working space (where the programming code is written) has been transcribed, as depicted in Table 1. Students' steps were further qualitatively analyzed into categories implying the type of change (e.g., keyword added (if), recursive function call added, parameter of recursive function call added, etc.). The dataset also reveals timestamps, and the system feedback provided by CodingBat (Parlante, 2022a), so that every change/step made by students can be reconstructed and students' reactions to the system feedback can be tracked. A detailed description of the dataset and the applied methodology is available online (Kiesler, 2022b).

Table 1 – Excerpt Representing the Structure of the Dataset on Students' Steps.

time	programming code	(qualitative) code	action (button)	system feedback
25:05	`public int factorial(int n) {` `}`	problem solving starts		
25:14	`public int factorial(int n) {` `}`	no change	Go button	Compile problems: missing return statement line:3 see Example Code to help with compile problems
25:17	`public int factorial(int n) {` `}`	no change	Hint button	First, detect the "base case", a case so simple that the answer can be returned immediately (here when n==1). Otherwise make a recursive call of factorial(n-1) (towards the base case). Assume the recursive call returns a correct value, and fix that value up to make our result.
25:49	`public int factorial(int n) {` `if ()` `}`	keyword added (if)		
26:06	`public int factorial(int n) {` `if (n == 1)` `}`	one condition added (base case)		
26:09	`public int factorial(int n) {` `if (n == 1) return` `}`	keyword added (return)		

3.2. Data Analysis

Due to this research's secondary nature, the primary dataset and its structure (see Table 1) predetermine the secondary analysis as a whole. In the case of the present dataset, students' sequence of problem solving steps/changes in the programming code can be analyzed to answer the research question *(1) How do first-year CS students write recursive functions in Java?* In order to answer RQ1, the lines of code students write are analyzed with regard to their steps, for example, whether they relate to the base case, the recursive function call, or the parameter of the recursive function call. In this context, the changes of the programming code will also be relevant. The number of changes are defined via the steps needed to come to the correct solution after having made a mistake that was executed via the Go-Button provided by CodingBat (e.g., deleting, adding, changing return values, parameters, etc.). Changes in indentations are ignored. A step aligns with the primary research's analysis, where every step (displayed per row in Table 1) was assigned a qualitative code. The qualitative codes of all 16 records will be analyzed and clustered with regard to these crucial elements of the recursive solution to the tasks. Timestamps, however, will not be considered.

Next, students' errors will be analyzed to answer the second research question *(2) Which challenges do first-year CS students encounter while recursively solving programming tasks in Java?* In this regard, the main components of recursion will be used as categories: the active flow, the passive flow, and the base case(s). If applicable, these categories are supplemented by further subcategories, which are inductively built and used to describe accompanying challenges, or error causes. In addition, commonly known mistakes from the literature (Close & Dicheva, 1997) are considered for the qualitative coding.

As a last step of this analysis, students' steps and challenges during recursive problem solving will be mapped to the known mental models described in Section 2 to answer research question *(3) To what extent are mental models from the literature reflected in students' problem-solving steps and errors?* As a part of this mapping, the definition of models will be used to identify patterns in the problem solving process that may be aligned with the construction of the mental models' categories as defined by Kahney (1983); Götschi et al. (2003); Sanders, Galpin, & Götschi (2006). The corresponding identification of indicators will help operationalize mental models for educators, and eventually help foster learners' understanding of recursion.

4. Results

4.1. How Students Write Recursive Functions in Java (RQ1)

The problem solving process to some extent manifests in how students write a program as a solution to a task. In the present dataset, all 16 solutions were analyzed with regard to the main components of the expected recursive functions and their sequence, e.g., the if statement related to the base case(s), the recursive function call(s) and its/their parameter(s), as well as the correct use of operators. The two tasks were analyzed separately, and the following analysis distinguishes between the two tasks.

4.1.1. Factorial of n

First of all, it should be noted that all 11 students who worked on this task were able to solve it. Among the four students who successfully solved the task without a single error (A02, A03, A04, B03), three different sequences were observed. In all of them, first-year students write the if statement representing the base case first. The sequence of the remaining components (recursive function call, parameter of recursive function call, and multiplication with n) deviates. However, writing the recursive function call in all cases precedes writing the parameter of recursive function call, which seems to be a successful sequence.

In the remaining seven problem solving processes, similar pattern were recognized. Six of the seven remaining students also started writing the if statement representing the base case first. Only one student (B04) who merely forgot a semicolon and was otherwise error-free wrote the if statement last and began with the recursive function call. Yet again, all students write the recursive function before adding its parameter. The multiplication with n is a step that seems to deviate in its position. Six of the eleven students (A01, A03, A04, A06, B02, B05) immediately write it after finishing the base case. Two of the eleven write it after the recursive function call (A02, B04), and the remaining three write it as a last step (A05, B01, B03).

Students' changes to their code are analyzed next. Due to the four error-free samples, seven samples with changes remain. Table 2 summarizes the changes students made to their code. It should be noted that the table does not represent the sequence of their steps. Most of the changes were related to the if statement and thus the base case. Four student had to adapt their code. Similarly, four students (A01, A05, B01, B02) edited the parameter of their recursive function call. The recursive function call, however, required

Table 2 – Number of changes made by students during recursively solving the factorial of n task.

Student Steps:	**A01**	**A05**	**A06**	**B01**	**B02**	**B04**	**B05**
if statement (base case)	-	5	-	4	5	-	1
multiplication with n	-	-	-	4	4	-	-
recursive function call	-	-	-	2	-	-	-
parameter of recursive function call	1	3	-	1	5	-	-

only one student (B02) to make changes to their initial code, which means that 10 of the 11 students correctly wrote that component. The multiplication with n required two students (B01, B02) to adapt their code.

4.1.2. Fibonacci Sequence

The Fibonacci task was successfully solved by two students (B01, B04), while two other students (B02, B03) aborted the problem solving process altogether without solving the problem. With the small sample size for this task (n=5), few data remains. Nonetheless, the sequence of students' steps is briefly summarized. The two students with no errors and changes applied the following sequence of steps:

1. if statement (first base case)
2. if statement (second base case)
3. first recursive function call
4. parameter of first recursive function call
5. add operator
6. second recursive function call
7. parameter of first second recursive function call

Although student B05 tried to use addition instead of two recursive calls first, the learner in general applied the same sequence of steps. Students B02 and B03 who did not develop a viable solution, made changes related to the second recursive function call, forgot it or misplaced it along with its parameter as part of the second base case (see, for example, Listing 1).

```
public int fibonacci(int n) {
    if(n==0) return 0;
    if(n > 1) n + fibonacci(n-1);
    return n + fibonacci(n-1);
}
```

Listing 1 – Example of student B02's positioning of the second recursive function call.

Students' changes of their code are summarized in Table 3. Again, two students (B02, B05) were concerned with the correction of their if statements, which represent the base case(s). Moreover, the first recursive function caused errors and therefore required changes from two students (B02, B05). Student

Table 3 – Number of changes made by students during recursively solving the Fibonacci task (n.a. = not available).

Student Steps:	**B02**	**B03**	**B05**
if statement (first base case)	-	-	1
if statement (second base case)	4	-	-
first recursive function call	1	-	2
parameter of first recursive function call	1	2	2
add operator	-	-	-
second recursive function call	-	n.a.	-
parameter of second recursive function call	-	17	-

B02, for example, somehow added the first recursive function to the base case (see Listing 1). B03 was especially busy with the edition of the parameter of the second recursive function call, as the second recursive function call itself was not written, and the error could not be corrected after 17 attempts. The parameter of the (first and second) recursive function call required changes from all three students B02, B03 and B05. It should be noted that CodingBat did not provide a hint or model solution for the task. Only the expected values were presented and compared with students' results (see right-hand side of Figure 1).

4.2. Students' Challenges During Recursive Problem Solving (RQ2)

In this section, the first-year CS students' challenges during recursive problem solving are summarized. For each tasks, a categorisation of student errors is presented and related to the main components of the expected recursive solution.

4.2.1. Factorial of *n*

Although the recursive computation of the factorial of n may seem simple, and all 11 students were capable of solving the problem, errors were made during the process. The left column of Table 4 summarizes the observations into categories of detected student errors. The number in brackets behind the bullet points indicates the frequency of the errors, and thus how many of the students made these errors.

The list reveals that not all errors were related to recursion. The first bullet point reveals a number of syntax and semantics problems resulting in compile errors or the CodingBat feedback "Bad Code". The second bullet point reveals that a student did not adhere to the starter code provided by CodingBat, which is always correct. The student thus tried to alter the function's signature and solve the problem by introducing an additional parameter and variable. The last three bullet points concern students' errors related to recursion, e.g., the condition or return values of the base case, the recursive function call, and its parameter. They thus comprise all three key components of the recursive solution to the problem. It should further be noted that 4 of the 11 students were able to solve the task without any error.

4.2.2. Fibonacci Sequence

The second task required students to write two base cases and use two recursive function calls with correct parameters. Two of the five students with available data were able to successfully solve the task without any error. Two students did not develop a viable solution and aborted the problem solving process. The right column of Table 4 summarizes the categories of detected errors.

The enumeration contains only logic errors, although the first bullet point is not directly related to recursion. Instead, the student tried to solve the problem with an additional help function. The three other error categories, however, are closely related to the main components of recursion. Similarly to the first task, learners experienced challenges when writing the base case, the first and second recursive function call, and the corresponding parameters. In addition, the selection of an operator for the calculation of the return value of the else-block seemed to be challenging. The lack of the second recursive function call (with parameter) and students' substitution of it via some other operation should be noted as main cause of errors.

4.3. What Students' Steps and Errors Reveal about their Mental Models of Recursion (RQ3)

The last research question aims at the identification of indicators of students' mental models of recursion. Unlike related work, this research did not analyze students tracing of the execution of recursive programs (Kahney, 1983; Götschi et al., 2003; Sanders et al., 2006). Thus, the implications of students' steps and errors on their models will be discussed based on the qualitative dataset. Nonetheless, the categories Götschi et al. (2003) used for the analysis of students' traces related to the active flow, the base case, and the passive flow were considered as starting point. However, due to their (Götschi et al., 2003) lack of a detailed definition and anchoring examples for the categories, the coding scheme could not entirely be replicated.

Table 4 – Overview of Students' Challenges While Recursively Solving Two Examplary Problems.

Factorial of *n* (*n*!)	**Fibonacci Sequence**
• Syntax and semantics errors – using incorrect operators as condition of the case case or parameter (2) – lack of return statement in else block (2) – lack of return value of else block – adding redundant return statement – mixing up `System.out.println` and return statements – lack of semicolon • Falsification of starter code – changing the functions' given correct signature by adding a parameter – follow-up error: introducing additional, unnecessary variable • Logic Error: Errors related to the base case – incorrect return value (zero) of base case (2) – incorrect condition of base case • Logic Error: Errors related to recursive function call – lack of recursive function call (with parameter) – adding a second, unnecessary recursive function call (plus parameter) • Logic Error: Errors related to parameter of recursive function call – multiplication of n is within the parameter of the recursive function call (2) – other incorrect parameter of recursive function call	• Logic Error: Declaration of additional function – addition of function signature with additional parameter • Logic Error: Errors related to base cases – incorrect return value of second base case – lack of second base case – incorrect condition of base case • Logic Error: Errors related to recursive function calls – addition with n (or other operation, e.g., $n-1$, $2*n$, $n-(2*(n-1))$, $n-3$, $n-2$, $n-(n/2)$, $n-(n-2)$, etc.) instead of second recursive function call (with parameter) as return values of the else-block (3) – lack of second recursive function call (with parameter) in else-block (2) – placement of second recursive function call (with parameter) as return value of the second base case • Logic Error: Errors related to parameters of recursive function calls – incorrect parameter(s) of first recursive function call

4.3.1. Active Flow

Some of the categories by (Götschi et al., 2003) could be applied to students' steps, changes and errors without requiring more data or details. Among them were code alterations related to the active flow of control, thus calling factorial(), until its argument becomes 1. In this context, the algebraic manipulation of the function call (+n,+n-1, etc.) instead of a second recursive call was observed (B03, B05). Thus the need for a second recursive function call was not clear to all first-year students. The "algebraic" category (Götschi et al., 2003) is an indicator of students having adopted the *step model* or *return value model*. Determining the parameter of the recursive function call was challenging for many students. In 7 of the 16 records and in both tasks, students changed the parameters after initial, unsuccessful attempts to execute their code. Nonetheless, all students wrote at least one recursive call with a parameter, and thus made "a new invocation with a new argument" (Götschi et al., 2003). Moreover, several students achieved a correct solution without requiring feedback from the system. It can be assumed that they share at least this component of the *copies model* of recursion.

4.3.2. Base Case

Two categories (Götschi et al., 2003) assigned to the base case were observed. The first one is the "check incorrect" category describing an incorrect test for a base case. This error occurred in both tasks (e.g., B01, B02; factorial of n, and B02; Fibonacci). The incorrect test definition may indicate the *odd model*, as students are not able to predict the program's behavior. The second category "base omitted" was detected when subject B03 tried to compute the Fibonacci sequence. The second base case was omitted.

Other errors related to the base case (see Table 4) occurred in both tasks, and concerned incorrect return values. This error is evaluated as yet another indicator for the *return value model* or the *step model*. Yet again, the six cases of immediate correct student solutions (out of 16) may imply that students developed the viable *copies model*, where the switch from active flow to passive flow takes place once the base case is reached. In any case, all students were aware of the necessity of a base case.

4.3.3. Passive Flow

Students' steps, their changes to the code and errors somewhat hint towards students' mental models of recursion. Although judgements about students' conception of the passive flow of control are challenging via the analysis of their program code, the "return problem" and "changed operations" implied non viable models. For example, student B02 added a recursive function call to the second base case in the Fibonacci task (see Listing 1), implying misconceptions about parameter passing and return value evaluation. The error could imply the *odd model*. The change of operations (addition, subtraction, multiplication, division), and their order was also observed when students worked on the parameter of the recursive function call. According to (Götschi et al., 2003) and (Kahney, 1983) this is an indicator of the *magic model*, because students seem to be "sensitive to the position of the different program segment" (Kahney, 1983).

4.3.4. Discussion and Implications for Teaching

The idea to analyze students' steps, and errors to gain insight into their mental models of recursion revealed a number of challenges and findings. These will be discussed in the following, along with the implications for introductory programming education. First of all, the categories and models developed in related work (Kahney, 1983; Götschi et al., 2003; Sanders et al., 2006) proved to be incomplete with regard to their categorization, category definitions and their application to student solutions. In addition, an evaluation of categories for all three components (active flow, base case, and passive flow) along with a mapping to the resulting model is not available. A full replication is thus impossible. However, some indicators for the models presented in prior work were identified. Among them were categories related to the active flow (algebraic), base case (check incorrect, base omitted) and passive flow (return problem, operation changed). This leads to the assumption that students' steps and errors do provide indicators on their mental models of recursion, especially, if their problem solving steps reveal errors related to the recursive function call(s), their parameter, the condition and return value of the base case, and the if statement in general. Therefore, the assessment of students' steps and errors seems to constitute a promising approach for educators to help facilitate student learning and the development of viable mental models. However, whether a student developed a viable mental model could not be confirmed, despite some students achieving the correct solution in either the factorial, or the Fibonacci task. As none of the students from the B test series succeeded in both tasks, the conclusion is that students did not develop fully viable models yet.

A second important finding concerns students' challenges and errors. For example, the numerous adaptions of students' code indicate that student do not know how to write a viable condition (if statement) with a return statement, even when receiving several types of tutoring feedback by a tool like CodingBat (Kiesler, 2022c). Recursion does not seem to be the only obstacle. Students are not convinced of their selection of parameters, they change them often, and seemingly random. This may indicate that their knowledge representation of parameter passing is not yet complete/viable. In case of the Fibonacci sequence, the many changes related to the second recursive function call, its parameter and the calculation of the return value seems to be most challenging for learners. They also need to recognize that two base cases with a certain return value are required. While analyzing students' application of the concept of recursion, we need to ask whether we as educators can evaluate mental models of recursion in isolation, without the review of related concepts, learning objectives, and competencies (Raj et al., 2022).

Third, the findings reveal that many of the first-year student solutions show indicators of non viable mental models of recursion. According to Luxton-Reilly (Luxton-Reilly, 2016), learners do understand the concept of recursion, but it may take them longer than educators (the experts) expect. George (2000b) suggests that the explicit simulation of a recursive algorithm's execution via diagrammatic traces may be the best method when teaching recursion to students. Wilcocks und Sanders (1994) suggest animations to illustrate the execution of recursive programs, the flows of control and the resulting "copies" (Kahney, 1983; Sanders et al., 2006). Scholtz and Sanders (2010) recommend to use numerous examples of recursive problems and algorithms. Close further outlines a number of approaches for teaching recursion to children, aged 10 to 14 (Close & Dicheva, 1997). It has further been argued that recursion should not be treated as an advanced topic, but rather be taught early in the curriculum (Astrachan, 1994; D. D. McCracken, 1987): "If recursion is presented as one powerful tool among others, through many examples and with opportunities to practice using it" (D. D. McCracken, 1987). Similarly, a gradual approach to recursion is recommended by Velazquez-Iturbide (2000). The present findings support these perspectives and implications for teaching recursion to novices.

5. Limitations

The limitations of this work are due to the small sample size and its origin from a single institution. Moreover, the limitations of the primary research apply with regard to the unnatural situation in the usability laboratory where the experimental setup had been realized. Students may have behaved differently in a natural setting. The transcription and qualitative analysis of the dataset, however, allows for the precise reproduction of students' steps in that setting, and an in-depth perspective into students' recursive problem solving in Java. More qualitative, pre-processed data on students' steps would nonetheless help generalize the understanding of how students solve recursive programming tasks in Java.

6. Conclusions and Future Work

The goal of this research was to identify the steps and challenges of novice learners in the context of recursive problem solving, and to explore what students' steps and errors reveal about their mental models of recursion. The present work utilized a publicly available, qualitative, and pre-coded dataset with students' problem solving steps during two Java exercises which had been gathered in a usability laboratory. The data analysis clustered students' sequences of steps, their changes to the code, as well as their errors in alignment with the main components of the expected recursive solutions. Although prior research (Götschi et al., 2003; Sanders et al., 2006) could not be replicated, indicators for students' mental models were identified. Among them are students' errors related to the base case, the recursive function calls, and their parameters. The second recursive call in the Fibonacci sequence seemed another challenge. Moreover, syntax and semantics errors (e.g., operators, return statements, declaration of functions, parameter passing, etc.) posed challenges to learners. The analysis whether, and to what extent non viable mental models, such as the *step model* or *return value model* and their indicators are reflected and observable in students' problem solving steps and errors will help educators identify non viable models, and foster students' understanding of recursion in terms of their mental representation of the concept. Learning environments may also adapt their feedback accordingly.

To conclude, recursion is still a challenging concept for novice learners, which is why the need for adequate, learner-centered instruction is a continuing one. The paper further results in implications for introductory programming education, such as more critical teaching practices, revisiting expected norms for introductory programming and becoming more realistic towards achievable learning outcomes of introductory courses. In future work, big data or learning analytics approaches may be used to analyze, or pre-process quantitative datasets on students' steps, which are available at repositories (Koedinger et al., 2010). Such research will help educators find more and early evidence of non viable mental models in novice learners' programs. Another currently pursued approach is the development of expert feedback for the two selected exercises and students' steps (see, e.g. Jeuring et al., 2022).

7. References

Ala-Mutka, K. (2004). *Problems in learning and teaching programming – A literature study for developing visualizations in the Codewitz-Minerva project.* Online. Retrieved from https://www.cs.tut.fi/~edge/literature_study.pdf

Astrachan, O. (1994). Self-reference is an illustrative essential. In *Proceedings of the twenty-fifth sigcse symposium on computer science education* (pp. 238–242).

Bhuiyan, S., Greer, J. E., & McCalla, G. I. (1994). Supporting the learning of recursive problem solving. *Interactive Learning Environments*, *4*(2), 115–139.

Close, J., & Dicheva, D. (1997). Misconceptions in recursion: diagnostic teaching. In *Proceedngs of the sixth eurologo conference "learning and exploring with logo* (pp. 132–140).

Dicheva, D., & Close, J. (1996). Mental models of recursion. *Journal of Educational Computing Research*, *14*(1), 1–23.

Dicheva, D., & Close, S. (1993). Misconceptions and Mental Models of Recursion. In *Proceedings of the fourth european logo conference* (pp. 12–20).

Duffy, T. M., & Jonassen, D. H. (1991). Constructivism: New implications for instructional technology? *Educational technology*, *31*(5), 7–12.

George, C. E. (2000a). Erosi—visualising recursion and discovering new errors. *ACM SIGCSE Bulletin*, *32*(1), 305–309.

George, C. E. (2000b). Experiences with novices: The importance of graphical representations in supporting mental models. In *Ppig* (p. 3).

Götschi, T., Sanders, I., & Galpin, V. (2003, jan). Mental models of recursion. *SIGCSE Bull.*, *35*(1), 346–350. doi: 10.1145/792548.612004

Große-Bölting, G., Schneider, Y., & Mühling, A. (2019). It's like computers speak a different language: Beginning students' conceptions of computer science. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research.* New York: ACM. doi: 10.1145/3364510.3364527

Heine, L. (2005). Lautes Denken als Forschungsinstrument in der Fremdsprachenforschung. *Zeitschrift für Fremdsprachenforschung: ZFF*, *16*(2), 163–186.

Heublein, U., Ebert, J., Hutzsch, C., Isleib, S., König, R., Richter, J., & Woisch, A. (2017). Zwischen Studienerwartungen und Studienwirklichkeit. In *Forum hochschule* (Vol. 1, pp. 134–136).

Heublein, U., Richter, J., & Schmelzer, R. (2020). Die Entwicklung der Studienabbruchquoten in Deutschland. *DZHW Brief*(3). Retrieved from `https://doi.org/10.34878/2020.03.dzhw_brief`

Jeuring, J., Keuning, H., Marwan, S., Bouvier, D., Izu, C., Kiesler, N., ... Sarsa, S. (2022). Steps learners take when solving programming tasks, and how learning environments (should) respond to them. In *Proceedings of the 27th acm conference on on innovation and technology in computer science education vol. 2* (p. 570–571). New York: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/3502717.3532168`

Johnson-Laird, P. N. (1989). Mental models. *Foundations of cognitive science*, 469-499.

Kahney, H. (1983). What Do Novice Programmers Know about Recursion? In *Proceedings of the sigchi conference on human factors in computing systems* (p. 235–239). New York: Association for Computing Machinery. doi: 10.1145/800045.801618

Kay, A., & Wong, S. H. S. (2018). Discovering Missing Stages in the Teaching of Algorithm Analysis: An APOS-Based Study. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research.* New York: Association for Computing Machinery. doi: 10.1145/3279720.3279738

Kessler, C. M., & Anderson, J. R. (1986). Learning flow of control: Recursive and iterative procedures. *Human-Computer Interaction*, *2*(2), 135–166.

Kiesler, N. (2016a). Ein Bild sagt mehr als tausend Worte–interaktive Visualisierungen in webbasierten Programmieraufgaben. In U. Lucke, A. Schwill, & R. Zender (Eds.), *DeLFI 2016–Die 14. E-Learning Fachtagung Informatik, 11.–14. September 2016, Potsdam* (Vol. P-262, pp. 335–337). GI. Retrieved from `https://dl.gi.de/20.500.12116/566`

Kiesler, N. (2016b, 4-6 July, 2016). Teaching Programming 201 with Visual Code Blocks instead of VI, Eclipse or Visual Studio–Experiences and Potential Use Cases for Higher Education. In *EDULEARN16 Proceedings* (p. 3171-3179). IATED. doi: 10.21125/edulearn.2016.0169

Kiesler, N. (2020a). On Programming Competence and Its Classification. In *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research.* New York: Association for Computing Machinery. doi: 10.1145/3428029.3428030

Kiesler, N. (2020b). Towards a Competence Model for the Novice Programmer Using Bloom's Revised Taxonomy – An Empirical Approach. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 459–465). New York: ACM. doi: 10.1145/3341525.3387419

Kiesler, N. (2022a). *Dataset: Recursive problem solving in the online learning environment CodingBat by computer science students.* Online. (Datenerhebung: 2017. Version: 1.0.0. Datenpaketzugangsweg: Download-SUF. Hannover: FDZ-DZHW. Datenkuratierung: İkiz-Akıncı, Dilek) doi: https://doi.org/10.21249/DZHW:studentsteps:1.0.0

Kiesler, N. (2022b). *Daten- und Methodenbericht Rekursive Problemlösung in der Online Lernumgebung CodingBat durch Informatik-Studierende* (Tech. Rep.). (`https://metadata.fdz.dzhw.eu/public/files/data-packages/stu-studentsteps$/attachments/studentsteps_Data_Methods_Report_de.pdf`)

Kiesler, N. (2022c, June). *An Exploratory Analysis of Feedback Types Used in Online Coding Exercises.* arXiv. Retrieved from `https://doi.org/10.48550/arXiv.2206.03077` doi: 10.48550/ARXIV.2206.03077

Kiesler, N. (2022d). *Kompetenzförderung in der Programmierausbildung durch Modellierung von Kompetenzen und informativem Feedback* (Dissertation). Johann Wolfgang Goethe-Universität, Frankfurt am Main. (Fachbereich Informatik und Mathematik)

Kiesler, N., & Schiffner, D. (2022). On the lack of recognition of software artifacts and it infrastructure in educational technology research. In P. A. Henning, M. Striewe, & M. Wölfel (Eds.), *20. Fachtagung Bildungstechnologien (DELFI)* (pp. 201–206). Bonn: Gesellschaft für Informatik e.V. doi: 10.18420/delfi2022-034

Knorr, P. (2013). Zur Differenzierung retrospektiver verbaler Daten: Protokolle Lauten Erinnerns erheben, verstehen und analysieren. In K. Aguado, L. Heine, & K. Schramm (Eds.), *Introspektive Verfahren und qualitative Inhaltsanalyse in der Fremdsprachenforschung* (pp. 31–53). Frankfurt: Peter Lang.

Koedinger, K., Baker, R., Cunningham, K., Skogsholm, A., Leber, B., & Stamper, J. (2010). A Data Repository for the EDM community: The PSLC DataShop. In C. Romero, S. Ventura, M. Pechenizkiy, & R. Baker (Eds.), *Handbook of educational data mining.* CRC Press: Boca Raton, FL.

Konrad, K. (2017). Lautes Denken in psychologischer Forschung und Praxis. In G. Mey & K. Mruck (Eds.), *Handbuch qualitative Forschung in der Psychologie* (pp. 2–21). Wiesbaden: Springer Fachmedien.

Kurland, D. M., & Pea, R. D. (1985). Children's mental models of recursive LOGO programs. *Journal of Educational Computing Research*, *1*(2), 235–243.

Leron, U. (1988). What makes recursion hard. In *Proceedings of the sixth international congress on mathematics education.*

Levy, D., & Lapidot, T. (2000). Recursively speaking: analyzing students' discourse of recursive phenomena. In *Proceedings of the thirty-first sigcse technical symposium on computer science education* (pp. 315–319).

Luxton-Reilly, A. (2016). Learning to program is easy. In *Proceedings of the 2016 acm conference on innovation and technology in computer science education* (pp. 284–289). New York: Association for Computing Machinery. doi: 10.1145/2899415.2899432

Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., … Szabo, C. (2018). Introductory programming: A systematic literature review. In *Proceedings companion of the 23rd annual acm conference on innovation and technology in computer science education* (pp. 55–106). New York: ACM.

McCracken, D. D. (1987). Ruminations on computer science curricula. *Communications of the ACM*, *30*(1), 3–6.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., ... Wilusz, T. (2001). A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-Year CS Students. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education* (p. 125–180). New York: ACM. doi: 10.1145/572133.572137

Norman, D. A. (2014). Some observations on mental models. In *Mental models* (pp. 15–22). Psychology Press.

Parlante, N. (2022a). *Codingbat.* Retrieved from `https://codingbat.com/about.html`

Parlante, N. (2022b). *Codingbat recursion 1 factorial.* Retrieved from `https://codingbat.com/prob/p154669`

Parlante, N. (2022c). *Codingbat recursion 1 fibonacci.* Retrieved from `https://codingbat.com/prob/p120015`

Pirolli, P. L., & Anderson, J. R. (1985). The role of learning from examples in the acquisition of recursive programming skills. *Canadian Journal of Psychology/Revue canadienne de psychologie*, *39*(2), 240.

Raj, R., Sabin, M., Impagliazzo, J., Bowers, D., Daniels, M., Hermans, F., ... Oudshoorn, M. (2022). Professional Competencies in Computing Education: Pedagogies and Assessment. In *Proceedings of the 2021 Working Group Reports on Innovation and Technology in Computer Science Education* (p. 133–161). New York: Association for Computing Machinery. doi: 10.1145/3502870.3506570

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137–172.

Sanders, I., Galpin, V., & Götschi, T. (2006). Mental models of recursion revisited. In *Proceedings of the 11th annual sigcse conference on innovation and technology in computer science education* (p. 138–142). New York: ACM.

Scholtz, T. L., & Sanders, I. (2010). Mental models of recursion: Investigating students' understanding of recursion. In *Proceedings of the fifteenth annual conference on innovation and technology in computer science education* (p. 103–107). New York: ACM.

Schwamb, K. (1990). Mental models: A survey.

Spohrer, J. C., & Soloway, E. (1986). Novice mistakes: Are the folk wisdoms correct? *Communications of the ACM*, *29*(7), 624–632.

Stone, C., & Goodyear, P. (1995). Constructivism and instructional design: epistemology and the construction of meaning. *Substratum: Temas Fundamentales en Psicologia y Educacion*, *2(6)*, 55–76.

Tew, A. E., McCracken, W. M., & Guzdial, M. (2005). Impact of alternative introductory courses on programming concept understanding. In *Proceedings of the first international workshop on computing education research* (pp. 25–35).

Troy, M. E., & Early, G. (1992). Unraveling recursion part ii. *Computing Teacher*, *19*(7), 21–25.

Velazquez-Iturbide, J. A. (2000). Recursion in gradual steps (is recursion really that difficult?). In *Proceedings of the thirty-first sigcse technical symposium on computer science education* (pp. 310–314).

Von Glasersfeld, E. (1996). Radikaler Konstruktivismus. *Ideen, Ergebnisse, Probleme. Suhrkamp, Frankfurt/Main*, *375*.

Whalley, J. L., & Lister, R. (2009). The BRACElet 2009.1 (Wellington) Specification. In *Conferences in research and practice in information technology series.*

Wiedenbeck, S. (1988). Learning recursion as a concept and as a programming technique. *ACM SIGCSE Bulletin*, *20*(1), 275–278.

Wilcocks, D., & Sanders, I. (1994). Animating recursion as an aid to instruction. *Computers and Education*, *23*(3), 221–226.

Winslow, L. E. (1996). Programming pedagogy–a psychological overview. *ACM Sigcse Bulletin*, *28*(3), 17–22.

Wu, C.-C., Dale, N. B., & Bethel, L. J. (1998). Conceptual models and cognitive learning styles in teaching recursion. In *Proceedings of the twenty-ninth sigcse technical symposium on computer science education* (pp. 292–296).

Xinogalos, S. (2014). Designing and deploying programming courses: Strategies, tools, difficulties and pedagogy. *Education and Information Technologies*, *21*(3), 559–588.