Learning to use Functions: The Relationship Between Misconceptions and Self-Efficacy*

Maria Kallia
King's College London
London
maria.kallia@kcl.ac.uk

Sue Sentance
King's College London
London
sue.sentance@kcl.ac.uk

ABSTRACT

Computer programming is one of the most researched subjects within computer science education; within this much attention has been focused on exploring the difficulties and common misconceptions that students experience when learning to program. The study reported here has two aims: firstly, to investigate students' misconceptions around functions by setting up a programming test of advancing difficulty and complexity based on the Bloom and Solo taxonomies, and secondly, to explore the impact that misconceptions have on students' self-efficacy in programming, along with students' self-evaluation and self-efficacy in computer science. Our study revealed seven misconceptions in the area of functions, three of which have never before been reported in the literature, to our knowledge. Additionally, the results suggest that misconceptions do not only prohibit students' progress and learning but have a significant impact on students' self-efficacy in programming.

CCS CONCEPTS

• **Social and professional topics** → **CS1**; **K-12 education**;

KEYWORDS

misconceptions in programming, functions, self-efficacy

ACM Reference Format:
Maria Kallia and Sue Sentance. 2019. Learning to use Functions: The Relationship Between Misconceptions and Self-Efficacy. In *SIGCSE '19: 50th ACM Technical Symposium on Computer Science Education, February 27–March 2, 2019, Minneapolis, MN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3287324.3287377

1 INTRODUCTION

Increasingly, computer programming is becoming integrated into secondary schools across the world, playing a central role in the computer science curriculum. This specific aspect of computer science has attracted much research attention due to the acknowledged and multifaceted difficulties that it causes students, some of which are known as misconceptions, false conceptions or misunderstandings [27]. Undoubtedly, identifying, understanding and addressing students' misunderstandings and misconceptions in programming is vital and imperative for helping students to reach a more precise conceptualisation [15]. Particularly in computer programming this needs to be conducted as early as possible, as misconceptions that are left unresolved can render a student unable to further continue with this subject and may also have an impact on students' self-efficacy in programming and in computer science.

In this study we were particularly interested in exploring students' misconceptions by creating assessment tasks of graded complexity, enabling us to capture a wide range of students' errors. In comparison with most existing studies, we decided to limit our research to one specific area in programming to uncover as many misconceptions as possible. Therefore, we have chosen to investigate misconceptions in the area of *functions* as the literature indicates that this is one of the most difficult parts of programming [16, 20, 32] and because from our teaching experience we felt that there are more misconceptions in these area than the ones that have already been reported. Secondly, we were also interested in investigating the potential impact that misconceptions may have on students' self-efficacy in computer programming by taking into consideration students' self-evaluation on the programming test and students' perceived ability in computer science.

The study endeavours to provide answers to the following research questions:

- RQ1: What are the misconceptions of 16 year old (11th Grade) students in the area of functions?
- RQ2: Do students' misconceptions impact students' self-efficacy in programming and how does the self-evaluation and efficacy in computer science may contribute to this?

To answer these questions, we employed a mixed methods research design. We first collected and qualitatively analysed students' responses in six different programming tasks with increased complexity. The analysis of this phase resulted in the identification of 7 misconceptions in functions, of which 3 have never before been reported as misconceptions in the literature, to the authors' knowledge. In the second phase, we quantitatively analysed students' performance and their responses on self-efficacy in programming and computer science with the SPSS software package. To this end, a correlation and a causal comparative group design were employed. In total, 83 students participated in the study.

*Produces the permission block, and copyright information

SIGCSE '19, February 27–March 2, 2019, Minneapolis, MN, USA

ACM ISBN 978-1-4503-5890-3/19/02.
https://doi.org/10.1145/3287324.3287377

2 BACKGROUND

2.1 Misconceptions in Sciences

Children, according to Piaget [24], search for understanding by interacting with the surrounding environment; learning comes as a combination of previous experience and present mental schemata which are developed through assimilation and accommodation. However, experiences do not always develop correct interpretations [38] and thus, the problem is that once misconceptions take seed in a child's mind, it is very difficult for them to be altered [11]. In fact, Longfield[19] asserts that humans have the tendency to hold on to information that lines up with their existing theories or understanding while they reject or abandon those that will put them to disequilibrium. Cognitive disequilibrium is a distressing state where humans try to remain in equilibrium and to achieve this, they discard information that does not align with their views [24]. Ben-Ari[5] postulates that the conceptual distance between students' own version of understanding and the scientific view determines students' existing misconceptions. Misconceptions, therefore, are developed by a person's experience [21] and represent an incorrect understanding of an idea which the literature has characterised as a misconception, preconception, naive belief, alternative conceptions or misunderstandings [35].

Most of the research work conducted around students' misconceptions is concentrated in the sciences where misconceptions have been separated into five distinct categories: *Preconceived Notions, Nonscientific Beliefs, Conceptual Misunderstandings, Vernacular Misconceptions, Factual Misconceptions* [9]. In computer programming, identifying students' errors or misunderstandings and misconceptions started early in around 80s. For example, one of the first studies conducted in this area was conducted by Perkins and Martin [23] who identified four types of fragile knowledge: *Partial knowledge, Inert knowledge, Misplaced knowledge* and *conglomerated knowledge*. Du Boulay [10] has also contributed significantly in this research area and emphasised concrete sources of students' misconceptions in programming: *Misapplication of analogy*, *Overgeneralisations* and *Interactions*. Du Boulay also referred to specific problematic areas in programming which he categorises into *Orientation, Notional machine, Notation, Structures*, and *Pragmatics*.

To identify students' misconceptions, several methodologies have been proposed. Kaczmarczyk et al. [15] used semi-structured interviews with a modified think-aloud protocol in which students discussed a subset of Java problems and the analysis of students' transcripts was based on grounded theory and qualitative analysis. Based on this work, qualitative analysis was also used by Veerasamy et al. [40] to reveal misconceptions in introductory programming in Python. Fleury [12] used interviews with students and a collection of Java programs while Sirkiä [32] studied students' errors in python by examining the log files from students' solution to visual program simulation exercises.

2.2 Bloom's and SOLO Taxonomies

In designing assessment tools for programming, many researchers have turned their attention to the Bloom and Solo taxonomies. Bloom's taxonomy is centered around the planning of educational objectives in a hierarchical manner, starting from less complex levels to more advanced ones. Specifically, in the revised Bloom's taxonomy, proposed by Anderson et al. [1], the taxonomy levels are *Remembering, Understanding, Applying, Analysing, Evaluating*, and *Creating*. Bloom's taxonomy has been used in computer science as an assessment method to categorise the assessment questions. Some of the studies adapted Bloom's taxonomy include Lister and Leaney[17], Starr et al.[36] and Thompson et al.[37].

The SOLO taxonomy, suggested by Biggs and Collis[6], is another way of evaluating students' performance. The specific taxonomy is also hierarchical and is mostly used to classify students' levels of understanding in the following levels: *Prestructural, Unistructural, Multistructural, Relational*, and *Extended abstract*. In computer science, the taxonomy has been successfully used for categorizing students' responses on examination (e.g. [13, 18]).

2.3 Self-Efficacy in learning

Self-efficacy is a construct rooted in the social cognitive theory of Albert Bandura [3]. As such, Bandura and Schunk [4, p. 31] define self-efficacy beliefs as *"people's judgment of their capabilities to organize and execute courses of action required to attain designated types of performances"*. In other words, self-efficacy is a construct referring to an individual's perceived capabilities in a specific task.

In academic and school settings, self-efficacy has attracted tremendous attention, especially in research that focuses mostly on the affective-emotional aspects of learning. There is generally a common agreement among these studies that students that have high perceptions of their capabilities engaged more, are better motivated and perform better than students that have low feelings of competence [4, 45]. Zimmerman [43] also highlighted the importance of self-efficacy on students' self-regulation learning. For example, students with high self-efficacy arrange their working schedule in a more efficient way than the students who demonstrate lower levels of efficacy [7] while also use learning strategies and self-evaluation processes that promote learning and evaluate the outcome of their self-monitoring correspondingly [44, 46].

Therefore, monitoring students' level of self-efficacy is a process that is critical and necessary for educators who are interested in investigating not only their students' actual performance but also factors that may contribute to this performance. To our knowledge, there are no studies that have investigated students' levels of self-efficacy in programming and how these levels differ from students who demonstrate misconceptions and students that do not. This is the gap that this study endeavors to address.

3 METHODOLOGY

3.1 Students' misconceptions

To explore students' misconceptions in functions, qualitative analysis, and specifically content analysis, was employed. Content analysis is a systematic coding and categorizing approach which is employed to investigate trends and patterns, their frequency and relationships in textual information [39].

Incorporating content analysis in computer programming is not a direct process. Shah et al. [30] presented a qualitative content analysis approach for programming errors which includes four steps: paraphrasing, generalization, categorization and check. Paraphrasing is a process that removes the code components not related with

the error (not content-bearing). Generalisation involves the description of the error in natural language and categorisation includes the process of producing more abstract versions of the generalisations made previously to produce categories that include similar errors. Finally, the check refers to the validation process of the errors and the category to which they have been assigned.

Based on the aforementioned approach described, we explored and categorised students' misconceptions in functions.

3.2 Students' misconceptions and self-efficacy in programming and computer science

For the second phase of the study, we employed a correlation and a causal comparative research design. To explore the relationship between misconceptions and students' self-efficacy, we employed an independent t test in which we compare the self-efficacy levels of students that showed evidence of misconceptions and students that did not. Additionally, to explore the relationship between self-efficacy in programming, self-evaluation on the programming test and computer science, correlation analysis was used. In particular Spearman's rho correlation was used since the data for all the examined variables, except efficacy in programming, were not normally distributed.

3.3 Participants

To answer the research questions of the study, we considered a homogeneous purposive sample. We were very careful to select students that had sufficient experience in programming and also in functions. Students' previous experience in functions was particular important to our study as the literature suggests that misconceptions are extremely tenacious and resistant to change. Therefore, we wanted the students to have the time to develop misconceptions in functions and hold onto these. Thus, we first contacted computer science teachers and explicitly asked them to participate in our study with their computing class only if the latter fulfilled the aforementioned criteria.

In total, 83 students (11th Grade/Year 12) participated in the study from 4 different schools, of which 18 were girls and 65 were boys. The test was written in Python as students had been exposed to this language mostly.

3.4 Instruments and Data collection

To investigate misconceptions in functions, we created 6 programming tasks that captured different levels of complexity. On designing our assessment tool, we took into consideration Scott's [29] argument that assessment tests should include questions from all six levels of Bloom's taxonomy as most of the students are able to respond to low-level questions and few students can answer the most complex questions. In this way, we had a more objective way to determine what each student had understood or not. The most influential study, however, was Shuhidan et al.'s [31], in which Bloom's taxonomy was used to classify the multiple-choice questions and SOLO taxonomy to evaluate students' responses. In the same vein, in our assessment test, Bloom's taxonomy was employed to categorise the questions to Bloom's different levels and SOLO taxonomy was used to evaluate students' level of understanding. Extra consideration was given to Whalley and Kasto's [41] paper in which they categorise different types of programming tasks into Bloom's and SOLO taxonomy. Building on these studies, we created 6 programming tasks of advancing difficulty which are presented in the end of this paper.

The first task asked the students to match the concepts of parameters, arguments, return statement, function definition, calling a function and print statement with the appropriate annotated text in the given code. The aim here was to see if the students had a basic understanding of the concepts that are included in this part of the curriculum. The second task included a short program in Python which included two functions and asked the students to identify and correct the errors (both syntax and logic). This task was a little more advanced than the first one as it included logical errors in addition to syntactical and, thus, a deeper understanding of functions was required by the students. The third task asked the students to read a program written in Python and answer two reading comprehension questions: *"What is the purpose of the program"* and *"What will be the output of that program"*. The fourth task was a Parson's Puzzle: the students were asked to put in the correct order a group of code statements in order for the program to succeed its purpose (it was given in written form). The fifth task was a fill-in-the-blanks exercise focusing on parameters, arguments and return variables, and the sixth task asked the students to write from scratch a program that creates and calls a function that calculates the area of a triangle.

When the students finished the programming tasks, they continued with the self-evaluation and self-efficacy questions. Specifically, the students responded to two 11-point(0-10) Likert-scale questions about how they evaluate themselves on the test (How do you think you did on the test from 0 to 10?) and their general efficacy in computer science (How do you think you get on with computer science course from 0 to 10?). These two questions were employed from Román-González et al.'s study [28]. Additionally, to measure the self-efficacy in computer programming we employed the Pintrich et al.'s [25] self-efficacy scale which was adapted to reflect the computer programming course and has a Cronbach's alpha of .93. We should note here that this self-efficacy scale (for programming) was given first before students move on to the programming tasks. The reason we distributed this questionnaire first was because we didn't want students' self-efficacy in programming to be influenced by their specific performance on the programming tasks.

The data were collected through an online survey tool and the link to the test was given to teachers. Each teacher dedicated an hour of to the test and the accompanied questionnaire.

3.5 Validity and Reliability

To ensure the validity and reliability of the interpretation of the misconceptions, two researchers were involved in the process: the first researcher produced the coder manual with the categories (misconceptions) which was used by the second researcher to categorise again the students' errors. The inter-rater reliability of Cohen's kappa was calculated and a substantial agreement was found: $k=0.675$, $p < .001$ (approximate 95% confidence interval on Kappa: 0.612 - 0.737). Only categories that both researchers agreed upon are presented in this paper.

Table 1: Students' misconceptions

Misconceptions	Percentage of students	Task
1: Students think that return statements should be used after the call to the function to return and print the function's output	48.19%	Task: 4 and 6
2: Students think that return value should have the same name with the function	24%	Task: 6
3: Students think that parameters should be explicitly defined to be used inside the function	19.27%	Task: 2
4: Students think that parameters and arguments should have the same name	15.6%	Task: 5 and 2
5: Students think that arguments should include the return variable	12%	Task: 5
6: Students think that local variables can be accessed from a function to another	10.8%	Task: 6
7: Students think that calling a function prints the result	8.4%	Task: 6

Figure 1: Examples: Students' misconceptions

Example – Misconception 1 (Task 4)

```
def output (grade):
  if grade > 12:
    result = "pass"
  else:
     result = "fail"

def main ():
   mark = int (input("insert grade:"))
   print (output(mark))
   return result
```

Example – Misconception 2 (Task 6)

```
def area (a, b):
  area = a*b
  return area

def main ():
  height = int (input("Enter the rectangle's height:"))
  width = int (input("Enter the rectangle's width: "))
  print (area (height, width))
```

4 DATA ANALYSIS AND RESULTS

4.1 Students' Misconceptions in Functions

The aim of the programming tasks analysis was to discover common errors that students make in programming and to uncover misconceptions that may held by the students in the area of functions. Table 1 summarises the misconceptions that the study uncovered, the percentage of students that demonstrated the misconception and the task that illustrated the misconception. Figure 1 depicts examples from students' code that demonstrate misconception 1 and 2.

4.2 Students' performance and self-efficacy on task and computer science

Table 2 shows the descriptive statistics of students' performance, self-evaluation, self-efficacy in computer science and self-efficacy in computer programming. The sample is split in two groups: students that demonstrated the misconceptions and students that did not.

To examine whether students' level of self-efficacy in computer programming differ between students that hold misconceptions and students that do not we employed the independent t test to compare the levels of self-efficacy between the two groups.

The results suggest that the self-efficacy levels in programming of students that hold misconceptions are significantly lower than students that did not show evidence of misconceptions ($t=3.614$, $p<.05$, $r=.41$) .

Additionally, to examine whether students' levels of self-efficacy in computer science differ between students that hold misconceptions and students that do not we employed the Mann-Whitney test since some of the data were not normally distributed.

The results suggest that the self-efficacy levels in computer science of students that hold misconceptions are significantly lower than students that did not show evidence of misconceptions ($U=386.0$, $z=-3.415$, $d=0.374$).

5 DISCUSSION

In this section we will discuss the research findings by answering the research questions addressed in the first part of this paper.

5.1 Students' misconceptions in Functions

The first research question related to students' misconceptions in functions. As table 1 illustrates, our study revealed seven misconceptions students demonstrate in this part of the curriculum. The first misconception that was made by a large number of students (48.19%) points out to a very specific error: students put the return statement under the calling statement in order to print and return the result of the function. Our interpretation is that students regard that putting the calling statement first and then putting the return statement is the way to return and print the result of a function. Another error that students made quite often in the tasks – and it relates again to the return value – is that students named the

Table 2: Descriptive Statistics

Misconceptions				Performance	Self-Evaluation	EfficacyCS	EfficacyProgramming
No	N	Valid	25				
		Missing	0				
	Mean			91.2	7.44	7.40	5.366
	Median			90.0	8.00	7.00	5.25
	Mode			88.0	8	7	6.0
	Std. Deviation			5.375	1.873	1.472	.8623
Yes	N	Valid	58				
		Missing	0				
	Mean			43.0	5.16	5.93	4.39
	Median			41.0	5.0	6.0	4.43
	Mode			29.5	5	6	3.5
	Std, Deviation			17.74	2.033	1.746	1.216

Table 3: Independent t test

						95% Confidence Interval of the Difference
	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower|Upper
efficacyProgramming	3.614	81	.001	.971110	.268728	.436426|1.505794

Table 4: Mann-Whitney U test

	performance	efficacyCS	self-evaluation
Mann-Whitney U	.000	386.000	287.5
Wilcoxon W	1711.000	2097.0	1998.5
Z	-7.198	-3.415	-4.386
Asymp. Sig. (2-tailed)	.000	.001	.000

return variable the same as their function. Although it is not clear to us if this error happened by mistake, the fact that 24% of students demonstrated this in their code led us to believe that some students erroneously believe that the name of the variable that must be returned should be the same as the name of the function. Finally, our study found one more misconception regarding return values: some students put the return variable to the arguments list (misconception 5). Our interpretation is that the students thought that in order to return a value back, the variable holding this value must be placed in the arguments. Although previous researchers [33, 34] have reported errors that students make with return values, this is the first time, to our knowledge, that these specific ways of handling return values are reported in the literature.

A comparison of the findings with those of other studies confirms that misconceptions 3, 4, 6 have been reported again in the literature. Specifically, misconception 3 refers to students' erroneous understanding that parameters must be assigned a value inside the function's body in order to be used. The same misconception was reported by Sirkiä and Sorva[33]. Additionally, misconception 6 which refers to students' erroneous belief that local variables in the function can be accessed outside the function was also reported by Sirkiä and Sorva [34]. Misconception 4 is about students' misunderstandings and confusion about naming parameters and arguments. Chen et al. [8] referred to students' misunderstandings about formal and actual parameters and therefore we categorise these error to a general confusion that students have with these concepts. Finally, our study found one other misconception: few students thought that calling a function will also print the result. Although we have included this misconception in the list, we are reluctant to support that this error stems from a misconception as only a few students demonstrated this error.

5.2 Do students' misconceptions impact students' self-efficacy in programming? What is the role of self-evaluation and efficacy in computer science?

Very little can be found in the literature on the question of if and how students' misconceptions can impact students' self-efficacy beliefs in programming. To this end, this study's second aim was to explore this relationship by comparing the self-efficacy levels between students that showed evidence of possessing the aforementioned misconceptions and students that did not. The results depicted in table 2 and 3 suggest that students who hold misconceptions in programming demonstrate significant lower levels of self-efficacy in programming than do students that do not hold misconceptions (r=0.41). The results are not surprising as students that demonstrated these misconceptions scored lower than the other students and there are many studies that correlate performance with self-efficacy beliefs [26, 42]. Nevertheless, the results highlight an important and often neglected factor that accompanied students to the journey from false beliefs to accurate conceptualizations.

Previous studies have noted the importance of previous experience in computing and students' self-efficacy in programming [2, 22]. In our study it is evident that students' efficacy beliefs in computer science are significantly different between the two groups with the group demonstrating misconceptions achieving lower levels of efficacy. This was somehow expected as the correlation between efficacy in computer science and efficacy in computer

Table 5: Correlations

			efficacyProgramming	Performance	self-evaluation	efficacyCS
Spearman's rho	efficacyProgramming	Correlation Coefficient	.1000	.546**	.639**	.714**
		Sig. (2 tailed)	.	.000	.000	.000
		N	83	83	83	83

**Correlation is significant at the 0.01 level (2-tailed).

programming is strong as table 5 depicts. On similar grounds, the study of Wiedenbeck et al. [42] showed that previous experience in programming affects self-efficacy which in turn impacts students' success in this course. The role of previous performance in programming and how this is contributing to students' self-efficacy in programming was also highlighted in the study conducted by Jegede [14] who found that students' performance in programming courses significantly predicts students' self-efficacy in programming.

Another interesting finding stems from students' self-evaluations. It seems that students who didn't have misconceptions tend to underestimate their performance while students with misconceptions tend to overestimate their performance. This is interesting indeed as it may suggest that students with misconceptions hold on to these mistaken ideas strongly and therefore they cannot accurately estimate their performance.

6 CONCLUSION

The current study set out to explore misconceptions that 11th Grade (Year 12) students hold with regards to functions, and secondly to investigate if students that demonstrate misconceptions experience statistically significant lower levels of self-efficacy in programming than students that do not. We also included two other variables that the literature suggests as potential factors impacting students' self-efficacy in programming: students' capability beliefs in computer science and their self-evaluations on the specific programming test.

One of the most significant findings to emerge from this study is the identification of 7 misconceptions, of which 3 have never before been identified in the literature to our knowledge. These misconceptions reflect the fact that the return variable seems to be confusing to students at this age and stage. The second major finding was that misconceptions in programming may have an impact on students' self-efficacy in programming. The results indicated that students who do not demonstrate these misconceptions had statistically significant higher levels of self-efficacy in programming and self-efficacy in computer science.

From a methodological point of view, we believe that our approach – which included a specific part of the computing curriculum along with an assessment tool based on the Blooms and SOLO taxonomies – is an effective approach for capturing the complexity of a programming area and depicting students' misunderstandings at multiple levels. From a pedagogical point of view, our study contributes to the research field of misconceptions and identified three new misconceptions in functions. As a final point, by linking misconceptions and other factors with self-efficacy in programming we have highlighted affective aspects of programming. These are often neglected and we recommend that more research focuses on the relationship between affective and cognitive factors within the learning and teaching of programming.

A APPENDICES

Programming Tasks

Task 4

The purpose of the following program is to print the result of a test (fail or pass). To pass the test the grade should be higher than 12. The code statements are numbered but are misplaced. Can you put the statements in the right order in order for the program to achieve its goal?

```
mark = int(input(" insert grade: "))
result = "fail"
result = "pass"
print (output(mark))
else:
if(grade>12):
def main ():
return result
def output (grade):
```

Task 5

The following program takes as an input two numbers and calculates their sum by calling a function. Fill in the blanks in order for the program to achieve its purpose.

```
def sum (____A_____):
  ___B___ = n1 + n2
  return result

def main ():
  number1 = int(input("type the first number"))
  number2 = int(input("type the second number"))
  print sum (____C____)
```

REFERENCES

[1] Lorin W Anderson, David R Krathwohl, Peter W Airasian, Kathleen A Cruikshank, Richard E Mayer, Paul R Pintrich, James Raths, and Merlin C Wittrock. 2001. A taxonomy for learning, teaching, and assessing: A revision of BloomâĂŹs taxonomy of educational objectives, abridged edition. *White Plains, NY: Longman* (2001).

[2] Petek Askar and David Davenport. 2009. An Investigation of Factors Related to Self-Efficacy for Java Programming among Engineering Students. *Online Submission* 8, 1 (2009).

[3] Albert Bandura. 1977. Self-efficacy: toward a unifying theory of behavioral change. *Psychological review* 84, 2 (1977), 191.

[4] Albert Bandura and Dale H Schunk. 1981. Cultivating competence, self-efficacy, and intrinsic interest through proximal self-motivation. *Journal of personality and social psychology* 41, 3 (1981), 586.

[5] Mordechai Ben-Ari. 2001. Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching* 20, 1 (2001), 45–73.

[6] JB Biggs and KF Collis. [n. d.]. Evaluating the Quality of Learning: The SOLO Taxonomy, 1982.

[7] Therese Bouffard-Bouchard, Sophie Parent, and Serge Larivee. 1991. Influence of self-efficacy on self-regulation and performance among junior and senior high-school age students. *International Journal of Behavioral Development* 14, 2 (1991), 153–164.

[8] C.L. Chen, S.Y. Cheng, and J.M.C. Lin. 2010. A study of misconceptions and missing conceptions of novice Java programmers. In *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS' 12)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 84–89.

[9] National Research Council, Division of Behavioral, Social Sciences, and Board on Science Education; Committee on Undergraduate Science Education Education. 1997. *Science teaching reconsidered: A handbook*. National Academies Press.

[10] Benedict Du Boulay. 1986. Some difficulties of learning to program. *Journal of Educational Computing Research* 2, 1 (1986), 57–73.

[11] Paul Eggen and Don Kauchak. 2001. *Educational Psycology: Windows on Classrooms. 8th.* Upper Saddle River, NJ: Pearson.

[12] Ann E Fleury. 2000. Programming in Java: student-constructed rules. In *ACM SIGCSE Bulletin*, Vol. 32. ACM, 197–201.

[13] Cruz Izu, Amali Weerasinghe, and Cheryl Pope. 2016. A study of code design skills in novice programmers using the SOLO taxonomy. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*. ACM, 251–259.

[14] Philip Olu Jegede. 2009. Predictors of java programming self efficacy among engineering students in a Nigerian University. *arXiv preprint arXiv:0909.0074* (2009).

[15] Lisa C Kaczmarczyk, Elizabeth R Petrick, J Philip East, and Geoffrey L Herman. 2010. Identifying student misconceptions of programming. In *Proceedings of the 41st ACM technical symposium on Computer science education*. ACM, 107–111.

[16] Maria Kallia and Sue Sentance. 2017. Computing Teachers' Perspectives on Threshold Concepts: Functions and Procedural Abstraction. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*. ACM, 15–24.

[17] Raymond Lister and John Leaney. 2003. Introductory programming, criterion-referencing, and bloom. *ACM SIGCSE Bulletin* 35, 1 (2003), 143–147.

[18] Raymond Lister, Beth Simon, Errol Thompson, Jacqueline L Whalley, and Christine Prasad. 2006. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. *ACM SIGCSE Bulletin* 38, 3 (2006), 118–122.

[19] Judith Longfield. 2009. Discrepant teaching events: Using an inquiry stance to address students' misconceptions. *International Journal of Teaching and Learning in Higher Education* 21, 2 (2009), 266.

[20] S. Madison and J. Gifford. 1997. *Parameter passing: The conceptions novices construct*. Technical Report. https://eric.ed.gov/?id=ED406211

[21] Ralph E Martin, Colleen M Sexton, and Jack A Gerlovich. 2002. *Teaching science for all children: Methods for constructing understanding*. Allyn and Bacon.

[22] J Owolabi and BA Adegoke. 2014. Multilevel Analysis of Factors Predicting Self Efficacy in Computer Programming. *International Journal on Integrating Technology in Education (IJITE)* 3, 2 (2014).

[23] DN Perkins and Fay Martin. 1986. Fragile knowledge and neglected strategies in novice programmers. In *first workshop on empirical studies of programmers on Empirical studies of programmers*. 213–229.

[24] Jean Piaget. 1971. Biology and knowledge: An essay on the relations between organic regulations and cognitive processes. (1971).

[25] P.R. Pintrich, D.A.F. Smith, T. Garcia, and W.J McKeachie. 1991. A manual for the use of the Motivated Strategies for Learning Questionnaire (MSLQ). (1991).

[26] Sarantos Psycharis and Maria Kallia. 2017. The effects of computer programming on high school students' reasoning skills and mathematical self-efficacy and problem solving. *Instructional Science* 45, 5 (2017), 583–602.

[27] Yizhou Qian and James Lehman. 2017. Students' misconceptions and other difficulties in introductory programming: a literature review. *ACM Transactions on Computing Education (TOCE)* 18, 1 (2017), 1.

[28] Marcos Román-González, Juan-Carlos Pérez-González, Jesús Moreno-León, and Gregorio Robles. 2017. Extending the nomological network of computational thinking with non-cognitive factors. *Computers in Human Behavior* 80, 2018 (2017), 441e459.

[29] Terry Scott. 2003. Bloom's taxonomy applied to testing in computer science classes. *Journal of Computing Sciences in Colleges* 19, 1 (2003), 267–274.

[30] Philipp Shah, Marc Berges, and Peter Hubwieser. 2017. Qualitative content analysis of programming errors. In *Proceedings of the 5th International Conference on Information and Education Technology*. ACM, 161–166.

[31] Shuhaida Shuhidan, Margaret Hamilton, and Daryl D'Souza. 2009. A taxonomic study of novice programming summative assessment. In *Proceedings of the Eleventh Australasian Conference on Computing Education-Volume 95*. Australian Computer Society, Inc., 147–156.

[32] T Sirkiä. 2012. *Recognizing Programming Misconceptions–An Analysis of the Data Collected from the UUhistle Program Simulation Tool*. Department of Computer Science and Engineering, Aalto University. Master Thesis.

[33] 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*. ACM, 19–28.

[34] Teemu Sirkiä and J Sorva. 2012. Recognizing Programming Misconceptions: An Analysis of the Data Collected from the UUhistle Program Simulation Tool. *Master's thesis, Department of Computer Science and Engineering, Aalto University* (2012).

[35] John P Smith III, Andrea A DiSessa, and Jeremy Roschelle. 1994. Misconceptions reconceived: A constructivist analysis of knowledge in transition. *The journal of the learning sciences* 3, 2 (1994), 115–163.

[36] Christopher W Starr, Bill Manaris, and RoxAnn H Stalvey. 2008. Bloom's taxonomy revisited: specifying assessable learning objectives in computer science. In *ACM SIGCSE Bulletin*, Vol. 40. ACM, 261–265.

[37] Errol Thompson, Andrew Luxton-Reilly, Jacqueline L Whalley, Minjie Hu, and Phil Robbins. 2008. Bloom's taxonomy for CS assessment. In *Proceedings of the tenth conference on Australasian computing education-Volume 78*. Australian Computer Society, Inc., 155–161.

[38] Fiona Thompson and Sue Logue. 2006. An exploration of common student misconceptions in science. *International Education Journal* 7, 4 (2006), 553–559.

[39] Mojtaba Vaismoradi, Hannele Turunen, and Terese Bondas. 2013. Content analysis and thematic analysis: Implications for conducting a qualitative descriptive study. *Nursing & health sciences* 15, 3 (2013), 398–405.

[40] Ashok Kumar Veerasamy, Daryl D'Souza, and Mikko-Jussi Laakso. 2016. Identifying novice student programming misconceptions and errors from summative assessments. *Journal of Educational Technology Systems* 45, 1 (2016), 50–73.

[41] Jacqueline Whalley and Nadia Kasto. 2013. Revisiting models of human conceptualisation in the context of a programming examination. In *Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136*. Australian Computer Society, Inc., 67–76.

[42] Susan Wiedenbeck, Deborah Labelle, and Vennila NR Kain. 2004. Factors affecting course outcomes in introductory programming. In *16th Annual Workshop of the Psychology of Programming Interest Group*. 97–109.

[43] Barry J Zimmerman. 2000. Attaining self-regulation: A social cognitive perspective. In *Handbook of self-regulation*. Elsevier, 13–39.

[44] Barry J Zimmerman and Albert Bandura. 1994. Impact of self-regulatory influences on writing course attainment. *American educational research journal* 31, 4 (1994), 845–862.

[45] Barry J Zimmerman and Anastasia Kitsantas. 1997. Developmental phases in self-regulation: Shifting from process goals to outcome goals. *Journal of educational psychology* 89, 1 (1997), 29.

[46] Barry J Zimmerman and Manuel Martinez-Pons. 1990. Student differences in self-regulated learning: Relating grade, sex, and giftedness to self-efficacy and strategy use. *Journal of educational Psychology* 82, 1 (1990), 51.