The Progression of Students' Ability to Work With Scope, Parameter Passing and Aliasing

Filip Strömbäck
Department of Computer and Information Science
Linköping University
Linköping, Sweden
filip.stromback@liu.se

Pontus Haglund
Department of Computer and Information Science
Linköping University
Linköping, Sweden
pontus.haglund@liu.se

Aseel Berglund
Department of Computer and Information Science
Linköping University
Linköping, Sweden
aseel.berglund@liu.se

Erik Berglund
Department of Computer and Information Science
Linköping University
Linköping, Sweden
erik.berglund@liu.se

ABSTRACT

Students need the ability to reason about the behavior of programs when working with advanced concepts like concurrency and abstraction. To achieve this, students require core programming skills that allow them to trace and predict the outcome of a program. While previous research indicates that teachers cannot expect students to acquire all core programming skills after their introductory CS course, less is known of students' progression in later years. In this study, we investigate 397 students' ability to predict the outcome of short computer programs. The participants are from different programs and progressions in their studies. We find that students, regardless of program and year, struggle with predicting the outcome of short programs that require an accurate mental model of some less readily apparent concepts, such as references. Further, we discover that there is no significant improvement in the first three years. Finally, we propose further avenues of research to improve these learning outcomes.

CCS CONCEPTS

• **Applied computing** → **Education**; • **General and reference** → *Empirical studies.*

KEYWORDS

mental model, tracing, CS1, computer science education

ACM Reference Format:
Filip Strömbäck, Pontus Haglund, Aseel Berglund, and Erik Berglund. 2023. The Progression of Students' Ability to Work With Scope, Parameter Passing and Aliasing. In *Australasian Computing Education Conference (ACE '23), January 30-February 3, 2023, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3576123.3576128

ACE '23, January 30-February 3, 2023, Melbourne, VIC, Australia

ACM ISBN 978-1-4503-9941-8/23/01.
https://doi.org/10.1145/3576123.3576128

1 INTRODUCTION

In an environment where software systems grow more complex and where concurrent systems grow ever more prolific, it is increasingly important for students of computer science (CS) to develop a solid understanding of their core programming skills.[1] In particular, skills related to the data model of the programming language, in part since a solid understanding of these skills form an important foundation for more advanced topics.

For example, the ability to reason about scope, parameter passing and aliasing is critical in order to properly reason about the behavior of abstractions in large systems (e.g., accidentally "leaking" internal data through a reference), or to identify shared data in a concurrent program. One reason for students' struggles with these concepts may be that their semantics are not readily visible to students. Previous research has shown that this kind of hidden semantics often leads to misconceptions [21]. For this reason, we refer to these skills as *subtle concepts* in this paper. Later courses typically treat these concepts as prerequisites, and students whose mental models do not accurately describe their semantics tend to struggle with understanding or applying the more advanced concepts.

While students need to have some proficiency with these subtle concepts in order to understand more complex topics, it is unrealistic to expect that students have mastered them after their introductory course(s). Rather, previous research has found that students' programming skills in general are weak after their first CS course [13, 15], both concerning reading and writing computer programs. As such, it is more reasonable to expect that students only have a basic understanding of the core programming concepts after their first CS course and that their proficiency with them increases throughout their education, while they are studying other programming-related subjects.

This paper aims to explore students' grasp of these subtle concepts and students' ability to predict the results produced by programs that require an understanding of these concepts. We do this using the same statistical methods as Haglund et al. [8], who investigated students' mental models using a survey where students were

[1] Fundamental concepts, core concepts, fundamental skills, etc. are all expressions used in this field of research, however, in this paper we will use core programming skills when referring to basic knowledge/skills/understanding within programming.

asked to trace small Python programs. This previous study was limited in that it only investigated students after their first CS course. This paper extends this research by including more students, both from different programs and from different years in their education. By doing this, we aim to determine if students' understanding of the data model (i.e., scope, parameter passing and aliasing) improves throughout their education, and if the earlier findings hold across different programs and programming languages. Based on these results we will then propose further areas of research to improve learning outcomes for the development of students' mental models.

The remainder of this paper is structured as follows: Section 2 introduces related work, Section 3 presents the method, and Section 4 presents the results. The results are then discussed in Section 5, and a conclusion is presented in Section 6.

2 RELATED WORK

2.1 Misconceptions

Students' misconceptions and the relation between misconceptions and core programming skills are subjects with research dating back to the 1980s [2, 3]. In computer science, the term *misconeption* is used to refer to an inaccurate understanding of some aspect of programming [19]. Many misconceptions relate to aspects that are not readily visually apparent to students, such as the semantics of references [21]. Sirkiä and Sorva [21] explain that it is valuable to have a cognitive conflict between students' understanding of a concept and their observations by making these aspects visible. Further, Qian and Lehman [19] underline the importance of not only documenting misconceptions but to strive for research that is oriented towards improving learning outcomes. Ma et al. [14] also studied how students understand references in the early stages of their CS education. Rather than examining the misconceptions in isolation, the authors studied them in the context of what non-viable mental models students used to reason about references. Within this framework, they found a correlation between non-viable mental models and performance among students. They also found that it is difficult to rectify problems in students non-viable mental models, since these non-viable models are accurate enough to describe many (but not all) situations that might arise. Finally, they concluded that a large portion of the students did not apply their mental models consistently when working with reference assignments.

2.2 Core Programming Skills

McCracken et al. [15] summarizes the topics that students are expected to learn during their first year in a computer science program. The skills are summarized in the form of categories that represent critical skills a student needs to be able to solve programming problems. The categories are 1) abstract a problem from its description, 2) generate sub-problems, 3) transform sub-problems into sub-solutions, 4) re-compose those sub-solutions into working programs, and 5) evaluate and iterate. In the study, students were found to lack these skills after their first year, regardless of the education system, country, or programming language used. Lister et al. [13] found that students often lack more rudimentary skills related to code comprehension. In particular, the students were not even able to read and understand code well enough to trace an existing program. Haglund et al. [8] investigated students' use of abstraction for problem-solving and grasping basic programming skills necessary for abstracting problems. They found that many of their students did not use abstraction well while solving problems. Further, they found that students generally lacked the understanding of subtle concepts necessary to understand and apply abstractions. Fisler et al. [7] describe the concepts of scope, mutation, aliasing, and parameter transfer as core skills. They find that these skills are not automatically internalized by students, nor can they be expected to be accrued fully during students' first year of studies in computer science. They suggest that courses need to teach and assess these skills beyond the first year. In addition, they found that knowledge transfer between languages does not occur naturally for students.

2.3 The Importance of Understanding Subtle Concepts

Abstraction is one topic in computer science where it is important for students to have a solid grasp of certain core programming skills [8], including those referred to here as subtle concepts. Abstraction is vital in order to build large and robust software, but it is difficult to teach [1, 9]. Part of this challenge is that abstraction, and other soft ideas within computer science, do not have rigid rules and is not possible to teach in relation to a single topic [9]. Abstraction is also discussed in a wider sense, for example in terms of refactoring code [1] or as a way of understanding algorithms [18] at varying levels. Further, Statter and Armoni [22] claim that students not only have to understand a certain level of abstraction but also must be able to move between the different levels to be able to work with abstractions efficiently. Abstraction is something that is learned over time, thus it is important to start early and teach it continuously to students [11].

Another topic where core programming skills, specifically the subtle concepts, matter is concurrency, which is typically covered later in a CS education. As noted by Kolikant [10], a formal computer science education is a meeting point of two computer-literate cultures: the user culture and the academic culture. Kolikant further argues that learning concurrency is an entry point for students to transition from the user culture into the academic culture due to the non-deterministic nature of concurrent programs. The author observed that many students use a trial-and-error approach when solving concurrency problems. This behavior categorizes the user culture and is generally not a viable approach when working with concurrent programs. Instead, students need to adopt the academic culture that relies more on (formal) reasoning compared to the user culture. The students thus need to shift their focus from the observed behavior of a program to also include a deeper understanding of the computational model in order to be able to reason about the behavior of concurrent programs properly. This computational model is closely related to mental- or conceptual models as defined by Fincher et al. [6].

Further, Strömbäck et al. [23] found that while most students in their study managed to correctly identify concurrency issues by their symptoms, only about half of the students were able to address them properly. The authors hypothesize that one reason for this discrepancy is a lacking understanding of the computational

Table 1: Overview of the amount of CS courses taken by each of the two cohorts at different points in their program. The table shows the number of credits for courses where programming is a central part of achieving the course goals. An asterisk (*) indicates that additional CS courses are available as electives.

	Total CS credits accrued at the end of...			
Cohort	1st course	1st year	2nd year	3rd year
CS masters	6	24	41	49*
	6	34	51	68*
CS bachelors	6	20	44*	52*

model, as it is exceedingly difficult to identify and protect shared data without the ability to reason about the behavior of programs. The same authors further explore students' understanding of the computational model and abstractions in concurrency [24], and find several potential shortcomings students might have with their computational model.

In summary, a solid understanding of core programming skills (including those related to subtle concepts) is important. Otherwise, students are not able to understand the computational model and are thus not able to transition from the user culture of trial-and-error to the academic culture where (formal) reasoning is more prominent.

3 METHOD

3.1 Cohort Description

The data in this study was gathered from 397 CS students at Linköping University. The students all attend one of three CS programs at the university. For the purposes of this study, we divide these students into two cohorts. Students of the *Master of Science in Computer Science and Engineering* and students of the *Master of Science in Computer Science and Software Engineering* programs make up the first cohort, henceforth referred to as CS masters. Students of the *Bachelor of Science in Computer Engineering* make up the second cohort, henceforth referred to as CS bachelors. These cohorts reflect two broad types of computer science programs in terms of the degree they pursue and in the length of their studies. This grouping also reflects the first programming course taken by each of the programs, meaning that the CS masters all take the same introductory CS course and that the CS bachelors take a different introductory CS course. The CS bachelors are expected to finish their studies after 3 years (180 credits) and the CS masters are expected to finish their studies after 5 years (300 credits).

As seen in Table 1 the number of credits[2] students take in courses where programming is a central part varies slightly between the cohorts. The CS masters and CS bachelors are similar, but they accrue their credits at different stages of their education. It is worth noting that the number of credits in CS starts varying in the latter years due to electives. The introductory programming courses offered to both cohorts is 6 credits. Three different languages are used in the introductory course. The CS masters use Python exclusively, and the CS bachelors start learning the fundamentals in Ada (i.e., basic programming constructs) and then transition to C++. These courses are overseen by 2 different teachers at the same department of computer and information science at the university.

After the first course, the two cohorts study different sets of languages. The CS bachelors mainly use C++ throughout the remainder of their education (with some exceptions, e.g., C in their operating systems course). The CS masters have more variation in the languages used. They start in Python, and then have courses in for example C, C#, Java and JavaScript.

3.2 The Survey

The survey consisted of a number of questions that each contained a piece of code and asked students what up to three variables would contain after executing the code. As can be seen from the questions in Fig. 1, multiple choice questions were not used. Rather, the students were free to enter whatever value they feel was correct for each variable. The survey was designed to focus on scope, parameter passing, aliasing (through references) and classes. The Python version of the survey (Fig. 1) was given to the CS masters, as their introductory course is taught in Python. The CS bachelors were given a C++ version of the survey as they mainly work with C++. The C++ version aims to preserve the semantics from the Python version by utilizing appropriate language constructs in C++. In particular, parameters were passed by value or reference as appropriate. In question 5, the variable `c` is a copy of `b` rather than a reference to the same instance. This was done mainly to preserve the similarity between the initializations. While this means that students do not have to realize that `c` is a reference in the C++ version, they still need to keep track of the separate instances of the class. The C++ version is available in full in Fig. 3 at the end of the paper.

For the CS masters, the survey was administered once to all students in the first three years. Students in this cohort were asked to answer the survey under teacher supervision during a session in a course that spans the first three years of this program. Students were provided with a paper copy of the survey and were asked to answer the questions by writing on their copy. The survey was also administered once to the CS bachelors. This time the survey was provided digitally in the form of a web-page. The first year students were asked to complete the survey under teacher supervision during a session of a smaller version of the course the CS masters take. Since this version of the course does not span multiple years, the second-year students were asked to complete the survey during a lecture of another course, supervised by the lecturer. The survey was given in the spring for all participants and any responses that did not answer the two last questions were considered incomplete and thus discarded.

The first page of the survey (both the paper- and the digital versions) contained information about the purpose of the survey. It also informed students that participation was voluntary, and that they would remain anonymous since no personal identifiers would be collected. This means that it is impossible to trace answers back to an individual, and as such this type of research does not need ethics approval in Sweden.

[2]Linköping University university adheres to the ECTS, where 60 credits equals a full year of studies and 1.5 credits equals 1 week (40 hours) of work.

Question 1 (5)

```
def f1(a, b, c):
  a = a + 5
  b.append(1)
  c = [1, 2]

a = 1
b = [1]
c = [1]
f1(a, b, c)
# What is a, b and c here?
```

Answer:
$a = 1, b = [1, 1], c = [1]$

Question 2

```
def f2(a, b):
  a.append(1)
  b.append(2)

a = []
b = []
f2(a, a)
# What is a and b here?
```

Answer:
$a = [1, 2], b = []$

Question 3 (7)

```
b = 10

def f3(a):
  b = 20
  return a + b

c = f3(b)
# What is a, b and c here?
```

Answer:
a is undefined, $b = 10, c = 30$

Question 4 (8)

```
def f4(l):
  l.append(5)

def f5(m):
  m.append(7)

a = []
b = []
c = a
f4(a)
f4(b)
f5(a)
# What is a, b and c here?
```

Answer:
$a = [5, 7], b = [5], c = [5, 7]$

Question 5

```
class C1:
  def __init__(self, a, b):
    self.first = a
    self.second = b

  def inc(self):
    self.first = self.first + 1
    self.second = self.second + 1

def inc(c):
  c.second = c.second + 1
  c.inc()
```

```
# cont. from previous column

a = C1(1, 2)
b = C1(3, 4)
c = b
a.inc()
inc(b)

x = [a.first, a.second]
y = [b.first, b.second]
z = [c.first, c.second]
# What is x, y and z here?
```

Answer:
$x = [2, 3], y = [4, 6], z = [4, 6]$

Figure 1: The five questions in the survey (Python version).

Table 2: Results from coding the prerequisites of each of the questions. Note: All questions include *simple statements*, *assignments*, *tracing*, and *operators*. These are not included in the table.

Part	array iteration	arrays	conditionals	dictionaries	functions: parameters	functions: return	functions: return values	functions: scoping	indirection	loop constructs	values and references	objects: scoping
1a					✓			✓			✓	
1b		✓							✓		✓	
1c		✓			✓			✓			✓	
2a		✓							✓		✓	
2b		✓			✓			✓				
3a								✓				
3b								✓				
3c					✓		✓					
4a		✓			✓				✓			
4b		✓			✓				✓			
4c		✓			✓				✓			
5a												✓
5b					✓			✓	✓		✓	✓
5c					✓			✓	✓		✓	✓

3.3 Measuring Skills

We explore differences in individual core programming skills by using the model proposed by Haglund et al. [8]. This model allows comparing students' proficiency with individual skills between different programs and years. Since this paper does not use the same questions as Haglund et al. we need to find which skills are necessary to answer each question correctly. For this, we use the method and the codebook proposed by Nelson et al. [17]. Two researchers independently coded the skills required to solve each question. After this, they compared their findings and discussed any differences until they reached an agreement. The result of this process is presented in Table 2. For the questions that were identical to those used in [8], the coding from Haglund et al. was used.

These skills were then used to model students' responses in terms of their skills using a generalized linear model (GLM) [16]. We denote each student's proficiency with a particular topic as p_i $(0 \leq p_i \leq 1)$. Using these proficiencies, we can then model the probability of the student answering a particular part of a question correctly as c_j. Using this notation, we can model the relation between the two as follows:

$$c_j \sim \text{Bernoulli}\left(\sum_{i=1}^{n} p_i s_{ij} \Big/ \sum_{i=1}^{n} s_{ij}\right)$$

Here, we use s_{ij} to denote whether part j assesses the topic i. Thus, $s_{ij} = 1$ if the part assesses the topic, and 0 otherwise. As we are interested in finding the proficiencies, p_i, for a particular group

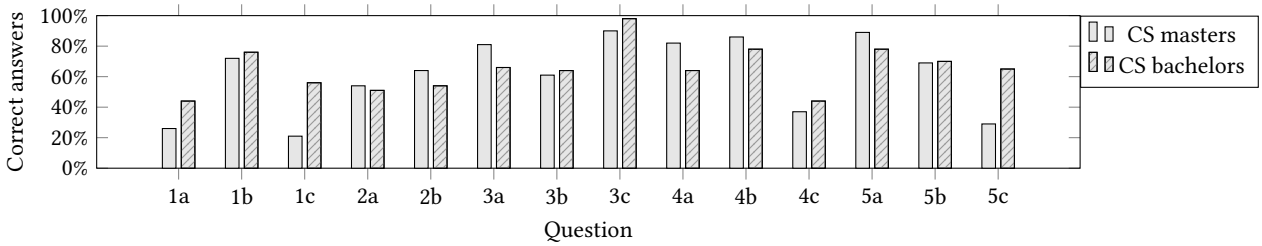

Figure 2: Overview of the two cohorts' answers to the questions.

and are interested in comparing groups of students, we extend the model to include different individuals and groups. Thus, we let c_{jxy} denote whether student x in group y answered part j correctly. This means that the full model is as follows:

$$c_{jxy} \sim \text{Bernoulli}\left(\sum_{i=1}^{n} p_{iy} s_{ij} \Big/ \sum_{i=1}^{n} s_{ij}\right)$$

We then fit the model to the data using a logistic link function to find the values p_{iy}, which can be interpreted as a measure of group y's proficiency with topic i. We then compare different cohorts by testing the pair of hypotheses $H_0 : p_{iy_1} = p_{iy_2}$ vs. $H_1 : p_{iy_1} \neq p_{iy_2}$ for all i.

3.4 Correlation of Incorrect Answers

To further investigate what weaknesses caused incorrect answers, we computed the most common incorrect answers for all questions in the survey. For the questions where more than 50% of all students answered incorrectly we also examined if incorrect answers to these questions were correlated with each other. To test for correlations, we performed pairwise χ^2 tests on the results of the questions. As such, for each pair of questions, a and b, we examined whether a correct answer to question a correlated with a correct answer to question b.

4 RESULTS

Table 3 contains an overview of the performance of each cohort. From the table, we can see that the performance of the two cohorts were similar. Comparing the performance using a Mann-Whitney U-test shows no significant differences between the performance of the cohorts on the survey as a whole.

Table 3: Overview of the answers from the two cohorts. The table shows the number students who answered in each cohort along with the total number of active students in each cohort along with the average percentage of questions answered correctly.

	Active	Answers		
Cohort	students	Total	Incomplete	Score
CS master	397	326	11	63%
CS bachelor	126	69	10	65%
Total	523	395	21	63%

Students' performance on the individual questions are shown in detail in Fig. 2. The figure shows that less than 50% of all students answered 1a, 1c, 4c, and 5c correctly. Comparing the performance on individual questions (again, using a Mann-Whitney U-test) shows no significant differences between the two cohorts, except for questions 1 and 5. The CS masters performed significantly worse than the CS bachelors on both these questions ($p = 0.0010$ for question 1 and $p = 0.0092$ for question 5).

Table 4 shows the performance of students in different years of their education. As shown in the table, there is no significant difference between the years in either of the two cohorts. Rather, the high p-value suggests that the distributions are likely the same. This means that students hardly improve their ability to predict the outcome of short programs that include these subtle concepts during their first three years. A slight but not significant improvement is visible for the CS bachelors who almost exclusively use C++.

4.1 Measuring Skills

Using the model to compare skills between the different years of the CS masters, we found that the *return values* skill decreased significantly between year 1 and year 2 ($p = 0.0029$), and that the *arrays* skill decreased significantly between year 1 and 3 ($p = 0.0355$). For the CS bachelors, we observed an increase in *objects: scoping* ($p = 0.0223$) between year 1 and year 2. Considering the many tests conducted, these p-values are relatively high, and the results should therefore be considered to be fairly weak, as indicated by the tests on the overall scores.

Comparing the two cohorts using the model revealed that CS masters performed better than CS bachelors on the set of skills that were common to all questions, namely *tracing*, *operators*, *simple statements* and *assignments* ($p = 0.0004$). This suggests that the CS masters performed better than the CS bachelors overall. Accounting for this difference, however, the CS bachelors outperformed the CS masters in questions that involvedn the following skills:

- *values and references* ($p < 0.0001$)
- *functions: parameters* ($p = 0.0059$)
- *functions: scoping* ($p = 0.0142$)
- *return values* ($p = 0.0492$)

4.2 The Most Common Incorrect Answers

The most common incorrect answers are presented in Table 5. The table also contains the frequency of incorrect answers overall (Total), and how common the most incorrect answer were out of all incorrect answers (Freq.). Since more than 50% of the answers to

Table 4: Correlation between students' performance between different years. As can be seen, the differences are too small to be significant.

	Year 1		Year 2		Year 3			
Cohort	n	Perf.	n	Perf.	n	Perf.	Trend	Significance
CS masters	123	62%	113	63%	78	64%	$\rightarrow$	$p = 0.96$
CS bachelors	28	63%	30	67%	-	-	$\nearrow$	$p = 0.62$

questions 1a, 1c, 4c, and 5c were incorrect, we consider these questions in further detail in this section. In spite of the two cohorts using different programming languages, the most common incorrect answer was the same for all but 5c. To further examine what misconceptions students might have when arriving at these incorrect answers, we examine each of the four questions in further detail:

Question 1a Almost all students that answered incorrectly stated that the variable would contain the value 6. This indicates that students who answered this question incorrectly did not understand that the variable `a` inside and outside the function were different variables. This is particularly surprising for the students doing the survey in C++ since the passing of a reference is explicit.

Question 1c The majority of CS masters thought that `c` would be modified outside of the function's scope. Even though the CS bachelors performed better on this question, the most common incorrect answer was the same in the two cohorts. It is not surprising that the CS bachelors performed better on this question since C++ is more explicit regarding which parameters are passed by reference.

Question 4c The most common incorrect answer to this question shows that students failed to realize that `c` and `a` refer to the same list. This particular incorrect answer was far more prevalent among the CS masters than among the CS bachelors.

Question 5c For CS masters, the most common incorrect answer to this question was that students failed to realize that variable `c` is a reference to variable `b`, instead believing that `c` was a copy of the contents of `b`. For the CS bachelors, most students realized that it was a copy. The most common answer was that it was undefined (4 students), but many other incorrect answers were present as well. For example, 2 students believed that `c` would be a reference to `b`, as was the case in Python.

In summary, the most common incorrect answers to these questions can be described by a model where modifications to function parameters are always visible to the caller, and where assignments make copies of the assigned value. This model describes the most common incorrect answers to some of the other questions (e.g., 3b) as well, while the most common incorrect answers to other questions are described by an opposite model. For example, the most common incorrect answer to question 1b (while still being fairly rare) indicates that parameters are always copies, rather than always references.

Tables 6 and 7 correlate students' ability to answer these four questions correctly. Since 6 tests were performed for each cohort, we treat pairs where $p < \frac{0.05}{6} \approx 0.008$ as significant. For the CS

Table 5: The most common incorrect answers for both cohorts. Answer contains the most common incorrect answer. Freq. is the frequency of the answer out of all incorrect answers. Total is the percentage of incorrect answers out of all submitted answers. Note: in the C++ version of question 1, the line `b.append(1)` was changed to `b.append(2)`.

	CS master			CS bachelor		
Question	Answer	Freq.	Total	Answer	Freq.	Total
1a	6	100%	74%	6	91%	56%
1b	[1]	71%	9%	[2]	50%	24%
1c	[1, 2]	96%	79%	[1, 2]	46%	44%
2a	[1]	82%	45%	[1]	76%	49%
2b	[2]	77%	36%	[2]	70%	46%
3a	10	75%	18%	10	80%	34%
3b	20	91%	41%	20	81%	36%
3c	40	35%	12%	20	100%	2%
4a	[5]	43%	18%	[5]	62%	36%
4b	[]	44%	14%	[7]	46%	22%
4c	[]	83%	63%	[]	36%	56%
5a	[2, 4]	14%	23%	[2, 2]	12%	29%
5b	[4, 5]	26%	40%	[3, 5]	14%	36%
5c	[3, 4]	61%	74%	(und.)	17%	41%

Table 6: Correlation between questions 1a, 1c, 4c and 5c for the CS masters. Each cell contains the p-value. Significant values are marked in bold.

	1a	1c	4c	5c
1a	**0.0000**	**0.0000**	**0.0000**	**0.0000**
1c	**0.0000**	**0.0000**	0.0670	**0.0051**
4c	**0.0000**	0.0670	**0.0000**	**0.0000**
5c	**0.0000**	**0.0051**	**0.0000**	**0.0000**

Table 7: Correlation between questions 1a, 1c, 4c and 5c for the CS bachelors. Each cell contains the p-value. Significant values are marked in bold.

	1a	1c	4c	5c
1a	**0.0000**	**0.0002**	0.5819	0.5656
1c	**0.0002**	**0.0000**	0.1182	0.0362
4c	0.5819	0.1182	**0.0000**	0.0296
5c	0.5656	0.0362	0.0296	**0.0000**

masters, Table 6 shows that there is a strong correlation between all questions, except between questions 1c and 4c. The situation is different for CS masters. As shown in Table 7, only 1a and 1c are correlated. This lesser degree of correlation might partially be due to the smaller number of CS bachelors, but also due to differences between Python and C++.

5 DISCUSSION

The results in Table 3 show that students answer between 63% and 65% of the questions correctly, and that this is true for students in the first three years of studying CS. Thus, students are not able to predict the outcome of short computer programs that involve understanding the subtle concepts to the extent we had hoped. As previously mentioned in Section 2, these concepts are important later in the education [5] when approaching topics like abstraction [1, 8, 12] and concurrency [23]. The low number of correct answers thus suggests that students will struggle when learning more advanced topics.

In the remainder of this section we discuss the validity of the results and the method, and the results considering our objective. Finally, we present future avenues for extending the work presented in this paper.

5.1 Validity

While the data set used in this paper is fairly large (395 students), the slight differences in the data collection might impact the quality of the data. For example, some of the supervisors that supervised the CS masters when completing the survey allowed limited discussions in the small groups while others did not. Regardless, the same trends are visible througout the data set which suggests that most students answered the survey individually and without testing the code as instructed. In particular, even though some of the CS masters were allowed to discuss the questions to some extent, they still performed worse on some of the most difficult questions compared to the CS bachelors, who were not allowed to discuss their solutions. Furthermore, Chamillard and Braun [5] argue that individual assessments with time constraints, such as the survey used in this paper, correlate well with students' performance on a final exam. This observation further strengthens the reliability of the results.

Another aspect worth considering is the potential selection bias within the cohorts. While the number of students who did not complete the survey were low among the cohorts, a larger portion of the CS bachelors chose not to participate compared to the CS masters. It is therefore possible that the students who chose to participate were the stronger portion of students in the cohort. Further, comparing the performance among those who participated and completed the survey to those who participated and did not complete it showed that the performance on the completed portion was similar. Among the CS bachelors, it was more common not to participate in the survey. This might be due to not all students attending the particular lecture where the survey was administered, or that they used the allotted time for an extra break in the lecture. Since more than half of the students participated (56%) and since the overall performance of the CS bachelors were similar to that of the CS masters (where a vast majority answered and completed the survey), we are confident that the trends visible in our data are representative despite these potential issues.

5.2 Development of Core Programming Skills

In this section, we examine the data related to the development of students' grasp of subtle concepts. Since Lister et al. [13] found that students lack rudimentary code comprehension skills after their first CS course, we expected to find that students' skills (including those related to subtle concepts) would increase throughout their education. While students likely improve their programming skills in general, the data from Table 4 shows that this is not the case when it comes to their ability to predict the outcome of short programs that require grasping these subtle concepts. From Fig. 2 we can further see that the four questions with a total of less than 50% correct answers (1a, 1c, 4c and 5c) all require a good understanding of scope, aliasing and indirection. These concepts have previously been found to be problematic by Fisler et al. [7], who found that students do not develop a solid understanding of them during their first year of studying CS. Further they are concepts where the behavior is not readily apparent, thus being a common area of student misconceptions [21]. As such, our results suggest that students' mental models of these subtle concepts do not improve after the first year. However, most would agree that a third year student is generally a stronger programmer than a first year student. Therefore it is important to not interpret these results to mean that students do not improve in their core programming skills at all. Students do improve in the sense that they are able to solve more complex problems with less guidance. Their mental models do, however, not significantly improve in a way that allows them to better predict the behavior of programs where scope, aliasing and indirection are important. These results also support the claims by Ma et al. [14], that it is difficult for students to re-consider the mental models they use when they are accurate enough to predict some, but not all, situations correctly. This is precisely what we see in the data presented in this paper: students have a mental model that is accurate enough to predict some of the behavior of some of the programs in the survey. Since it is accurate enough, students are not motivated to revise their model to address the remaining inaccuracies.

Since Fisler et al. [7] also found that students' understanding of scope, aliasing and indirection do not transfer between languages, one explanation for the lack of improvement could be that the CS masters only work with Python in their introductory course(s), and then mainly work with other languages. Since these skills do not transfer between languages, any progress made in other languages would not be visible as the survey was conducted in Python. This could also explain why the CS masters' answers to one of the 4 most difficult questions had a strong correlation to their ability to answer the other ones correctly. Furthermore, not working with Python for an extended period might mean that students have forgotten some details of the semantics. Reimer et al. [20] found some evidence of this five months after an intensive introductory course. Some of the participants mentioned that they no longer remembered certain details since they had not worked with Python in a long time, which indicates that this aspect has some impact at least.

However, since we saw no significant improvement in the CS bachelors, who almost exclusively work with C++ during their first two years, the above issues related to not working with Python is likely not the main reason for the lack of improvement, even though more exposure to the language may be beneficial. Furthermore, since the survey for both the CS master and CS bachelors was administered in the spring, any improvements during the first year are not visible.

As such, our results concur with the conclusions of Fisler et al. [7]: that these skills need to be taught explicitly and throughout the curriculum. This observation is similar to the suggestions by Koppelman and van Dijk [11], that abstraction also needs to be taught throughout the curriculum. As it is often difficult for novices to realize the importance of all details of the semantics of programming languages, it is a good idea to illustrate the implications of the rules in the context of new topics. For example, illustrating why the scoping rules are sensible when introducing abstraction through functions and/or objects. This approach helps students to see the reason for certain behaviours and thus avoid the situation where students simply try to memorize all rules, which has been shown to decrease retention [4]. Fisler et al. [7] further suggests that implementing these concepts in a small programming language aids students' understanding of them.

5.3 Differences Between Programs

In this section, we examine the data that relates to the difference in performance between the two programs. As described in Section 3, there are a number of differences between the two cohorts. They attend different introductory courses that are given in different programming languages, and with different goals to reflect the long-term objectives in their respective programs. In spite of these differences, we found no significant difference in the overall performance of the two cohorts.

The lack of difference in overall performance was particularly surprising since the CS masters received the survey in Python while the CS bachelors received it in C++. Since Python does not require type declarations there is no clear indication in the source code of the semantics of parameter passing and assignments. Python is even inconsistent to some extent: the statement `a += b` that appends elements to a list is not equivalent to `a = a + b` as one might expect, since the `+=` operator modifies the list that `a` refers to, rather than creating a new list, and assigning `a` a reference to that list (this is why we preferred `a.append()` in the survey). In contrast, C++ makes this information explicit in the type declarations. For this reason, we initially thought the C++ version of the survey would be much easier than the Python version, in particular question 1. To our surprise, this turned out not to be true, since the two cohorts performed similarly on almost all questions. In particular, more than 50% of the CS bachelors answered question 1a incorrectly.

By comparing the most common incorrect answers for the two cohorts (Table 5) we find that the most common incorrect answers to all questions except for 3c, 4b, 5a, 5b and 5c were the same for both cohorts (taking into account the change to question 1b in the C++ version). This suggests that students struggle with a similar set of concepts regardless of which programming language was used. We can, however, see that the CS bachelors performed slightly better on three of the most difficult questions (1a, 1c and 4c) compared to the CS masters. Since all of these questions require a good understanding of references, this slight increase is likely either due to the more explicit nature of C++ (i.e., references are denoted explicitly in the source code), or that the CS bachelors have focused on C++ in their first years rather than using different languages as was the case with the CS masters.

While the most common incorrect answers were similar between the two cohorts, the correlation between correct answers to different questions revealed an interesting difference between the two cohorts. For the CS masters (Table 6), we found a correlation between all of the four most difficult questions (1a, 1c, 4c and 5c) except between 1c and 4c. As described in Section 4.2, the most common incorrect answers for these questions all correspond to a model where modifications to parameters are always visible to the caller, and where assignments make copies of data. As such, these results suggest that CS masters at least had some level of consistency when answering the questions. Other questions with a higher number of correct answers, such as 1b, show that some students used other models. This particular incorrect answer can be explained with a model where parameters are always copied, for example. We saw some consistency in this type of answers as well, but we have not analyzed all possible models in detail as it was not the main focus of this paper.

Table 7 shows that the CS bachelors seem to be less consistent when answering the questions. For this cohort, we only found a correlation between questions 1a and 1c. This may, of course, be in part due to the smaller number of students in this cohort, but it is likely not the only factor. In this case, a likely explanation is that students have not yet realized the similarities between different situations. For example, question 1 utilizes references in parameter passing. The students seem to understand this fairly well, perhaps since it is important to consider the impact of using reference parameters in C++. Questions 4c also utilizes references, but in this case for variables instead of function parameters. This situation is less common, at least in early stages of learning C++, and the lack of correlation to questions 1a and 1c suggests that students have not yet realized that reference variables behave the same as reference parameters. Finally, that correct answers to question 5c are not correlated to other questions is not entirely surprising as the code in this version of the survey makes a copy of the class. Based on the results to question 4c, it seems like this is the behavior students expect, regardless of whether or not references are used. However, since question 5c in C++ is fairly simple (the answer is the values in the code), we would have expected that students had performed better on this question. In summary, our findings for the CS bachelors are similar to those of Ma et al. [14]: that this cohort does not necessarily apply their mental model of these concepts consistently. However, the CS masters seem to do so to a larger extent.

Finally, the comparison between the different programs using the statistical model (Section 4.1) indicated that the CS bachelors performed worse overall compared to the CS masters, but that the CS bachelors were stronger in some areas, for example values and references. Interestingly enough, these areas roughly correspond to the aspects where C++ is more explicit than Python (i.e., *functions: parameters*, *functions: scoping*, and *values and references*).

Question 1

```
int a{ 1 };
vector<int> b{ 1 };
vector<int> c{ 1 };

void f1(int a, vector<int> &b,
               vector<int> c) {
    a = a + 5;
    b.push_back(2);
    c.push_back(2);
}
int main() {
    f1(a, b, c);
    // What is a, b and c here?
}
```

Answer:
$a = 1, b = [1, 2], c = [1]$

Question 2

```
vector<int> a{};
vector<int> b{};

void f2(vector<int> &a,
        vector<int> &b) {
    a.push_back(1);
    b.push_back(2);
}
int main() {
    f2(a, a);
    // What is a and b here?
}
```

Answer:
$a = [1, 2], b = []$

Question 3

```
int b = 10;

int f3(int a) {
    int b = 20;
    return a + b;
}
int main() {
    int c = f3(b);
    // What is a, b and c here?
}
```

Answer:
a is undefined, $b = 10, c = 30$

Question 4

```
void f4(vector<int> &l) {
    l.push_back(5);
}
void f5(vector<int> &m) {
    m.push_back(7);
}
int main() {
    vector<int> a{};
    vector<int> b{};
    vector<int> &c{a};
    f4(a);
    f4(b);
    f5(a);
    // What is a, b and c here?
}
```

Answer:
$a = [5, 7], b = [5], c = [5, 7]$

Question 5

```
class C1 {
public:
    C1(int a, int b)
        : first{a}, second{b} {}
    int first;
    int second;
    void inc() {
        first = first + 1;
        second = second + 1;
    }
};
void inc(C1 &c) {
    c.second = c.second + 1;
    c.inc();
}
```

```
// Cont. from previous column

int main() {
    C1 a{1, 2};
    C1 b{3, 4};
    C1 c{b};
    a.inc();
    inc(b);

    vector<int> x{a.first, a.second};
    vector<int> y{b.first, b.second};
    vector<int> z{c.first, c.second};
    // What is x, y and z here?
}
```

Answer:
$x = [2, 3], y = [4, 6], z = [3, 4]$

Figure 3: The five questions in the survey (C++ version). Includes and using namespace std; are omitted for brevity. Note that questions 1 and 5 differs slightly from the Python version. In question 1, 2 is appended to b instead of 1 to be able to differentiate incorrect answers. In question 5, the variable c is a copy and not a reference.

6 CONCLUSION

As argued in the introduction, it is important that students are able to properly reason about the data model when working with different types of abstractions or with concurrent programs. Otherwise, students will have difficulties with identifying situations where aliasing occurrs, which will lead to situations where abstractions will fail to work as intended, or failure to properly synchronize shared data properly. As such, acquiring this ability is an important desired outcome of a computer science education. The findings of this study reinforce previous findings [7] that students' mental models are not accurate enough to allow them to reason about and predict the behavior of short programs involving parameter passing, references, indirection, and scoping after their first computer science course. Further, the findings of this study show that students further on in their academic pursuits do not significantly improve in their ability to reason about and predict the behavior of programs that utilize these subtle concepts, even in their third year of studying computer science. This should not, however, be interpreted as students not progressing in their programming skills. These findings relate to students' ability to predict the outcome of programs that include subtle concepts and not their general ability to write programs of greater complexity. It does however indicate that students do not develop mental models that can accurately predict the semantics of these kinds of short programs. They likely develop strategies to overcome this shortcoming as most would agree that a third-year student is in general a stronger programmer than a first-year student. Though, as discussed, this has implications, especially when working with problems where trial-and-error is impossible or impractical (e.g., refactoring of large code bases). While the nature of this study does not allow us to answer the question of how to improve these outcomes, it compels us to ask it.

To this end, we propose the application of a framework known as constructive alignment to study how the activities students take part in and the assessments administered to them align with desired outcomes. The goal is to better understand why these misconceptions, relating to subtle concepts exist, and how to improve the actual outcomes. We propose conducting the following studies:

(1) A phenomenographic analysis of teaching staff (TAs and Teachers) views of these subtle concepts in their courses with regard to activities.
(2) A phenomenographic analysis of teaching staff (TAs and Teachers) views of these subtle concepts in their courses with regard to assessments.
(3) An analysis of current activities provided to students with a focus on how they relate to these subtle concepts.
(4) An analysis of current assessments provided to students with a focus on how they relate to these subtle concepts.

REFERENCES

[1] Russ Abbott and Chengyu Sun. 2008. Abstraction Abstracted. In *Proceedings of the 2nd International Workshop on The Role of Abstraction in Software Engineering* (Leipzig, Germany) *(ROA '08)*. Association for Computing Machinery, New York, NY, USA, 23–30. https://doi.org/10.1145/1370164.1370171

[2] Piraye Bayman and Richard E. Mayer. 1983. A Diagnosis of Beginning Programmers' Misconceptions of BASIC Programming Statements. *Commun. ACM* 26, 9 (sep 1983), 677–679. https://doi.org/10.1145/358172.358408

[3] Benedict Du Boulay. 1986. Some Difficulties of Learning to Program. *Journal of Educational Computing Research* 2, 1 (1986), 57–73. https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9 arXiv:https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9

[4] Naomi R. Boyer, Sara Langevin, and Alessio Gaspar. 2008. Self Direction & Constructivism in Programming Education. In *Proceedings of the 9th ACM SIGITE Conference on Information Technology Education* (Cincinnati, OH, USA) *(SIGITE '08)*. ACM, New York, NY, USA, 89–94. https://doi.org/10.1145/1414558.1414585

[5] A. T. Chamillard and Kim A. Braun. 2000. Evaluating Programming Ability in an Introductory Computer Science Course. In *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education* (Austin, Texas, USA) *(SIGCSE '00)*. Association for Computing Machinery, New York, NY, USA, 212–216. https://doi.org/10.1145/330908.331857

[6] Sally Fincher, Johan Jeuring, Craig S. Miller, Peter Donaldson, Benedict du Boulay, Matthias Hauswirth, Arto Hellas, Felienne Hermans, Colleen Lewis, Andreas Mühling, Janice L. Pearce, and Andrew Petersen. 2020. Notional Machines in Computing Education: The Education of Attention. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE-WGR '20)*. Association for Computing Machinery, New York, NY, USA, 21–50. https://doi.org/10.1145/3437800.3439202

[7] Kathi Fisler, Shriram Krishnamurthi, and Preston Tunnell Wilson. 2017. Assessing and Teaching Scope, Mutation, and Aliasing in Upper-Level Undergraduates. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 213–218. https://doi.org/10.1145/3017680.3017777

[8] Pontus Haglund, Filip Strömbäck, and Linda Mannila. 2021. Understanding Students' Failure to use Functions as a Tool for Abstraction – An Analysis of Questionnaire Responses and Lab Assignments in a CS1 Python Course. *Informatics in Education* 20, 4 (2021), 583–614. https://doi.org/10.15388/infedu.2021.26

[9] Orit Hazzan. 2008. Reflections on Teaching Abstraction and Other Soft Ideas. *SIGCSE Bull.* 40, 2 (June 2008), 40–43. https://doi.org/10.1145/1383602.1383631

[10] Yifat Ben-David Kolikant. 2004. Learning Concurrency as an Entry Point to the Community of Computer Science Practitioners. *Journal of Computers in Mathematics and Science Teaching* 23, 1 (2004), 21–46. https://www.learntechlib.org/p/12871

[11] Herman Koppelman and Betsy van Dijk. 2010. Teaching Abstraction in Introductory Courses. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education* (Bilkent, Ankara, Turkey) *(ITiCSE '10)*. Association for Computing Machinery, New York, NY, USA, 174–178. https://doi.org/10.1145/1822090.1822140

[12] Barbara Liskov and John Guttag. 2000. *Program Development in Java: Abstraction, Specification, and Object-Oriented Design* (1st ed.). Addison-Wesley Longman Publishing Co., Inc., USA.

[13] Raymond Lister, Elizabeth S. Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, Beth Simon, and Lynda Thomas. 2004. A Multi-National Study of Reading and Tracing Skills in Novice Programmers. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education* (Leeds, United Kingdom) *(ITiCSE-WGR '04)*. Association for Computing Machinery, New York, NY, USA, 119–150. https://doi.org/10.1145/1044550.1041673

[14] Linxiao Ma, John Ferguson, Marc Roper, and Murray Wood. 2007. Investigating the Viability of Mental Models Held by Novice Programmers. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* (Covington, Kentucky, USA) *(SIGCSE '07)*. ACM, New York, NY, USA, 499–503. https://doi.org/10.1145/1227310.1227481

[15] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz Wilusz. 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* (Canterbury, UK) *(ITiCSE-WGR '01)*. Association for Computing Machinery, New York, NY, USA, 125–180. https://doi.org/10.1145/572133.572137

[16] J. A. Nelder and R. W. M. Wedderburn. 1972. Generalized Linear Models. *Journal of the Royal Statistical Society. Series A (General)* 135, 3 (1972), 370–384. https://doi.org/10.2307/2344614

[17] Greg L. Nelson, Filip Strömbäck, Ari Korhonen, Marjahan Begum, Ben Blamey, Karen H. Jin, Violetta Lonati, Bonnie MacKellar, and Mattia Monga. 2020. Differentiated Assessments for Advanced Courses That Reveal Issues with Prerequisite Skills: A Design Investigation. In *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (Trondheim, Norway) *(ITiCSE-WGR '20)*. Association for Computing Machinery, New York, NY, USA, 75–129. https://doi.org/10.1145/3437800.3439204

[18] Jacob Perrenet and Eric Kaasenbrood. 2006. Levels of Abstraction in Students' Understanding of the Concept of Algorithm: The Qualitative Perspective. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Bologna, Italy) *(ITICSE '06)*. Association for Computing Machinery, New York, NY, USA, 270–274. https://doi.org/10.1145/1140124.1140196

[19] Yizhou Qian and James Lehman. 2017. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Comput. Educ.* 18, 1, Article 1 (oct 2017), 24 pages. https://doi.org/10.1145/3077618

[20] Yolanda J. Reimer, Michael Coe, Lisa M. Blank, and Jeffrey Braun. 2018. Effects of Professional Development on Programming Knowledge and Self-Efficacy. In *2018 IEEE Frontiers in Education Conference (FIE)*. 1–8. https://doi.org/10.1109/FIE.2018.8659041

[21] Teemu Sirkiä and Juha Sorva. 2012. Exploring Programming Misconceptions: An Analysis of Student Mistakes in Visual Program Simulation Exercises. In *Proceedings of the 12th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '12)*. Association for Computing Machinery, New York, NY, USA, 19–28. https://doi.org/10.1145/2401796.2401799

[22] David Statter and Michal Armoni. 2020. Teaching Abstraction in Computer Science to 7th Grade Students. *ACM Trans. Comput. Educ.* 20, 1, Article 8 (jan 2020), 37 pages. https://doi.org/10.1145/3372143

[23] Filip Strömbäck, Linda Mannila, Mikael Asplund, and Mariam Kamkar. 2019. A Student's View of Concurrency - A Study of Common Mistakes in Introductory Courses on Concurrency. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (Toronto ON, Canada) *(ICER '19)*. Association for Computing Machinery, New York, NY, USA, 229–237. https://doi.org/10.1145/3291279.3339415

[24] Filip Strömbäck, Linda Mannila, and Mariam Kamkar. 2021. The Non-Deterministic Path to Concurrency – Exploring how Students Understand the Abstractions of Concurrency. *Informatics in Education* 20, 4 (2021), 683–715. https://doi.org/10.15388/infedu.2021.29