Teachers' Views and Experiences on Teaching Second and Subsequent Programming Languages

Ethel Tshukudu
University of Glasgow
Computing Science
Glasgow, UK
ethel.tshukudu@glasgow.ac.uk

Quintin Cutts
University of Glasgow
Computing Science
Glasgow, UK
quintin.cutts@glasgow.ac.uk

Olivier Goletti
UCLouvain
ICTEAM/INGI
Louvain-la-Neuve, Belgium
Leiden University
LIACS
Leiden, The Netherlands
olivier.goletti@uclouvain.be

Alaaeddin Swidan
Open University of the Netherlands
Computer Science
Heerlen, The Netherlands
alaaeddin.swidan@ou.nl

Felienne Hermans
Leiden University
LIACS
Leiden, The Netherlands
f.f.j.hermans@liacs.leidenuniv.nl

ABSTRACT

Motivation More and more high schools are teaching programming, and in many cases, teachers teach multiple programming languages to the same group of students. **Objectives** The goal of this paper is to explore the views of high-school teachers on second and subsequent programming languages, including their motivation for teaching multiple languages, their struggles, and their use of *transfer strategies* when they teach their second or third programming language. **Method** The study consists of semi-structured interviews with 23 high-school teachers in two European countries. **Results** Our findings indicate that school pupils face the same issues as university students when moving from first to subsequent languages. Furthermore, the teachers' attitudes towards second language learning are highly variable, both positive and negative, with some supportive teaching strategies used, but many less helpful ones in evidence too. **Discussion** Our findings suggest that the value of second language learning needs to be highlighted in teacher professional development materials more strongly and that teachers might need more support in implementing transfer strategies.

CCS CONCEPTS

• **Social and professional topics → Computer science education**.

KEYWORDS

programming languages; transfer; code comprehension; conceptual development; syntax; semantics; K12

ACM Reference Format:
Ethel Tshukudu, Quintin Cutts, Olivier Goletti, Alaaeddin Swidan, and Felienne Hermans. 2021. Teachers' Views and Experiences on Teaching Second and Subsequent Programming Languages . In *Proceedings of the 17th ACM Conference on International Computing Education Research (ICER 2021), August 16–19, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3446871.3469752

1 INTRODUCTION

Computer Science (CS) programs at universities tend to involve more than one programming language (PL). University sequences will often include some mix of object-oriented (e.g. Java), systems-level (e.g. C), and functional (e.g. Scheme) languages, as well as domain-specific languages, for example, databases and the web, such as SQL and HTML. Now that programming is increasingly taught in high school too, national school curricula (e.g. the US-based K-12 CS Framework [1], and the English CS curriculum for schools [2]) also include the expectation to teach multiple programming languages.

The learning of a second or subsequent programming language has been a subject of study for many decades. While earlier work concentrated largely on the issues faced by professional programmers when switching programming languages [3–7], more recent research has focused on near novice programmers and their progression from first to second languages [8–13]. Research on programming language transfer has reported that conceptual transfer from the first to subsequent programming language often does not occur as expected in the classroom. Students often show little ability to apply knowledge of what they learnt in previous languages taught in the classroom to new languages, and this is especially the case for abstract concepts [8, 14–16]. Furthermore, research reports that negative transfer from the previous language may impede learning of a new language [3, 11, 17]. Despite the reported failures

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICER 2021, August 16–19, 2021, Virtual Event, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8326-4/21/08…$15.00
https://doi.org/10.1145/3446871.3469752

of transfer, some successes have been reported too [5, 10, 14]. These usually occur when previous languages share syntax and semantic similarities with new languages.

In order to strengthen transfer, prior work has examined teaching for transfer in programming in higher education [13, 16]. These studies have shown that when educators focus specifically on transfer in the classroom, it might improve student learning. Therefore teachers' understanding of transfer strategies is crucial to the quality of computing education. Exploring their views and methods of transfer strategies can allow us to identify specific needs for teacher education. Unlike prior work, which mostly explored teaching for transfer at tertiary education institutions, we focus on secondary education (K-12).

Therefore, the goal of this paper is to report on the current practices and attitudes of high-school teachers towards second language learning, and teachers' awareness, from their own classroom experiences, of the transfer issues. To answer our four research questions, twenty-three K-12 teachers, each from a different school in two European countries were interviewed.

As a starting point, we focus on the reasons for teaching multiple languages. The approach a teacher takes to teaching a second language will be significantly shaped by their beliefs on the value and purpose of teaching the second language, and so our first research question is:

RQ1: Why do teachers teach a second programming language?

It is unclear whether school pupils encounter the same transfer issues when learning their second programming language as university students. Hence our second research question is:

RQ2: What kinds of benefits/problems do teachers notice when teaching second or subsequent programming languages?

We also explore how subsequent programming languages are taught, with a special emphasis on the transfer strategies between languages. It could be the case that high-school teachers, who typically receive more extensive training than university professors, are already aware of and using methods of teaching that take account of the positive and negative issues associated with transfer. Hence our final research questions are:

RQ3: What are the views of computing teachers on the use of transfer strategies?

RQ4: What types of transfer strategies do computing teachers use?

Analysis of the 23 teacher interviews supports the following findings: teachers have a range of motivations for teaching more than one programming language, some of which are unlikely to align well with successful transfer learning; transfer issues are experienced in school classrooms, just as in universities; teachers are mostly aware of these transfer issues to some extent, but do little to make use of this knowledge actively in their approach to teaching a second language; some teachers use supportive teaching strategies, but many less helpful strategies are used too.

2 BACKGROUND

In this section we discuss prior work on the challenges that students face when learning programming. We also cover work on transfer between programming languages and existing instructional strategies favoring transfer.

2.1 Research on different programming languages

Students in computing education are constantly faced with navigating between different programming languages in their education. Many University Computer Science (CS) departments teach students multiple programming languages in introductory programming courses. For example, Davies et al.[18] reported nationwide survey results of 371 undergraduate CS departments in the US on languages and methods they use in a three-course introductory sequence (CS0-CS1-CS2).[1] This study reported that the most common languages used in a CS0 course were Alice, Python, Visual Basic, and JavaScript, with Alice being the most popular. After completing CS0, students are often required to take CS1 or/and CS2 which mostly teach programming with different object-oriented languages such as Java and C++.[18]. Teaching of a visual programming language such as Scratch in CS0 before transitioning to Java in CS1 is also reported in other studies [19, 20]. In some cases, institutions adopted a text based language such as Python before transitioning to Java [21].

Similar approaches are adopted at some primary and secondary schools. For example, in the UK computing curriculum, the requirements before age 11 can be fully satisfied in Scratch. However, when students are at the age of 11–14, the curriculum recommends that they should use two or more programming languages, at least one of which is textual, to solve a variety of computational problems [2].

2.2 Issues with Syntax in Programming Education

Many aspects of learning the first programming language are challenging. The first issue that learners struggle with is the syntax of programming languages. The precision that is needed while programming is often a factor contributing to the difficulty in learning programming. It requires *"a level of attention to detail that does not come naturally to human beings"* [22]. Denny et al. found that weak students submit source code with syntax errors in 73% of cases and, even the best students do so in 50% of cases [23]. Altadmri and Brown [24] analyzed 37 million compilations by 250,000 students and found that the most common error to be a syntax error: mismatched brackets, which occurred in almost 800,000 compilations. Other researchers found that common programming languages like Java and Perl are harder to understand than a random language, stressing the difficulties that novices face in understanding syntax [25]. When it comes to learning the second programming language, syntax remains an issue. Studies have shown that during initial stages of learning a new language, whether programming [26] or natural [27], students make syntax matches between the prior language and the new language. Attempts to address these

[1] In this context, CS0 is the introductory course with no pre-requisites while CS1 and CS2 are the first and second required courses in the sequence respectively

syntax transition issues have resulted in tools to support interfaces with both visual and corresponding text language for students to form analogies at the syntax level as they transition [10, 12, 28, 29]. Interestingly enough, students assess learning syntax as more problematic than teachers [30], which might help us understand why little effort is given to the explicit explanation of syntax in many programming classrooms.

2.3 Research on transfer between programming languages

Research suggests that blocks-based programming environments, although successful in changing attitudes and motivating learners [8], may not sufficiently prepare them to transition to text-based programming languages [11, 31, 32]. However, a study by Weintrop and Wilensky [11] reported that block-based programming environments could help if the teacher indeed taught with transfer in mind. Powers et al. [31] reported transition problems after observing students transitioning from 7 weeks of learning Alice to Java within a semester. They reported that the students struggled with Java syntax and some could not map the concepts such as object declarations, state variables, and methods in Java with the concepts they had learnt in Alice. Another study [32] with similar findings reported that students who had Pascal knowledge and were learning Alice could not make the logical connection of control flow concepts (e.g. loops) between these two languages without explicit instruction. Weintrop and Wilensky [11] also reported that students struggled with Java errors such as undefined variables and incompatible types when transitioning from block-based languages to Java.

Other studies have reported transition problems that students often struggle with when transitioning between different textual languages [14, 16, 17, 33]. For example, in their experiment of students transitioning from Pascal to a new Ada language, Walker and Schach [17] reported that some students preferred to write Ada programs using a familiar *while loop* construct from their prior language instead of switching to the Ada *loop-exit-end* structure. This means that they failed to take advantage of the new features the new language offered. Another study [16] reported that programs that students developed in a new language (object-oriented Java) were influenced by the knowledge from their prior language (Racket functional language). Other studies reported plan transfer when experienced programmers solved problems in a new language [3–5].

Despite the negative effects of transfer reported, researchers have also reported on some positive transfer effects. Successful transfer occurs especially when the prior language and the new language are similar. For example, Scholtz and Wiedenbeck [3] reported that it was easier for students to learn the looping constructs in the new Icon language because they looked and worked in a similar way to the looping constructs in the Pascal language that they learned before Icon.

An effort to help understand the transfer process better is reported in the recent validated Model of Programming Language Transfer (MPLT) by Tshukudu and Cutts [15]. The MPLT describes the transition process and how novice programmers' learning of programming concepts is affected during the shift [15]. In the model, learners automatically effect a transfer of semantics between languages based on syntax similarities, which can result in both positive and negative transfer. Furthermore, they suggest that the transition to new languages may sometimes not be as automatic as educators may expect and this can be a challenge for relatively novice programmers. We believe that instruction can be designed to promote the transfer of learning.

2.4 Known transfer strategies

Thorndike and Woodworth [34] were among the first to assess learning transfer through formal tests and put forward that learning was not about general skill. Since transfer involves prior knowledge, reinforcing initial learning is the first step to consider. Actively identifying and activating prior knowledge can help transfer instead of seeing it as a passive process. For example, Klahr and Carver have assessed LOGO initial learning and linked it to transfer to related tasks [35]. In addition, context matters, and learning transfer is enhanced when concepts are taught in multiple contexts instead of overly contextualised examples [36]. Self-regulation strategies can help students in the process of learning and transferring knowledge [37]. Learning transfer also benefits from a more abstract understanding of the concepts [35]. This indeed shows a more advanced stage of learning in the SOLO taxonomy [38], also often used to assess levels of novice's understanding in computer science [39, 40]. Developing this more abstract comprehension can be done by highlighting the main subgoals in the instructional material [41] or by comparing and contrasting worked examples [42]. Goletti et al. [43] also suggest that using explicit instructional practice favors transfer.

Perkins and Salomon [44] propose two specific techniques to promote the positive transfer of learning, respectively, Hugging and Bridging: Hugging is aimed at increasing the similarities between the learned and target domains while Bridging promotes more abstract conceptual associations between the initial learning and its target domain. Prior work that adopted these transfer techniques is mostly in the context of transfer from block-based languages to text-based languages [10–12, 45, 46]. For example, in their experiment of transitioning students from Alice to Python, Tabet et al. [45] used the bridging techniques to transfer concepts from Alice to Python. This intervention was carried out by explicitly showing similar examples in both Alice and Python (e.g. *loops, if-statements, methods*) to help students effectively transfer concepts. The results showed that students who were in the transfer intervention found it easier to transition from Alice to Python than those who were not. Another study by Dann et al. [10] used the bridging technique by using a hybrid environment that allowed students to switch between blocks and text notation when transitioning from Alice to Java. These studies show that transfer strategies can be successful in helping students transfer however they do not cater for negative transfer, which has been reported to persist in second language learning [15].

To cater for both positive and negative transfer, Tshukudu and Jensen [13] conducted a transfer intervention aligned to the MPLT. The empirical study aimed to investigate the effectiveness of explicit instruction in teaching second programming language constructs. The study was on undergraduate students who were transitioning

from Python to Java language. Students were allowed to compare and contrast semantic similarities and differences between equivalent code fragments in Java (new language) and Python (known language) and any misconceptions were explicitly corrected. The results showed benefits in conceptual transfer and correction of misconceptions caused by the negative semantic transfer.

We are not aware of research that investigates current school teachers' experiences, attitudes and approaches towards second language learning, hence we make this our starting point.

3 RESEARCH SETUP

The goal of this paper is to **understand computing teachers' views and practices on the use of transfer strategies in teaching second and subsequent programming languages**. To that end, we have conducted interviews with 23 in-service teachers, 13 in English and 10 in Dutch.

3.1 Participants

Twenty-three K12 computing in-service teachers, each from a different school in two European countries participated in the study: participants T1-T23. Participation in the study was voluntary, and the teachers were assured of confidentiality and anonymity of responses before the interviews. We recruited the teachers via multiple channels. In Scotland, we made direct contact with a list of teachers who are interested in CS research. In the Netherlands, we put an announcement in a weekly newsletter of i&i[2]: an association of Computer Science teachers in primary and secondary schools in the Netherlands. In addition, participants were contacted through a Twitter account publicly that is followed by many teachers. As a result, the participants in this study are positioned towards the end of the spectrum where teachers are actively engaging in research activities and attending conferences that promote new ideas and activities in CS education.

Table 1 provides an overview of the teachers' demographics and teaching-related information. Their experience in teaching Computing ranges from 5 years to 35 years. The participants all teach at public secondary/high schools, while two teachers have additional teaching load related to the final year of primary school. The teachers we interviewed teach students with ages ranging from 11 years to 18 years. In terms of the curriculum that the teachers follow, there are differences between the two countries. In Scotland, there exists a national CS curriculum where pupils in the first two years of the secondary/high school (S1 and S2) will typically follow one 50-minute session per week, although this is sometimes more or less, and there is no mandatory exam. Starting from the third-year high school (S3) pupils who select computing science subject start having between two or four 50-minute sessions per week in addition to getting examinations. In the Netherlands, there exists no national curriculum but guidelines on what to teach digital literacy and computer science in secondary/high school, which are split into domains or themes. Pupils do not have mandatory exams when following these domains throughout the years of their school. As a result, teachers in the Netherlands have more flexibility and responsibility in deciding what and how to teach when it comes to Computer Science and how to evaluate it for their pupils.

[2]https://ieni.org/

3.2 Interview protocol

The semi-structured interviews were conducted by the authors, the scripted questions were supplemented by follow-up questions, probes and comments where needed.

In order to answer RQ1 on teachers' motivation to teach a second programming language, we asked them to justify their choice of programming languages and the learning outcomes they were pursuing when teaching it. To answer RQ2 on the problems and benefits of teaching more than one language, we then asked questions about whether they observed advantages or drawbacks when teaching a subsequent programming language and whether it related to the first or previous programming language. To answer RQ3 on their views on transfer strategies, we asked them questions on the perceived impact of previously taught languages on students learning the second/subsequent programming language and if they thought it was beneficial to implement transfer strategies. To answer RQ4 on the transfer strategies they use in their teaching, we asked them to describe their programming language teaching methods; specific points of attention were how they referred to the first or previous programming language(s).

To collect open-ended data on teachers' thoughts and beliefs about the transfer and teaching of second and subsequent programming languages, semi-structured interviews were used. The interviews with each teacher lasted 50 minutes on average. All interviews were performed through video calls, recorded, transcribed, and then coded. All teachers were requested to sign a written consent form. In the form, we briefly introduced the goal of the research and the guarantees that any data taken from the interviews will be anonymized. The interview script used for the study can be found at *http://ccse.ac.uk/studymaterials*.

3.3 Coding process

A thematic analysis [47] was used to analyze and interpret themes within the data. The transcribed interviews were imported into Excel for detailed coding analysis. The coding process was in three distinct phases. Firstly, each of the four research questions was coded by one of the four authors individually. The author then discussed the initial codes with one other author, and conflicts were discussed and resolved. Small themes of codes were merged, and larger themes were split using an iterative process. In that process, we sought to find themes that described the core issues of one research question. We aimed to have about three large themes per research question, potentially with sub-questions. The discussions among the first and second coder often concerned the precise grouping of themes and sub-themes, as well as the in-depth discussion of why some quotes were or were not included in (sub)themes.

Finally, all codes and their implications were discussed with the remaining two authors, who were not involved in the initial coding process or the subsequent discussion. In this final consolidation step, the codes were also compared across research questions where relevant, to prevent overlap between research questions and to find generic themes to discuss in-depth in the discussion section.

4 RESULTS

The data revealed several themes, which will be presented in four subsections corresponding to the research questions.

Code	Gender	Country	School level (pupils age)	Teaching experience	Programming languages taught
T1	Male	Scotland	Secondary/High School (12-18 years)	21 years	Scratch, Blocky and micro:bit block editor,HTML, Python
T2	Male	Scotland	Secondary/High School (12-18 years)	10 years	Scratch, Scratch and micro:bit block editor,HTML, Python, PHP
T3	Male	Scotland	Secondary/High School (12-18 years)	18 years	Scratch, HTML, Python, PHP
T4	Female	Netherlands	Secondary/High School (15-17 years)	5 years	Python, HTML/CSS JavaScript, C#
T5	Male	Netherlands	Secondary/High School (15-16 years)	5 years	HTML, Scratch, Javascript
T6	Male	Netherlands	Secondary/High School (15-17 years)	5 years	Processing (Java), Arduino, Web (PHP), Python
T7	Male	Netherlands	Secondary/High School (16-18 years)	11 years	HTML, Python, Java
T8	Female	Scotland	Secondary/High School (12-18 years)	15 years	Blocky, Scratch and HTML, Python and SQL, Python, HTML/CSS Javascript, MySQL and PHP
T9	Female	Scotland	Secondary/High School (12-18 years)	30 years	Blocky, Scratch, Python
T10	Male	Scotland	Secondary/High School (12-18 years)	35 years	Blocky, Scratch, Java, HTML
T11	Male	Scotland	Secondary/High School (12-18 years)	16 years	Blocky, Scratch ,Javascript and VB, Javascript
T12	Female	Scotland	Primary school (8-11 years) Secondary/High School (12-18 years)	19 years	Scratch, micro:bit block editor, Turtle Python, Python
T13	Male	Netherlands	Secondary/High School (16-18 years)	35 years	HTML, JavaScript, PHP, Python
T14	Male	Scotland	Secondary/High School (12-18 years)	24 years	Scratch(S1), HTML/CSS SQL, VB, Javascript, PHP MySQL
T15	Male	Scotland	Secondary/High School (12-18 years)	15 years	Scratch, Python
T16	Male	Scotland	Secondary/High School (12-18 years)	23 years	Truebasic, Scratch,VB, Python, Java
T17	Female	Netherlands	Secondary/High School (16-18 years)	14 years	HTML/CSS, Flowcharts Flowgarithm, Python
T18	Male	Netherlands	Secondary/High School (16-18 years)	3 years	Python, Database SQL, JavaScript
T19	Male	Netherlands	Secondary/High School (13-18 years)	14 years	Scratch, Python
T20	Male	Netherlands	Secondary/High School (16-18 years)	5 years	HTML/CSS, JavaScript, AJAX and JSON, PHP MySQL, Python
T21	Male	Netherlands	Secondary/High School (16-18 years)	32 years	Python, PHP
T22	Female	Scotland	Primary school (11-12 years) Secondary/High School (12-18 years)	23 years	Micro:bit block editor, Scratch, Python, Javascript SQL HTML/CSS, PHP
T23	Male	Scotland	Secondary/High School (12-18 years)	15 years	Scratch, Python

Table 1: Details over the participating teachers. Teacher's code reflects the order of their interview, where T1 is the first interviewed and T23 is the last interviewed.

4.1 RQ1: Reasons for multiple languages

When exploring the reasons high-school teachers provide for teaching multiple languages, we found three main reasons: languages are part of the curriculum or there is teaching material for specific languages (7 teachers), starting with a simple language (13 teachers), and different languages have different impacts on students (10 teachers). The latter two categories can be further subdivided, as we will explain in the remainder of this section.

4.1.1 Part of the curriculum or materials. We firstly see reasons why teachers teach multiple languages which are not directly related to specific programming languages or teaching practices. Seven teachers named such reasons, for example, a set curriculum as the main reason to teach different languages or the fact that materials were available for specific languages.

4.1.2 Start with simple language. The biggest category that we found among the 23 interviewed teachers, named thirteen times, is that teachers want to start with a simple language. Often, but not always, this simple language is a block-based language. Teachers explained that at one point, simpler languages do not suffice anymore because children outgrow them, or they do not offer enough possibilities to teach with. At that point, there is the need for a second or third language. We distinguish four subcategories in the overarching category of "Starting with a simple language".

Category	Number of teachers
Part of the curriculum	7
Start with a simple language	13
- Simple language provides engagement	8
- Simple language avoids errors	5
- Simple language builds confidence	4
- Simple language supports focus on concepts	3
Different languages have different benefits	10
- Different languages have different applications	4
- Knowing different languages gives a different perspective	6

Table 2: Reasons teachers provide for teaching multiple languages

Simple language provides engagement. The first reason we observe named by eight teachers is that a simple language helps students to quickly get engaged in programming. For example, T1 states that *"We start with a simple language so we don't scare them"*. T8 adds that *"[Scratch] is easier and [uses] blocks and students can throw in something really quickly."*

Simple language builds confidence. Five teachers name the fact that a simple language build self-confidence in students, to increase the chances that they will enjoy programming. For example, teacher T16 states that they *"start with blocks to avoid frustration"*.

This seems to be related to the fact that, as we know from prior research as described in Section 2, syntax can be a large barrier for students. Starting with a simple, often block-based, language helps to push the syntax barrier to later in the course when students might feel more prepared for programming.

Simple language avoids errors. Five teachers indicate that they specifically start with a simple language because a simple language makes it easier for students to avoid syntactic mistakes. For example, teacher T9 states *"We start with Scratch to ... eliminate Syntax errors (S1-S2)."*

Teaching multiple language is seen as an opportunity to teach students in thinking about programming languages and their differences. For example, T5 explains that *"we explain that with Scratch you cannot make a website, that makes the student want to move to a new language."* Avoiding errors can be related to our previous category of building confidence.

Simple language supports focus on concepts. The final reason that three teachers specifically choose a simple language is because they believe that a simpler language makes it easier for students to focus on programming concepts rather than on syntactic issues. For example, T17 states *"We use Executable flowcharts, because they help to explain the three basic concepts: sequence, selection, iteration. We then "dress it up" with a textual programming language."*

4.1.3 Different languages have different benefits. The third main category is that the use of different languages has different educational benefits for students. Ten teachers name this as a reason for using multiple languages. For example, T20 states *"I specifically use different languages for different goals. First HTML/CSS to show them code is everywhere"*. We divide this main category further in to two subcategories.

Different languages have different applications. One of the reasons that it can be beneficial to teach multiple languages, named by four teachers, is to show students that different programming languages have different applications. If you want students to see how broad programming is, that can be done with different languages. T6 explains *"Different programming languages help to teach different parts of programming. For example C++ is difficult, but helps to teach hardware."*

Knowing different languages gives a different perspective. Finally, six teachers mentioned the fact that students' knowledge of multiple languages will give them a better perspective on programming languages. For example, T21 hopes that *"students will learn a new language sooner if they have seen two [languages] already."*

4.2 RQ2: Problems/benefits of teaching multiple languages

Teachers were asked to reflect on benefits and problems they observe in students when they transfer to another programming language. We categorized their answers into a list of benefits and another list of problems. Table 3 provides a summary of both the benefits and problems categories.

4.2.1 Benefits observed when moving to another programming language. We categorize the benefits into two main categories: Positive effect on understanding programming concepts in PL2 (15 teachers), and Positive effects on cognitive abilities (7 teachers).

Positive effect on understanding programming concepts in PL2. This is the biggest category under the benefits, highlighted by 15 different teachers. We distinguish here three subcategories. First, students transfer the understanding of basic programming concepts from one language to another. For example T4 states that students *"understand how they can use if-statements in VB because they have learnt that in Scratch, it happens quite naturally"*. Similarly, T9 highlights that students *"automatically transfer the concept of the if-statements and repeat statements from Scratch"*. Second, the understanding of concepts benefits from students making an explicit connection to the previous programming language to help to solve problems in the new language. This connection could come from only remembering the concept from the first language or due to teachers repeating the concept again. For example T4 highlights that *"Sometimes students themselves say "you would do this in Python like this or that, but in C# I do it differently now""*. According to

Benefit Category	Number of teachers
Positive effect on understanding programming concepts in PL2	15
- Transfer the understanding of basic programming concepts from one language to another	11
- Students make the connection/link to the previous language	7
- Happens only under specific conditions	2
Positive effect on cognitive abilities	7
- Transfer of problem solving skills and abstract thinking	4
- Broadening students perspectives	3
Problem Category	
Negative effect of syntax transfer	8
- Context switching is difficult and takes time	5
- Mixing syntactic elements from two languages	5
Negative effect of semantic transfer	10
- Misconceptions: Students making assumptions on semantics	6
- Struggle with semantics	6
Students make no connection to/with previous language	4
Students demotivated or less confident to learn a new language	3
- Students making generalizations on lack of ability based on a bad previous experience	3
- Stuck in one language that they know	1

Table 3: Benefits and problems highlighted by teachers of teaching multiple languages

some teachers, this connection could lead to students learning PL2 more quickly, as T6 puts it: *"concept repetition, [and then] you have the basis to pick up another language quickly"*. Third, some teachers stress that for any benefit on understanding to occur, specific conditions should apply, such as *"good instructions, peer discussion among the pupils and focus on code comprehension questions"*, as highlighted by T15.

Positive effect on cognitive abilities. Teachers recognize that multiple languages benefit developing cognitive abilities linked to programming. Under this category we recognize two subcategories. First, teachers report that students transfer problem solving skills and higher level thinking from PL1 to PL2. T7 explains that *"students learn that when they have a problem they need to split it to parts and then they can solve those small parts one by one. By doing multiple programming languages students would see that the way of thinking remains the same"*. T19 adds that learning multiple languages *"helps more in the thinking process, higher level thinking over what programming languages do"*. Second, some teachers stress that learning multiple languages broadens students' perspectives on programming languages and their applications: *"Knowing multiple languages allows the student to do different things"* according to T20, and students then know that *"a specific programming language is not sacred"* according to T18.

4.2.2 Problems observed when moving to another programming language. Teachers on the other hand identify problems when students move from PL1 to PL2. We categorize these problems into four main categories: Negative effect of syntax transfer (8 teachers), Negative effect of semantic transfer (10 teachers), Students make no connection to/with previous language (4 teachers), and Students demotivated/less confident to learn a new language (3 teachers)

Negative effect of syntax transfer. Teachers identify syntax as a major issue when transferring to a new programming language. We have under this category two subcategories. One subcategory relates to students' context switching from one language to another is difficult and takes time, taking into account that syntax differences between languages *"are sometimes big"*. T1 believes that students *"struggle with syntax more than concepts"* while T2 says that a new language *"throws a barrier with syntax"*. The second subcategory focuses on the problems that rise from students mixing syntactic elements from two languages. For example, T21 mentions that " *students write php sometimes in the python style: colon instead of, or mixed with, parenthesis for loops"*.

Negative effect of semantic transfer. This relates to the students understanding of the semantics of particular symbols or constructs in a new programming language. We recognize again two subcategories. First, students hold misconceptions based on previous knowledge. This could be knowledge from another domain, such as math or knowledge acquired from a previous programming language. For example, T1 says that *"the = symbol is a problem, they transfer the math equal"*, and T17 *"for example with Structurizr you had to put an* and *instead of a + in Python to concatenate strings"*. The second subcategory relates to a general struggle with semantics. A big part of this struggle stems from moving between strictly and less strictly typed languages. T10 highlights an example: *"When students switch to Java from Python, parameter passing is a challenge because Python does not explicitly ask you to declare your data types while Java does"*.

Students make no connection to previous language. This main problem category points out the case of transfer being unsuccessful as students fail to make the connection between the same concept in different languages. We note that the four occurrences of this category are when students transfer from a block-based programming environment, Scratch, to another programming language. For example, T8 describes the issue: *"I think they have transfer problems*

to Scratch, I think it's because they are given a block in Scratch and they don't think what's inside the blocks so they just pull them in, they don't notice it's a repeat block which is the same as a for-loop in Python"

Students demotivated or less confident to learn a new language. The motivation to learn a new programming language could be affected according to three teachers. This happens as students make generalizations on their lack of programming ability based on a first *"bad"* experience with programming. For example T23 thinks that *"when pupils go onto a second language but are not confident in the concepts and structures that appear in both, it can make them less confident"*. The second subcategory relates to students who remain stuck in one programming language. According to T19, this happens either because they they master one language and see no value in learning another one, or because they look down on other languages.

4.3 RQ3: Views on the use of transfer strategies

When exploring the views of computing teachers on the use of transfer strategies, we found that there are three main teachers' views. Table 4 provides a summary of these views.

4.3.1 Do not believe they have to implement transfer strategies. The results reveal that a considerable number of teachers (12) do not believe they have to implement transfer strategies in the classroom when teaching second or subsequent programming languages. The rationale for this belief is divided in five subcategories.

Implementing transfer strategies takes time. Four teachers expressed that it takes a lot of time to implement transfer strategies given that they are restricted by the class time. For example, T10 said, *"It takes an awful lot of time to teach for transfer and use it as an opportunity for learning. It takes time to unpack the concepts students are learning but there is no time"*.

Implementing transfer strategies confuses students. Two teachers expressed that transfer strategies might not always yield positive results. These teachers believe that implementing transfer strategies in the classroom will confuse students because it might become complicated. For example, T2 states, *"If I introduce the deeper semantics of a language, it's too early, I loose too many, we don't want it to get overly complicated"*.

Implementing transfer strategies puts students off . Two teachers also believed that transfer strategies might yield negative results such as demotivating students to learn a new language. For example, T8 stated that, *"we don't teach for transfer, we don't go that far. I am always worried about putting them off"*.

Mapping prior language to new language is difficult. Two teachers expressed that they find it challenging to map a block based environment to a text-based environment. For example, T1 states, *"Its difficult to convert a game environment from Scratch to Python"*.

Assume students transfer implicitly therefore no need for transfer strategies. The last reason teachers (2) don't implement transfer strategies is because they believe students automatically transfer without the need for help. For example, T11 states, *"I believe transfer happens quite naturally for students from Scratch to VB"*.

4.3.2 Believe transfer strategies are important and knowledge of more than one language helps foster transfer. Nine teachers believe that transfer strategies are important and when students know more than one language, it gives them the ability to transfer conceptual knowledge from one language to the other. For example, T19 states that, *"There are analogies between languages, but students do not see that for themselves, someone needs to tell them. With more concepts and analogies, students can learn a new language faster. Students become more resilient to new materials/products because they have seen something similar, helps more in the thinking process, higher level thinking over what programming languages do"*.

4.3.3 Don't think students have strong comprehension of the first language for any benefits of conceptual knowledge to carry over to a new language. Four teachers highlighted that students do not have strong understanding of the first language which then results in students having almost nothing to transfer to the new language. These teachers believe that a lot of teachers do not focus enough on the development of conceptual knowledge when teaching first programming languages. For example, T23 said, *"I think the biggest risk is not teaching the first language in a comprehensive way, especially Scratch, when it is sometimes seen as an "introduction" rather than a learning tool"*. A similar view is shared by T3 who states that, *"We then don't teach the constructs of the first language, but rather the game environment"*.

4.4 RQ4: Types of transfer strategies

When looking for the specific strategies teachers use to teach for transfer, we noticed that among the 23 interviewed teachers only nine intentionally use transfer strategies and one that did it unintentionally. The most notable transfer strategies were grouped in three different categories and are presented in Table 5.

4.4.1 Referring to the first programming language. The first main category of strategies is referring to the first programming language when teaching the second one. This category contains the three following strategies:

Explicitly referring to the first programming language. Six teachers used references to the first programming language. This strategy aims at bridging the gap between a concept used in the first language and its reuse in the second one. For example, T20 says that, when teaching a new language he would *"just repeat the concept briefly, but not too deeply in the new language. Indicate what differences and similarities are with the previous language"*.

Comparing code in different programming languages. Five teachers reported illustrating a concept by showing two versions of the same program, one in the first programming language and one in the new language. Sometimes, teachers even show both versions at the same time or, going even further, explicitly detailing the translation process from a representation to another (block to text, text to text, flowcharts to text). For example T13 says that: *"For most concepts I introduce through the course I map them to the Scratch ones to show how they work in both languages"*. T15 said he would *"for example show a JavaScript program on the whiteboard and together with students transform it to PHP"*.

Category	Number of teachers
Do not believe they have to implement transfer strategies	12
-Implementing transfer strategies confuses students	2
-There is not enough time to implement transfer strategies	4
-Mapping prior language to new language is difficult	2
-Assume students transfer implicitly	2
-Implementing transfer strategies puts students off	2
Believe that knowledge of more than one language helps in transfer	9
Do not think students have strong comprehension of the first language for any benefits to carry over to a new language	4

Table 4: Views on transfer strategies

Category	Number of teachers
Referring to PL1	
- Explicitly referring to PL1	6
- Comparing code in different programming languages	5
- Discussing similarities	4
Favoring transition	
- Putting the accent on concepts	8
- Using visual representations	4
- Using dedicated activities	3
- Avoiding language specific constructs	1
Preparing for transfer	
- Referring to everyday life	3
- Reinforcing first PL knowledge	1

Table 5: Types of transfer strategies

Discussing similarities. Four teachers mentioned especially pointing out similarities and differences between two different versions of a resolution in two different programming languages. By asking questions to student as *"How would you do it in C#/Unity?"*, for example.

4.4.2 Favoring transition. This category consists of four instructional strategies favoring the transition from a first programming language to a second one.

Putting the accent on concepts. Eight teachers mentioned putting the accent on concepts as a strategy for transfer. For example naming and abstracting concepts when seeing them for the first time in the first programming language. This allowed reminding previously seen concepts in the second language. Sometimes referring explicitly to the first language sometimes just mentioning the concept. For example, T22 says: *"We teach concepts that we have taught before, but we flag what they looked like in the first language when we introduce them in the second language."*

Using visual representations. Four teachers use an intermediate representation such as flowcharts to abstract the concept and make it more generic before reusing it in a second programming language. For example, T19 said: *"I teach them the logic flow in flowcharts first and when I move to Python I keep showing the flowcharts in one charts."*

Using dedicated activities. Three teachers reported favoring the transition between programming languages by using pedagogical strategies or specific tools. One strategy mentioned was allowing students to solve an assignment in the language of their choice. To reduce the load inherent to the problem, T21 gives to students the same assignment in the second language as in the first so that they can focus only on the new syntax. We regrouped in this category the use of specific tools such as developing environments that allow the students to switch from blocks to text and back, like T11 do. A last activity worth mentioning is T15 proposing assignments where students have to mix two different languages, forcing them to pay attention to the differences in syntax.

Avoiding language specific constructs. Finally, T20 says he *"tries to stay away from very JavaScript-specific concepts"* when another more generic syntax was available so that the *"switch is easier"*.

4.4.3 Preparing for transfer. One last smaller but interesting category are efforts made by teachers to anticipate transfer. In particular, for the following example, we argue that they reinforce students engagement.

Referring to everyday life. Two teachers mentioned that they look at websites students know and look at the code there. Another, T13, compared objects properties in an object oriented class to the fields of an element in a software they used earlier in class.

Reinforcing first programming language knowledge. One teacher mentioned transfer can only occur if a strong understanding of the first programming language is built. Therefore, T23 said that *"concepts transfer if students have a strong comprehension of their first language and first language understanding is stronger if students had a chance to discuss, debate and interrogate concepts"*. Even though he is not recognising this as a strategy by himself.

5 DISCUSSION

5.1 Reflections on Research Questions

5.1.1 RQ1: Reasons for multiple languages. There is a wide variation in teachers' reasons for teaching multiple programming languages. Some feel they have to because it is part of the curriculum. Most of them tend to associate the reasons with increasing students' engagement and excitement in programming; they entice the students with the first simple language before laying the hard graft on them with the second. These reasons appear to be justifiable considering that prior work has stressed the difficulties that novices face in understanding syntax [23, 24, 48]. These kinds of teachers do not believe that the first language should even teach concepts, hence they focus on creating games as part of their classes.

The deficit views are rather an issue for concern; not that they are necessarily wrong, but most teachers in our study did not mention being motivated to teach multiple languages because they can help students develop conceptual understanding. Some prior work has proposed that if students learn second/subsequent programming languages appropriately, their conceptual knowledge develops and becomes restructured [15]. Furthermore, the Computing curricula in earlier years had courses such as *Comparative Programming Languages* which were popular with common aims of deepening students' conceptual understanding [49].

5.1.2 RQ2: Problems/benefits of multiple languages. Teachers revealed that their students experience transfer problems. Most of these issues are captured by the prior programming language transfer research. Without explicit instruction, students can experience negative effects of semantic transfer from prior languages when learning constructs in a new language [15]. The other issues reported are when students fail to connect common concepts between the two languages, which usually occurs when they learn abstract concepts in a new language [14, 31, 32]. Other issues such as syntax problems have been reported in prior work, which investigated near novices problem-solving in new languages [3, 8, 11].

Despite these reported failures of transfer, teachers also reported some successes of conceptual transfer. These usually occur when students learn constructs that share similar syntax and semantics in the previous language and the new language [15]. Other positive issues such as transfer of problem-solving skills have been reported in [3–5], and they are also attributed to language similarities.

These results show that the same issues arise in schools as have been reported at universities, and this gives us a clear purpose in attempting to roll out transfer intervention strategies in schools. These interventions can have a much higher chance of success if done well; as school systems share curricula and teachers have a culture of adopting materials rather than developing them from scratch.

5.1.3 RQ3: Views on the use of transfer strategies. The beliefs expressed by teachers in answering RQ1 have significantly shaped their views and approach in teaching second languages. For instance, a majority of the teachers in our study do not believe that implementing transfer strategies in the classroom is a priority in teaching a second language. They report their own perceived ideas of the challenges that might arise when implementing transfer strategies. Their views mostly imply that they do not have much faith in the conceptual knowledge that the students have acquired in the first language. MPLT [15] has, however, shown that in most cases, students automatically effect semantic and conceptual transfer from prior languages based on syntax similarities. If teachers are not aware of these transfer mechanisms, it can impact learning of the second language negatively, as shown in RQ2 results and as confirmed in prior work [10, 13]. The 39% of the teachers, however, do believe in the importance of transfer strategies and appreciate that students' knowledge of multiple programming languages can help foster conceptual knowledge transfer and deepen conceptual understanding. As a result, these are the teachers who try to implement some form of transfer strategies in the classroom (RQ4).

5.1.4 RQ4: Types of transfer strategies. The transfer strategies reported in the study mostly include referring back to the first programming language. Strategies that put emphasis on conceptual transfer by showing different versions of the same code in the prior language and new language, or giving the same assignment in two different languages are in line with Hugging and Bridging techniques [44]. Other strategies use the idea of showing a concept in a different context [36]. These explicit transfer strategies have been used in prior work and have been reported to be successful in helping students transfer to new languages [10–12, 45, 46]. One teacher also explicitly mentioned paying attention to a strong understanding of the first programming language which aligns with reinforcing initial learning [35]. Although these are commendable efforts, only 39% of the teachers use transfer strategies and some of these transfer strategies seem to be rushed and informally implemented. For instance, teachers mention words like: *"I repeat the concept briefly, but not too deeply"* and *"I flag concepts"*.

5.2 Implications for CS Education

The findings emerging from this research indicate that a good number of teachers' transfer strategies reflected a range of good approaches widely promoted in the transfer literature [13, 44]. However, a majority of teachers did not believe they have to implement transfer strategies in the classroom. Based on these results, we give specific recommendations to teachers, researchers, and curriculum designers on areas where more should be done to support teachers in adopting transfer strategies in the classroom.

5.2.1 Recommendations for teachers. We recommend that teachers should consider their students' prior programming language knowledge and design classroom activities that engage their initial

understanding to help them evolve their conceptual knowledge as they learn new programming languages. Furthermore, teachers should use explicit instruction to help their students correct negative syntax and semantic transfer from prior languages [13]. Lastly, teachers should be strongly encouraged to add to their reasons for teaching multiple languages the benefit of deepening conceptual understanding about programming language concepts. By doing so, it will support the development of much stronger computational thinking skills in their students.

5.2.2 Recommendations for researchers. If transfer strategies are to be used effectively by computing teachers, a carefully planned pedagogy is required. Therefore this presents an opportunity for researchers to work with experienced teachers to develop, implement and evaluate a pedagogy where transfer interventions tackle specific student transfer problems (e.g. concepts, syntax, and semantics) presented in this study and transfer research [3, 11, 15, 50]. The analysis of insights and views collected from researchers is an important step toward improving second and subsequent programming language teaching. In addition to existing work on transfer interventions [10, 12, 13], further research is also required to understand how teaching for transfer can deepen conceptual knowledge and improve learning to program.

5.2.3 Recommendations for curriculum designers. CS curricular designers should draw on research and use it as guidance to design teaching material that aims to promote transfer as students transition between programming languages. We believe that the aims of any curricula should be to facilitate transfer and learning progression and development of conceptual understanding. Furthermore, teachers should be allowed to participate in a professional development program on transfer that covers the value of teaching multiple PLs to develop conceptual knowledge, benefits and challenges of second language learning, and teaching techniques for transfer.

5.3 Threats to validity

The goal of this paper is to understand the views of teachers on transfer and transfer strategies. This study suffers from a number of threats to validity. Firstly, we only interviewed teachers in two European countries, making it harder to generalize our findings, since the situation in other countries could be different. Secondly, we only concentrated on the situation in high schools, while many schools nowadays also teach programming at younger ages. As such, specific transfer issues experienced by children coming into high school with programming experience are outside of the scope of this paper.

6 CONCLUDING REMARKS

This study aims to understand the views and strategies high-school teachers employ when teaching a second or subsequent programming language. To that end, we have conducted interviews with 23 in-service teachers. Our most important findings include the fact that many teachers initially use simple programming languages, hence creating an opportunity for transfer learning (RQ1). Furthermore, teachers recognize both benefits and problems from transfer from PL1 to PL2. The benefits reported relating mainly to the role that PL1 can play in preparing for PL2. The problems reported by the teachers include; syntax problems, semantic difficulties (especially holding misconceptions), and the fact that students sometimes fail to see the connection between the two languages (RQ2). We observe that teachers mostly don't see the need to implement transfer strategies and don't capitalize upon the opportunity transfer could bring (RQ3). Finally, we see that some teachers do use well-known transfer strategies (RQ4). That indicates that future work could build on that willingness and spread the use of efficient transfer strategies.

Our paper opens up several other avenues for further research. For example, initial studies have shown that learning outcomes could be improved if teaching methods were adjusted in line with the predictions of MPLT [13, 14]. However, these interventions have not been explored in the context of high schools, and have not been used to study the transfer of block-based to text-based languages. In-depth experiments like [15] in these contexts would be a valuable next step. Researchers can carry out future work on transfer with teachers who already consider the importance of conceptual development and who already use transfer strategies. If this leads to better outcomes, the findings can underpin wider teacher professional development programs on transfer.

REFERENCES

[1] K–12 computer science framework, 2016.

[2] Neil CC Brown, Sue Sentance, Tom Crick, and Simon Humphreys. Restart: The resurgence of computer science in uk schools. *ACM Transactions on Computing Education (TOCE)*, 14(2):1–22, 2014.

[3] Jean Scholtz and Susan Wiedenbeck. Learning second and subsequent programming languages: A problem of transfer. *International Journal of Human-Computer Interaction*, 2(1):51–72, 1990.

[4] Jean Scholtz and Susan Wiedenbeck. Learning a new programming language: a model of the planning process. In *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences*, volume 2, pages 3–12. IEEE, 1991.

[5] Jean Scholtz and Susan Wiedenbeck. Using unfamiliar programming languages: the effects on expertise. *Interacting with Computers*, 5(1):13–30, 1993.

[6] Jean Scholtz. Adaptation of programming plans in transfer between programming languages: a developmental approach. In *Empirical studies of programmers: Sixth workshop*, page 233. Intellect Books, 1996.

[7] Quanfeng Wu and John R Anderson. Problem-solving transfer among programming languages. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ARTIFICIAL INTELLIGENCE AND PSYCHOLOGY ..., 1990.

[8] Michal Armoni, Orni Meerbaum-Salant, and Mordechai Ben-Ari. From scratch to "real" programming. *ACM Transactions on Computing Education (TOCE)*, 14(4):1–15, 2015.

[9] Divna Krpan, Saša Mladenović, and Goran Zaharija. Mediated transfer from visual to high-level programming language. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 800–805. IEEE, 2017.

[10] Wanda Dann, Dennis Cosgrove, Don Slater, Dave Culyba, and Steve Cooper. Mediated transfer: Alice 3 to java. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 141–146, 2012.

[11] David Weintrop and Uri Wilensky. Transitioning from introductory block-based and text-based environments to professional programming languages in high school computer science classrooms. *Computers & Education*, 142:103646, 2019.

[12] Michael Kölling, Neil CC Brown, and Amjad Altadmri. Frame-based editing: Easing the transition from blocks to text-based programming. In *Proceedings of the Workshop in Primary and Secondary Computing Education*, pages 29–38. ACM, 2015.

[13] Ethel Tshukudu and Siri Annethe Moe Jensen. The role of explicit instruction on students learning their second programming language. In *United Kingdom & Ireland Computing Education Research conference.*, pages 10–16, 2020.

[14] Ethel Tshukudu and Quintin Cutts. Semantic transfer in programming languages: Exploratory study of relative novices. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, pages 307–313, 2020.

[15] Ethel Tshukudu and Quintin Cutts. Understanding conceptual transfer for students learning new programming languages. In *Proceedings of the 2020 ACM Conference on International Computing Education Research*, pages 227–237, 2020.

[16] Igor Moreno Santos, Matthias Hauswirth, and Nathaniel Nystrom. Experiences in bridging from functional to object-oriented programming. In *Proceedings of the 2019 ACM SIGPLAN Symposium on SPLASH-E*, pages 36–40, 2019.

[17] Karen P Walker and Stephen R Schach. Obstacles to learning a second programming language: An empirical study. *Computer Science Education*, 7(1):1–20, 1996.

[18] Stephen Davies, Jennifer A Polack-Wahl, and Karen Anewalt. A snapshot of current practices in teaching the introductory programming sequence. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 625–630, 2011.

[19] Ursula Wolz, Henry H Leitner, David J Malan, and John Maloney. Starting with scratch in cs 1. In *Proceedings of the 40th ACM technical symposium on Computer science education*, pages 2–3, 2009.

[20] David J Malan and Henry H Leitner. Scratch for budding computer scientists. *ACM Sigcse Bulletin*, 39(1):223–227, 2007.

[21] Cindy Marling and David Juedes. Cs0 for computer science majors at ohio university. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 138–143, 2016.

[22] I. T. Chan Mow. Issues and Difficulties in Teaching Novice Computer Programming. In *Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education*, 2008.

[23] Paul Denny, Andrew Luxton-Reilly, Ewan Tempero, and Jacob Hendrickx. Understanding the syntax barrier for novices. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, pages 208–212. ACM, June 2011.

[24] Amjad Altadmri and Neil Brown. 37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data. *SIGCSE 2015 - Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 522–527, 2015.

[25] Andreas Stefik and Susanna Siebert. An Empirical Investigation into Programming Language Syntax. *Trans. Comput. Educ.*, 13(4):19:1–19:40, November 2013.

[26] Deborah J. Armstrong and Bill C. Hardgrave. Understanding mindshift learning: The transition to object-oriented development. *MIS Quarterly*, 31(3):453–474, 2007.

[27] Nan Jiang. Lexical representation and development in a second language. *Applied Linguistics*, 21, 03 2000.

[28] Michael Homer and James Noble. Combining Tiled and Textual Views of Code. In *Proceedings of the Second IEEE Working Conference on Software Visualization*, pages 1–10. IEEE Computer Society, 2014.

[29] David Weintrop and Uri Wilensky. How block-based, text-based, and hybrid block/text modalities shape novice programming practices. *International Journal of Child-Computer Interaction*, 17:83–92, 2018.

[30] Martinha Piteira and Carlos Costa. Learning Computer Programming: Study of Difficulties in Learning Programming. In *Proceedings of the 2013 International Conference on Information Systems and Design of Communication*, ISDOC '13, pages 75–80, New York, NY, USA, 2013. Association for Computing Machinery. event-place: Lisboa, Portugal.

[31] Kris Powers, Stacey Ecott, and Leanne M Hirshfield. Through the looking glass: teaching cs0 with alice. In *Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 213–217, 2007.

[32] Dale Parsons and Patricia Haden. Programming osmosis: Knowledge transfer from imperative to visual programming environments. In *Procedings of The Twentieth Annual NACCQ Conference*, pages 209–215, 2007.

[33] Susan Wiedenbeck. An analysis of novice programmers learning a second language. In *Empirical studies of programmers: Fifth Workshop: Papers presented at the Fifth Workshop on empirical studies of programmers, December 3-5, 1993, Palo Alto, CA*, page 187. Intellect Books, 1993.

[34] E. L. Thorndike and Robert S. Woodworth. The influence of improvement in one mental function upon the efficiency of other functions.(I). *Psychological review*, 8(3):247, 1901.

[35] David Klahr and Sharon McCoy Carver. Cognitive objectives in a LOGO debugging curriculum: Instruction, learning, and transfer. *Cognitive Psychology*, 20(3):362–404, July 1988.

[36] Mary L. Gick and Keith J. Holyoak. Schema induction and analogical transfer. *Cognitive psychology*, 15(1):1–38, 1983.

[37] Katerine Bielaczyc, Peter L. Pirolli, and Ann L. Brown. Training in self-explanation and self-regulation strategies: Investigating the effects of knowledge acquisition activities on problem solving. *Cognition and instruction*, 13(2):221–252, 1995.

[38] John B. Biggs and Kevin F. Collis. *Evaluation the Quality of Learning: The SOLO Taxonomy (Structure of the Observed Learning Outcome)*. Academic Press, 1982.

[39] Linda Seiter. Using SOLO to Classify the Programming Responses of Primary Grade Students. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 540–545, Kansas City Missouri USA, February 2015. ACM.

[40] Cruz Izu, Amali Weerasinghe, and Cheryl Pope. 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*, pages 251–259, Melbourne VIC Australia, August 2016. ACM.

[41] Lauren E. Margulieux, Mark Guzdial, and Richard Catrambone. Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, ICER '12, pages 71–78, New York, NY, USA, September 2012. Association for Computing Machinery.

[42] Elizabeth Patitsas, Michelle Craig, and Steve Easterbrook. Comparing and contrasting different algorithms leads to increased student learning. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research - ICER '13*, page 145, San Diego, San California, USA, 2013. ACM Press.

[43] Olivier Goletti, Kim Mens, and Felienne Hermans. Tutors' Experiences in Using Explicit Strategies in a Problem-Based Learning Introductory Programming Course. In *Proceedings of the 2021 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '21)*, Virtual Event, Germany, June 2021. ACM Press. (to appear).

[44] David N Perkins, Gavriel Salomon, et al. Transfer of learning. *International encyclopedia of education*, 2:6452–6457, 1992.

[45] Nour Tabet, Huda Gedawy, Hanan Alshikhabobakr, and Saquib Razak. From alice to python. introducing text-based programming in middle schools. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, pages 124–129, 2016.

[46] Monika Mladenović, Žana Žanko, and Andrina Granić. Mediated transfer: From text to blocks and back. *International Journal of Child-Computer Interaction*, page 100279, 2021.

[47] Victoria Clarke, Virginia Braun, and Nikki Hayfield. Thematic analysis. *Qualitative psychology: A practical guide to research methods*, pages 222–248, 2015.

[48] Andreas Stefik and Susanna Siebert. An empirical investigation into programming language syntax. *ACM Transactions on Computing Education (TOCE)*, 13(4):1–40, 2013.

[49] KN King. The evolution of the programming languages course. In *Proceedings of the twenty-third SIGCSE technical symposium on Computer science education*, pages 213–219, 1992.

[50] David Perkins and Fay Martin. Fragile knowledge and neglected strategies in novice programmers. ir85-22. 1985.