Understanding Conceptual Transfer for Students Learning New Programming Languages

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

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

ABSTRACT

Prior research has shown that students face transition challenges between programming languages (PL) over the course of their education. We could not find research attempting to devise a model that describes the transition process and how students' learning of programming concepts is affected during the shift. In this paper, we propose a model to describe PL transfer for relative novices. In the model, during initial stages of learning a new language, students will engage in learning three categories of concepts, True Carryover Concepts, False Carryover Concepts, or Abstract True Carryover Concepts; during the transition, learners automatically effect a transfer of semantics between languages based on syntax matching. In order to find support for the model, we conducted two empirical studies. Study 1 investigated near-novice undergraduate students transitioning from procedural Python to object-oriented Java while Study 2 investigated near-novice postgraduate students doing a transfer from object-oriented Java to procedural Python. Results for both studies indicate that students had little or no difficulty with transitioning on TCC due to positive semantic transfer based on syntax similarities while they had the most difficulty transitioning on FCC due to negative semantic transfer. Students had little or no semantic transfer on ATCC due to differences in syntax between the languages. We suggest ways in which the model can inform pedagogy on how to ease the transition process.

CCS CONCEPTS

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

KEYWORDS

programming language; transfer; code comprehension; concepts; syntax; semantics; Java; Python

ACM Reference Format:
Ethel Tshukudu and Quintin Cutts. 2020. Understanding Conceptual Transfer for Students Learning New Programming Languages. In *Proceedings of the 2020 International Computing Education Research Conference (ICER '20), August 10–12, 2020, Virtual Event, New Zealand.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3372782.3406270

ICER '20, August 10–12, 2020, Virtual Event, New Zealand

ACM ISBN 978-1-4503-7092-9/20/08…$15.00
https://doi.org/10.1145/3372782.3406270

1 INTRODUCTION

As they progress with their education, Computer Science students are expected to program using the different sets of linguistic concepts found in different programming languages (PL). Many PLs share similar concepts with different syntactic representations such as variable-assignments, while-loops, and methods, and understanding of these concepts can be transferred when learning new PLs. Uncovering how students gradually deepen their understanding of these concepts as they switch languages is a necessary step in expediting PL learning [25]. Prior work on PL transfer is mainly concerned with plan transfer by experienced programmers solving problems in a new language. By comparison, little is known about how students transfer their understanding of conceptual/semantic knowledge as they learn new languages. It is this gap that the paper aims to address by proposing a model of PL transfer that describes conceptual transfer for relative novices. Such a model is deemed necessary in computer science education because it can contribute to a deeper understanding of the nature of transfer and serve as guidance as well as help educators and students better account for the success and failure of conceptual transfer in learning new programming languages.

The paper offers two key contributions. First, a model of PL transfer is proposed, based on existing literature on programming and natural language transfer. The purpose of the model is to demonstrate the:

- role of syntax similarity and subsequent semantic transfer as a bridge to learning new PLs
- understanding of the relationship between PLs in the mind of a novice programmer

Second, two studies demonstrating PL transfer are presented to provide supporting evidence for the model.

The paper is structured as follows: Section 2 contains the literature review, Section 3 presents the key theories driving the design of the model of PL transfer presented in Section 4. The studies validating the model are presented in Section 5 and the final section provides recommendations for instructors, and conclusions.

2 RELATED WORK

In this section we review literature in programming language transfer, we start by reviewing transfer to first PL followed by transfer to second and subsequent PLs.

2.1 Transfer to the First PL

Prior work has explored aspects of novice programmers transferring knowledge into the process of learning their first PL. For example, students' understanding of natural language may interfere with

their understandings of the meanings of programming language constructs when learning their first PL [4, 30, 32, 44]. Prior knowledge of mathematics can also interfere in learning programming concepts, for example, the concept of a variable is used in both mathematics and programming but has significantly different meanings in each [9]. These interferences could be because when learning the first PL, students are learning new concepts and they use their intuition to try and understand these concepts. Intuitive interaction relies on the *unconscious* transfer of previously acquired knowledge, and therefore is not apparent to the student during learning [3]. This transfer has been seen as an important source of error in computing education leading to misconceptions in understanding programming concepts as summarised in Qian and Lehman's paper [32]. Stefik et al. explore which words and symbols novices find intuitive in PL [44] . Their study found that variations in the first PL syntax matter to novice programmers such that they find some syntax more intuitive than others.

By comparison, however, there is very limited research on how students move on to learn a second and subsequent PL, when they already have established conceptual knowledge from their first programming language. Finding the answer to this question is particularly important because several studies reported transition challenges relative novices face as they switch between programming languages [1, 2, 5, 13, 16, 18, 24, 26, 28, 48, 50, 51].

2.2 Transfer to second and subsequent PL

Early work on learning second and subsequent programming languages was mainly focused on experiments which observed experienced programmers solving programming problems in a new language. Participants in these studies were reported to start solving problems by using familiar plans from the prior language. This transfer was negative when they had to implement the plans in a new language that had different constructs from the prior language [34, 35, 38], however was positive if the new language had similar constructs as the prior language[52].

For instance, Scholtz and Wiedenbeck [35] reported negative plan transfer on experienced programmers with C and Pascal knowledge planning in the new Icon language. They discovered that in the early stages of learning a new language experienced programmers use a top-down and depth-first approach to solve a problem[36] and change their plans repeatedly when they reach implementation as they become familiar with new language constructs, this resulted in delays in solving the problem. This fits with the observations of plan transfer made by their later studies [34, 38]. However, in investigating novice programmers, Scholtz et al. [37] reported that novice programmers tend to use a bottom-up approach in solving problems in a new language.

Wu and Anderson[52] reported positive plan transfer between LISP and PROLOG and between LISP and PASCAL. Participants were reported to use the familiar algorithms in the first language to cut down the planning in the second language which reduced the number of revisions needed to finalise the plan. Overall they concluded that this advantage in transfer was because of the commonalities that the three languages shared in some concepts such as recursion. Syntactic interference was reported to be minor.

2.3 Transfer to second and subsequent PL by Novice Programmers

One common and underlying limitation with the empirical work just reviewed is that it mainly focused on transfer of problem solving skills by experienced programmers. But learning of programming languages is more than problem solving and plans, it is also about understanding the underlying programming language constructs that are used to write programs and how they behave when they are executed. In addition, prior work gives little attention to how relative novices move on from their first or second language, often only with a limited knowledge of plans[31, 37]. When learning programming, novice programmers tend to focus more on the syntax of a programming language[42, 44] which is intimately tied to their understanding of programming language constructs[15]. Taking this into consideration, we are interested in investigating how a novice's understanding of programming concepts in their first PL transfers to learning new languages. This study is also inspired by researchers who call for studies that extend beyond first-year courses and aims to explain how students evolve their understanding of semantics over time given that they learn different sets of linguistic concepts during their education[15, 25].

We intend to investigate this conceptual transfer rigorously, but unlike prior work, we use the program comprehension approach which is being recognised as key to learning programming in recent years [39, 40].

This study addresses the need for PL transfer research in CS education by proposing a model of PL transfer that was designed on the basis of our preliminary transfer study [46] and borrows from programming language and natural language theories, as explained in the next section.

3 THEORETICAL BACKGROUND AND PRELIMINARY STUDY

This section discusses the theories that guided the design of PL transfer model to be presented in Section 4. The model draws on Program Comprehension models, second language acquisition models based on semantic transfer and cross-linguistic similarity models.

3.1 Program Comprehension

The program comprehension models we have found relevant to review for this study [31, 39, 42, 47] explain the cognitive processes involved in the programmer's reading and understanding of a program. These models have common cognitive elements: Knowledge structures and Assimilation processes.

3.1.1 Knowledge Structures: The programmer is viewed as having two categories of static programming knowledge which are *Text-structure knowledge* and *Plan knowledge*. These categories are presented with different names in different comprehension models.

Pennington explains text-structure knowledge as consisting of a number of control-flow constructs such as iteration, conditional and sequence statements [31]. In Shneiderman and Mayor's model [42], this knowledge is referred to as *syntactic knowledge*. Schulte describes this knowledge as the *Structure* dimension, which consists of the way the program text is structured and the resulting execution of the program [39].

The Plan knowledge basically consists of patterns of program instructions that go together to accomplish certain functions[31, 39].

3.1.2 Comprehension process: The process of program comprehension in the models draws from the natural languages text comprehension models by Kintsch [22, 23] which describes comprehension as a chunking process. This view is shown in Schulte's model [39] which describes comprehension process as bottom-up and yet chaotic and flexible. The comprehension process is described as usually starting with reading code at atoms(language elements), to blocks, to inferences about the relations between blocks to recognizing the overall structure.

These program comprehension models indicate how programmers understand program code. However they refer mostly to comprehension in the case of one language with little attention given to how programmers understand a program in a new language. In this study, the interest is on comprehension mechanisms in the case of learning a second language, given that the programmer has already acquired syntax, semantics and plan knowledge in their first language. Unlike prior work which focused on the programmer's plan knowledge transfer, we focus in depth on the programmer's text-surface knowledge and how it transfers to a new language. This knowledge consists of the way the program text is structured, and accordingly the resulting execution of the program[40].

The program comprehension models [31, 39, 40, 43] have drawn from natural languages text comprehension models but no study has looked at natural language transfer in order to understand programming language transfer, therefore, we make this our starting point.

3.2 Natural Language transfer theories

Jiang[19, 21] has developed a model of lexical development in second language acquisition. He describes the lexical representation as having four components consisting of the lemma with syntax and semantics components (e.g. parts of a sentence and their meaning), and the lexeme with morphological and formal components (e.g. how a word is formed and its context, spelling and pronunciation). In this study we will focus just on the lemma because syntax and semantics are applicable also to programming languages. Programming languages, just like natural languages use syntax, which is a set of rules that inform us how to combine words and symbols to create well formed sentences/programs.

One important aspect emphasised by Jiang is that the lexical representation components (in our case, only syntax and semantics) are highly connected such that activation of one component in the mind of a language learner results in automatic simultaneous activation of the other components. Jiang's model is based on a notion of *semantic transfer* and consists of three stages:

(1) Stage 1 (Lexical Association Stage): A lexical association between Language 2 (L2) word and its Language 1 (L1) translation happens when a L2 lexical entry (e.g. a word) is introduced to the learner.
(2) Stage 2 (Lemma Mediation Stage): The word is then associated with the L1 word which already exists in the learner's knowledge structure and, crucially, its *semantic* meaning is *transferred* from L1 to L2.
(3) Stage 3 (Full Integration Stage): Finally, continued exposure to contextualised word input in L2 helps to develop L2 semantic context in the knowledge structure such that the dependency on the meaning from the L1 is weakened.

Jiang's model has provided useful insights on how learners learn new languages based on their first languages. Empirical work has confirmed the validity of this model in the natural languages context [19–21] based on reaction time studies. Most of the components of this model such as syntax and semantics can be used to explain how new programming languages are learnt. However, this model does not address in depth how the Stage 1 of lexical association occurs.

One of the ways that lexical association may occur when learning new languages is through cross linguistic-similarities. Ringbom [33] proposes that comprehension of a new language can start with perceiving lexical similarities to elements of the language they already know which is followed by the assumption of the associated semantic or functional similarity.

They propose three similarity relations between languages:

(1) Similarity Relation: This means that an item or pattern in the target language(L2) is perceived as functionally similar to a form or pattern in the prior language (L1). For instance, cognate words like *mine* in *English* which means the same as *mein* in *German*.
(2) Contrast Relation: This means that an item or pattern in L2 is perceived in important ways as differing from L1 form, though there is an underlying similarity between them. For instance, deceptive cognates such as *lecturer* in English and *lektura* which means reading in Polish.
(3) Zero Relation: This means that an item or pattern in L2 appears to have little or no perceptible relation to L1 or any other language the learner knows. For instance, *talk* in English which is *rozmowa* in Polish.

From the theories of natural languages reported above, it seems clear that when learners learn their second language, semantic transfer will / is likely to occur. In some cases, semantic transfer can be influenced by cross-linguistic similarities.

3.3 Preliminary study

Prior to the work presented in this paper, a qualitative study was conducted that explored these aspects of code comprehension, semantic transfer and cross-linguistic similarity theories in the context of programming language transfer. This work represents a first step in developing an overall understanding of the possibility for semantic transfer based on language similarities to explain transfer between programming languages. The study explored semantic transfer in an experimental setting of students transitioning from their first year introductory to programming course in Python to second year object-oriented Java course. The findings indicate that during the initial learning stages, learners relied mostly on their syntactic matching between Python and Java and subsequent semantic transfer which affected their learning positively when the syntax and semantics between these languages were similar (e.g. *if-statements*) and negatively when the syntax between the languages was similar but had different semantics (e.g. *variable reassignment*

to different type). Students could not transfer their semantic knowledge on concepts they *perceived* as new in the second PL because they could not map the new syntax to Python syntax (e.g. *objects* and *dictionaries as data structures*. Details of this study can be seen in [46].

3.4 Discussion

We have reviewed above, code comprehension models indicating how programmers process understanding of program code. In the models, we learnt about the programmer's knowledge structures. We went on to examine the natural language transfer theories that view semantic transfer and cross-linguistic similarities playing a crucial role in learning new languages. A small scale qualitative exploration of these theories in our preliminary study validated the notion of semantic transfer based on syntactic similarities between programming languages. In order to quantitatively investigate this notion further, in the next section we propose a model suitable for programming language transfer based on our preliminary research findings.

4 MODEL OF PL TRANSFER FOR RELATIVE NOVICES

In this section we merge and refine existing models of code comprehension [31, 39, 42], semantic transfer [21], and cross-linguistic similarities [33] into a single developmental model of PL transfer, building on our earlier findings.

4.1 Knowledge Structures in the PL transfer model

We derive the programmer's knowledge structures from code comprehension models in related work, which propose that the programmer has knowledge about the structure (text-surface and program execution) of the program and the plans/function of the program. [31, 39]. Plan knowledge is not the focus of this study as explained in section 2. The PL transfer model proposes 3 levels of knowledge: concepts, semantics and syntax, with a clear separation of concepts and semantics which is not the case for the mentioned code comprehension models. Recognising the separation of conceptual knowledge and semantic knowledge means that we believe that when the programmer learns a second PL they might be faced with having to restructure their conceptual knowledge to have two representations of semantics that are specific to each language. For example, the programmer who knows both Python and Java has knowledge of the *for-loop* concept shared by both Python and Java language, however, this concept has an *iterator-based* semantics in Python while in Java the semantics is index-based. If semantics and concepts are seen as one thing in the knowledge of a programmer, how then can they represent this scenario?

In our proposed model, the programmer's knowledge is built of a network of nodes that are connected at different levels (syntax, semantics and concepts), following a connectionist approach of cognitive learning [6, 8], this is shown in Figure 1.

- **The Syntax Level**: The programmer has knowledge of the syntax of a known programming language. For example, *int mark =4;*.
- **The Semantic Level**: The semantics provide meaning to a well-formed syntax. This level is concerned with specific 'implementations' of the higher level concepts. For example, *variable **mark** is being declared as an integer and initialized to 4.*
- **The Concept Level**: This level contains the underlying concepts of programming languages known by the programmer. For example, *Variable Introduction.*

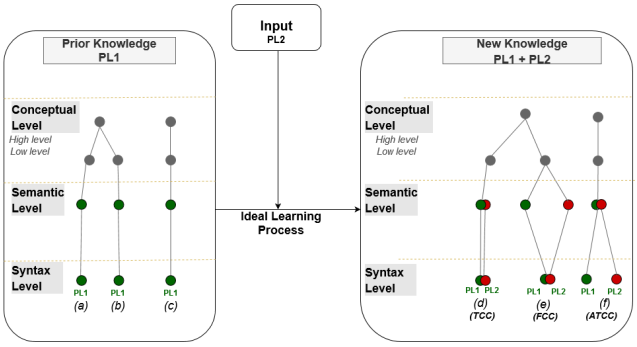

Figure 1: Model of PL transfer.

When a programmer is learning their first programming language (PL1), associations between the nodes are established, as shown in the left side of Figure 1. Each 'route' running from the Conceptual Level down to the Syntax Level represents a concept, its semantics or implementation in the first language, and its representation in the syntax of that language. At the prior knowledge level, where students know just one language, there is no branching across the semantic or syntax levels - because for a given concept, there is just one implementation and syntax known to the learner. At the Conceptual level, a tree structure may emerge, given that a concept may have sub-concepts - e.g. the concept of repetition has the sub-concepts of fixed loops and conditional loops (as captured in concepts (a) and (b) in the diagram). A concept presented as in (c) in the diagram could be, for example, a return instruction from a function - in a single language, there is usually only one form of this concept.

Once students learn the second programming language (PL2), more interesting knowledge structures are developed. Completely new concepts may be introduced, not shown in the diagram but would appear as a new 'tree' at the conceptual level. Concepts (d), (e) and (f) all represent important but different kinds of ideal relationships between the concepts found in the two languages in a programmer's mental representation. These concept relationships are represented by the following concept categories:

- *True carryover construct (TCC)*: is a construct with *similar* syntax and same underlying semantics in PL1 and PL2, this is represented by the (TCC) concept branch in Figure 1. For example, a *while loop* in Python and Java.
- *False carryover construct(FCC)*: is a construct with *similar* syntax but *different* semantics in PL1 and PL2, this is represented by the (FCC) concept branch in Figure 1. For example an *integer division* in Python 3 and Java.

- *Abstract true carryover construct(ATCC)*: is a construct with *different* syntax but same semantics in PL1 and PL2, this is represented by the (ATCC) concept branch in Figure 1. Examples are constructs whose implementation details are hidden such as *data abstraction (objects)* [49] in Java which at a low level are data structures like Python *dictionaries*.

It should be noted that at this stage there is no clear/objective measure of programming language similarity, we will refer back to this point later.

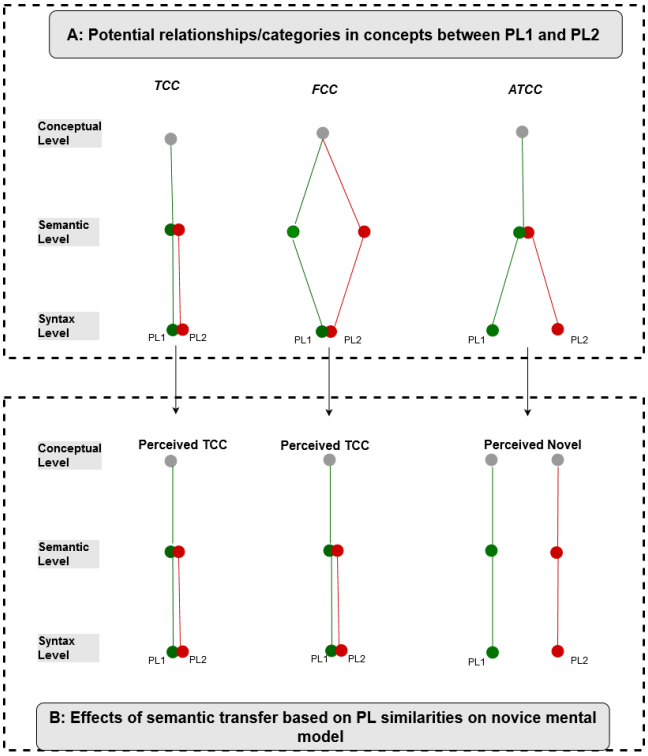

Figure 2: Constructs categories a learner encounters during learning process, and the consequences

4.2 Predictions of what actually happens during learning PL2

The learning process is derived from natural language theories explained in Section 2. Jiang's model of semantic transfer [19, 21] and Ringbom's cross-linguistic similarity relations [33] are merged and refined to suit the programming language transfer context. At this step of model building, we used an inductive approach[45] to refine theory based on the qualitative data from the preliminary study [46].

At the initial stages of learning PL2, programmers will engage in learning three categories of concepts, TCC, FCC and ATCC. These three categories are shown in *Box A* on Figure 2. The crucial point is that each of these categorised constructs can be *perceived* or *actual*. The *Box B* of Figure 2 predicts the learning process when students encounter these constructs during learning a new language.

(1) If a learner encounters a TCC, they will perceive it as a TCC because of syntax match, hence semantic transfer will be appropriately effected from PL1 to PL2 and the learning will be positively impacted. This is represented in *Box B*, in the TCC column of the diagram. See how, even with two languages, the diagram is a straight line from concept to matched semantics to matched syntax.
(2) If a learner encounters a FCC, they will perceive it as a TCC because of syntax match, hence semantic transfer will be inappropriately effected from PL1 to PL2 and the learning will be negatively impacted. This is represented in the central column of *Box B* in the FCC part of the diagram, which does not match the correct learning in *Box A*.
(3) If a learner encounters an ATCC, they will perceive it as novel since there is little or no syntax match, hence there will be little or no semantic transfer . This is represented in the right column of *Box B* in the ATCC part of the diagram, which does not match the correct learning in *Box A*.

5 RESEARCH STUDIES

In this section, two empirical studies are presented that represent an initial validation of the overall proposed model of PL transfer. The section also presents the research questions addressed, the methods involved, data analysis and interpretation.

Given the proposed learning process by the model of PL transfer outlined in the previous section, we hypothesise the following when learners learn PL2. Specifically, we expect differences in the understanding of concepts in PL2 (measured by the score of individual concepts in a given quiz) in the three construct categories outlined by the model (TCC, FCC, ATCC) such that *at the early stages of learning a PL2*:

- Hypothesis 1: There will be no significant difference in the score for concepts involving TCC between PL1 and PL2.
- Hypothesis 2: There will be a significant difference in the score for concepts involving FCC between PL1 and PL2 such that PL2 score will be less than PL1 score.
- Hypothesis 3: There will be significant difference in the score for concepts involving ATCC between PL1 and PL2 such that PL2 score will be less than PL1 score.

5.1 Study 1:

5.1.1 Study 1 Participants: The participants of this study were undergraduate students who were transitioning from procedural Python which they had completed in their first year to object oriented Java which they were now undertaking in their second year. They were encouraged to participate in the study by being given one mark towards their continuous assessment. Their data was only included in the study if they volunteered to participate by signing a consent form. In a normal classroom setting of introductory programming courses it is a challenge to find participants with only one programming language, our research focus. For example, we found some participants who self-reported having knowledge of multiple languages including some already having exposure to Java programming language. There were a total of 120 participants who agreed to their data being used for the study, of which 70 had no Java experience and 50 had Java experience. The data that will be presented will be for the 70 participants who had no Java

Category	Python	Java

TCC (While loop)	`sum = 0`
	`i = 0`
	`while i < 3:`
	`  sum=sum+i`
	`  i += 1`
		`int sum = 0;`
		`int i = 0;`
		`while (i < 3){`
		`  sum=sum+i;`
		`  i += 1;}`

FCC (Array equality)	`e = [1, 2, 3]`
	`f = [1, 2, 3]`
	`print(e==f)`
		`int[] e = {1, 2, 3};`
		`int[] f = {1, 2, 3};`
		`System.out.println(e==f);`

ATCC (object aliasing)	`n1={'name':'Joseph',`
	`'age': 51}`
	`n2={'name':'Vic',`
	`'age': 35}`
	`n1=n2`
	`n2['age']=n2['age']+1`
	`print(n1['age'])`
		`public class Robot {`
		`  String name;`
		`  int age;`
		`  public Robot(String n, int w){`
		`  this.name=n;`
		`  this.age=w; }`
		`  public static void main(String[]args){`
		`  Robot n1=new Robot("Joseph", 51);`
		`  Robot n2=new Robot("Vic", 35);`
		`  n1=n2;`
		`  n2.agga();`
		`  System.out.println(n1.age);}`
		`  public int agga(){`
		`  age=age+1;`
		`  return age;}}`

Figure 3: Example of concept categories used in the study

exposure. The participants reported an age range of 18-23 years. All the participants had an average of 1 year programming experience.

5.1.2 Study 1 Materials: The quiz material consisted of concepts derived from the first year Python programming course in conjunction with material they were learning for the second year Java programming course. This covered the following concepts: expressions, operators, control structures (indefinite and definite loops), functions, data structures and objects and classes.

Given that the quality of the quiz materials was critical for the purpose of this study, careful procedures were followed in the construction of the quiz content. The common concepts shared between the two languages were syntactically and semantically categorised according to the model's FCC, TCC and ATCC concepts by both the authors.

The initial step was to categorise based on the similarity of syntax between Java and Python for the same concept. The choice of stimulus words or characters in the syntax expected to influence the participants perception of similarity were chosen guided by the first qualitative study findings. In the preliminary study, participants based their similarity between programming languages on lexical tokens. This type of intuitiveness is also reported in Stefik's work [44]. Examples of the tokens that participants recognise are : keywords (*for, if-else, while*), literals (*int, Boolean, String*), operators (*+, -, /, **) and separators. Secondly, to determine whether Python syntax shares the same semantics as Java syntax for a given concept, code fragments were compiled/executed in each language respectively. Finally based on these steps, each concept was categorised as belonging to FCC, TCC or ATCC. Examples of the categorisations are in Figure 3, which demonstrates a while loop (TCC), equality (FCC) and aliasing (ATCC). The rest of the materials for the study are available at http://ccse.ac.uk/studymaterials.

It should be noted that the degree of perceived similarity can differ from learner to learner, based on several factors, such as, the degree of exposure to other languages, and the depth of knowledge of programming constructs.

5.1.3 Study 1 Procedure: The study used a within-participant analysis which meant that all the participants were exposed to the same paper-based quizzes within the same conditions: thus, each participant's performance on one quiz was compared with his or her own performance on the other quizzes. This type of experimental design was used in order to reduce the participants' variability that may have been attributed to differences between subjects. The changing of the questions and randomisation of the order for each quiz on the same construct helped counter the learning effect that can result from participants being exposed to similar tests more than once.

The quizzes were administered at the end of the third week of participants learning Java during normal teaching times in their Java class and participants were given 25 minutes to complete each quiz. Each quiz consisted of short answer questions in which the participants were asked to read and hand execute code and write down the output. This type of inquiry allowed us to determine the participants' knowledge more effectively, rather than choosing or guessing from a set of multiple choice answers. Figure 4 shows a sample Java question. This question was testing for knowledge of the concept of methods which includes method calling, parameter passing and scoping.

What will be the output of the following **Java program** fragments?

```
public class Main {
  public static int gen(int g, int s){
    int a=g+s;
    cen();
    return a;
  }
  public static void cen (){
    int a=2;
    System.out.println(a);
  }
  public static void main (String[] args){
    System.out.println(gen(4,3));
  }}
```

Figure 4: Sample questions given to the participants in a quiz

In order to investigate the process of programming language transfer, participants were given both the Python and Java quiz on same constructs, an example question from which is shown in Figure 4. The Python quiz was given in order to establish the baseline of participants' conceptual and semantic knowledge that they have already acquired from first year. The Java quiz was administered first (least known language) then the Python (most known language). This was to reduce the chances of stimulating transfer from Python to Java. A rest break of 10 minutes was given between the quizzes.

5.1.4 Study 1 Analysis. The participant responses were recorded in an Excel document and marks were allocated such that 0 was for incorrect answer and 1 for was for correct answer for all the construct categories, TCC,FCC and ATCC. A participant could get a total of 12 marks with each concept category (TCC, FCC and ATCC) containing 4 questions. In analysing the results using the R package, the mean score in each category was calculated. This helped us to analyse a more accurate within-participant score for conceptual understanding. The non-parametric Wilcoxon signed-rank test was thereafter used to compare significance in the score between the quizzes because the dependent variable (scores) were

not normally distributed. These tests are appropriate for measuring our within-participant score results. In order to determine whether the statistical tests in this paper are significant or not, we followed the usual consensus adopted by computer science education research [11]. The generally accepted p=.05 is regarded as the cut-off point, therefore, statistical significance that is accepted is when *p-value* is less than .05.

5.1.5 Study 1 Results. In order to examine Hypothesis 1, 2 and 3, a comparison is made between the TCC, FCC and ATCC scores of the Python quiz and the Java quiz. This was to investigate whether participants showed comparable levels of conceptual knowledge in the Python quiz and in the Java quiz. Furthermore we wanted to test if semantic transfer from the known Python language affects learning of Java in a positive or negative way as predicted by the model. Figure 5 shows the results of grouping participant mean scores of correct answers by concept category and language from the two quizzes(Java and Python) which were conducted in the third week of participants learning Java programming.

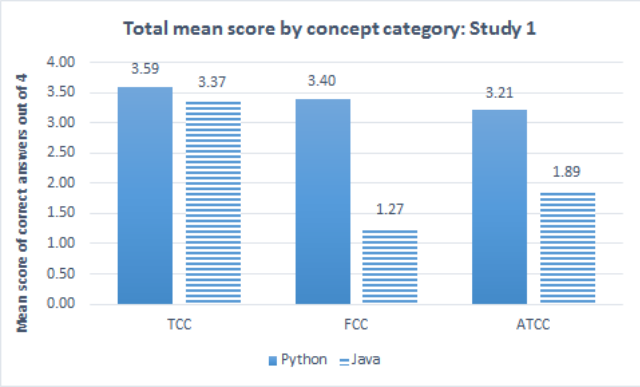

Figure 5: Participants mean score grouped by concept category and programming language: N=70

A Wilcoxon signed rank test using paired-samples indicated that there was a significant difference in performance as represented by the score data between the Python quiz and the Java quiz for the FCC (p <0.001), ATCC (p <0.001) and TCC (p= 0.018). In general, participants performed much better in Java in TCC (3.37, out of 4) than on FCC (1.27) and ATCC (1.89). Figure 5 shows that the mean score for FCC in Python was 3.40 while in Java it dropped significantly to 1.27. A similar result was seen in the ATCC which showed a large significant drop in mean score from Python (3.21) to Java (1.89). Each category comprised of questions on different constructs, as a result a Wilcoxon signed rank test was computed for each construct for further analysis as shown in Table 1.

There are interesting trends that are important to note in Table 1. The first, is that all the four individual TCC constructs are highly consistent and showed no significance difference between Python and Java quiz score, *string concatenation* (p= .424), *operator precedence* (p=.766), *while loop* (p=0.266) and *functions* (p=.069). In the case of FCC constructs, the scores of each of them reduced significantly in Java as compared to Python, *array equality* (p<.001), *string coercion* (p<.001), *string multiplication* (p<.001) and *integer division* (p=.024) . Most participants performed poorly in Java compared

Table 1: Mean scores of individual concepts tested in Study 1: N=70

Category	Construct	Python	Java	Wilcoxon P-Value
TCC	String concatenation	0.99	0.96	0.424
TCC	Operator precedence	0.86	0.84	0.766
TCC	While loop	0.91	0.86	0.266
TCC	Functions	0.84	0.71	0.069
FCC	Array equality	0.94	0.11	<0.001
FCC	String coercion	0.83	0.14	<0.001
FCC	String multiply	0.90	0.47	<0.001
FCC	Int division	0.73	0.54	0.024
ATCC	Object retrieval	1.00	0.59	<0.001
ATCC	Object update	0.63	0.59	0.298
ATCC	Object assignment	0.93	0.58	<0.001
ATCC	Object aliasing	0.63	0.11	<0.001

to Python on all these constructs. Thus, most participants are less accurate in their Java answers when Java and Python syntax look similar but their semantics are different. This implies that there is a negative semantic transfer from the participant' first language (Python) to the new language (Java).

On the ATCC constructs, there was a significant difference between the score of Python and Java constructs, *objects retrival* (p<.001), *object update* (p=.298), *objects assignment* (p<.001) and *objects aliasing* (p<.001). These seem to follow the same pattern as the FCC constructs. Thus, the results show that as syntax becomes more distant on similar concepts, the accuracy of answers decreases in the new language(Java).

5.1.6 Study 1 Discussion. The results of the study show important interactions in syntax and semantics of Python and Java as perceived by the programmer as well as how these perceptions determine which construct category gets affected when students are learning a new language.

First, it can be seen that participants got higher marks for the TCC category in both the Python and Java quiz. Specifically, as predicted in hypothesis 1, each TCC construct did not show a significant difference in mean score between Python and Java. We argue that it is because in the TCC category, Java shares similarities in both syntax and semantics with Python such that there is positive semantic transfer as shown in the assimilation process in Figure 2. This figure shows that the mental model of the student does not need to be restructured when they encounter a TCC, this is shown by no branching in the conceptual, semantic and syntactic level. Consequently, it can be said that, in the TCC category conceptual knowledge in the prior language (Python) can be a strong predictor of performance in a new language (Java).

Although the TCC showed insignificant differences between the two languages, functions/methods showed the most drop in the mean score from Python (.84) to Java (.71) than the other TCC constructs. This could be because the researchers had categorised a function as a TCC however not all students recognised it across the board to the same extent, this brings in the dimension of student variations when recognising similarities[33]. Some students get confused about Java methods because unlike Python they have declaration of parameter and method return types which might cause misconceptions as reported in other studies [17, 44].

As predicted in hypothesis 2, the FCC score between Python and Java was significant. Because these constructs looked similar to Python constructs participants perceived them as TCCs hence transferred their Python understanding of these constructs to Java. This means that relative novices mostly hold *one* semantic model of the FCC in two languages even if the semantics of that concept has changed, this is shown in Figure 2 in the centre. Our model proposes that at early stages of learning PL2, they struggle with holding *two* semantic representations of the same concept in two languages.

An example is the responses the participants gave for the output of the *equality (==) of two array*s shown in Figure 3 in the FCC category. Participants performed significantly better in the Python version (94% correct) versus the Java version (11% correct). By examining the incorrect responses given by students in Java, more than three-quarters of them (77%) gave an answer that corresponded to their Python answer.

Another example of inappropriate semantic transfer is the participants response of *string coercion* question in Java. Most of the participants (76%) responded incorrectly that this Java statement will produce a Type error: `System.out.println("Friday is no:" + 1 )`, this was the same response participants (83%) gave in the Python version: `print ("Friday is no:" + 1 )`, which was correct . Participants believed that the compiler cannot implicitly convert an integer value to a string in Java, which is the behaviour expected in Python. This provides compelling evidence for the finding that they transferred semantics of these constructs from Python to Java.

Lastly, as predicted in hypothesis 3, the ATCC mean score differences between Python and Java was significant for all the constructs. Participants performed better in each of these concepts in Python as compared to Java. This could be because of the distance in syntactic similarity between the two languages such that participants did not recognise they are already familiar with these concepts. Unlike TCC, they struggled to learn them simply by syntactic mappings between the two languages.

An example is when participants were struggling to transfer their knowledge of *object aliasing* which they knew in Python *dictionaries* to Java *objects* as shown in Table 1. Participants performed significantly better in the Python (63% correct) versus the Java (11% correct). It is evident that, even in their PL1(Python), 37% of the students still struggled with the concept of aliasing, this has been evidenced by other studies [12, 27]. However, what is important to note is that the participants who understood the concept in Python failed to transfer the knowledge to Java, most of the participants who got Java objects aliasing wrong gave the answer of *copy semantics* instead of *reference semantics*. It should be noted that both Python and Java use *reference semantics* in composite data structures[49]. In this example, both Java objects and Python dictionaries represent a data structure, the difference is that unlike dictionaries, Java objects implementation details are hidden[41]. These results are consistent with findings from Fisler at al. [12] when students failed to transfer their knowledge of aliasing from Scheme to Java.

In summary, the data from the quizzes of Python and Java have supported Hypothesis 1,2 and 3 of the model as explained above. The data has provided evidence that surface structure similarities of programming languages do affect relative novices when transitioning to new languages in the case of Python to Java.

5.2 Study 2

The previous study revealed some fundamental issues of semantic transfer on relative novices transferring from Python to Java and how the impacts of this transfer is positive for TCC and negative for FCC and ATCC. The second study builds on and gains deeper insights into the claims that this model makes. We take a different approach by investigating participants who have knowledge of Java and are at their early stages of transferring to Python.

5.2.1 Study 2 Participants: The participants of this study were postgraduate Masters degree students studying Information Technology who did not study computer science for their undergraduate degree. Most of them had their first exposure to programming in the Java course that was taught the previous semester. They were now enrolled for second semester in a web application development course in which they needed to pick up Python as well as web applications. They were recruited in their class and encouraged to participate in the experiment as a form of revision for their exam. A total of 50 participants agreed to participate in the study, of which 33 only had 1 prior Language (Java) before learning Python, 17 has knowledge of mixed languages including Python and Java (e.g C++, C, VB, lisp and swift). The data that will be presented will be for the 33 participants who only had Java exposure before learning Python. All the 33 had less than one year experience of Java programming language with an average of 6 months experience. Participants' ages ranged from 23 to 37 years.

5.2.2 Study 2 Materials: The design of the contents of the quiz followed the same steps as explained in Study 1. The quiz material consisted of concepts derived from the first semester Java programming course in conjunction with material they were learning for the second semester Internet Technology course using Python. More programming concepts that were not tested in Study 1 were included in this study such as *for-loops*, *if-conditional-statement* and *array-lists*.

5.2.3 Study 2 Procedure: In line with the procedure set out in Study 1, this study used a within-participant design and was carried out after one and a half months of participants learning Python (week 6). The test was allocated during a normal one hour class session. Participants were given paper-based handouts with questions similar to the one presented in Figure 4 for both the Python and Java test. The questions were code comprehension which covered shared constructs between the languages as shown in Table 2. Unlike in Study 1, the Python quiz (least known language) was given first and the Java (most known language) was administered thereafter. Participants were given 25 minutes to complete each quiz, however more time was allocated if needed.

5.2.4 Study 2 Analysis. The participant responses were recorded in Excel document and marks were allocated such that 0 was for incorrect answer and 1 was for correct answer for TCC, FCC and ATCC. A participant could get a total of 18 marks with each concept category (TCC, FCC and ATCC) containing 6 questions. Just like study 1, the non-parametric Wilcoxon signed-rank test was used

to compare significance in the score between the quizzes because the dependent variable (scores) was not normally distributed.

5.2.5 Study 2 Results. In order to examine Hypothesis 1, 2 and 3 a comparison is made between the TCC, FCC and ATCC scores of the Python quiz (new language) and the Java quiz(prior language). Similarly to the findings in study 1, participants generally performed much better in Python in the TCC (5.33, out of 6) than on FCC (3.12) and ATCC (3.84) as presented in Figure 6. In order to explore

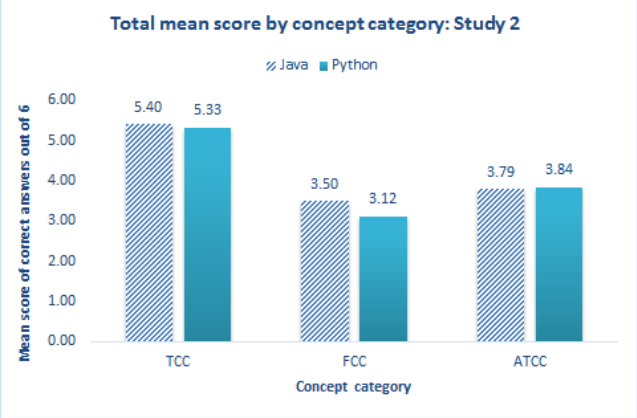

Figure 6: Study 2 participants mean score grouped by concept category and programming language: N=33

this further, a mean score of the individual constructs tested was calculated as presented in Table 2. All the six TCC scores showed no significant difference between the two tests as predicted in hypothesis 1, *string concatenation* (p= 1), *operator precedence* (p=1), *functions and parameters* (p=.233), *functions and scope* (p=.789), *while loop* (p=.223) and *If-conditional* (p=1).

In the case of FCC constructs, the mean scores of most of the constructs tested were significantly different in Java and Python *for loop* (p<.001), *string coercion* (p<.001), *string multiplication* (p=.008), *array equality* (p=.013) and *type checking* (p<.001). However the *int division* (p=.063) did not show significant difference. But unlike Study 1 and hypothesis 2 prediction, there were mixed results in the score of Java (PL1) and Python (PL2): for some instances as expected Java score was higher than Python score(e.g. *for-loop*) while for other instances unexpectedly Python score was higher than Java score (e.g. *array equality*).

As for ATCC constructs, subjects performed almost the same in Java and Python with no significant difference for all constructs except for *object aliasing* (p=.023) which showed a significant difference. Again like the FCC, it was a mixed result, with students showing more understanding in Python than Java (e.g. *list update* while in other instances they showed more understanding in Java (e.g. *object aliasing*). So the match was more easily made from Java to Python than from Python to Java.

5.2.6 Study 2 Discussions. In general, as in Study 1, across all three construct categories, students had higher scores in the TCC categories as compared to both ATCC and FCC. As predicted in hypothesis 1, each TCC construct score did not show any significant difference between Java and Python. Students were seen to be

Table 2: Mean scores of individual concepts tested in Study 2: N=33

Category	Construct	Java	Python	Wilcoxon P-Value
TCC	If Condition	1.00	0.99	1
TCC	String concatenation	1.00	0.97	1
TCC	Operator precedence	0.96	0.94	1
TCC	Functions parameters	0.88	0.79	0.233
TCC	Functions scope	0.79	0.76	0.789
TCC	While loop	0.76	0.88	0.223
FCC	For loop	0.91	0.15	<0.001
FCC	Int division	0.85	0.60	0.063
FCC	String coercion	0.67	0.09	<0.001
FCC	String multiply	0.39	0.69	0.008
FCC	Array equality	0.33	0.67	0.013
FCC	Type checking	0.33	0.90	<0.001
ATCC	Object retrieval	0.97	0.97	1
ATCC	Object update	0.97	0.97	1
ATCC	Object aliasing	0.53	0.27	0.023
ATCC	List retrival	0.58	0.73	0.145
ATCC	List update	0.56	0.73	0.145
ATCC	List aliasing	0.21	0.18	0.80

breezing through constructs such as *while loops*, *if-conditionals* and *functions* in either direction from Python to Java (Study 1) or from Java to Python (Study 2). This result corroborates the findings of other researchers[28] when they were observing programmers transitioning from procedural language to object-oriented language.

As predicted in hypothesis 2, the FCC individual mean score between Java and Python was significantly different for each construct. However, unlike in Study 1 where all the FCC scores were only lower for the Java language (PL2) as compared to Python (PL1), the semantic transfer was happening in both directions such that the score for FCC was negatively affected in both languages for different constructs. For example, inappropriate semantic transfer was observed in the participants response of *string coercion* question in Python. Participants performed significantly better in the Java (67% correct) versus the Python (1% correct). They mostly responded that the Python statement will not produce a Typeerror:`print ("Friday is no:" + 1 )` because this behaviour is allowed in their existing Java knowledge: `System.out.println ("Friday is no:" + 1 )`. This result showed the influence of Java(PL1) on Python(PL2) learning and corroborates with Study 1 results, where now the results were vice-versa with Python (PL1) influencing Java (PL2) learning) as shown is section 5.1.6.

Still in the FCC category however, unexpected results that were not predicted were the influence of PL2 on PL1. An example was the *Array equality*, where most students scored a higher mark in the Python(67%) than the Java(33%). This kind of transfer in both directions is known as bi-directional transfer which has also been reported by other natural language transfer studies [7, 10, 29]. Possible explanations could be that: the construct is more intuitive in Python than Java; or the students had their first encounter with the construct in the Python language instead of Java language; or lastly, the student' prior knowledge on the construct was still very fragile hence susceptible to change. It should also be noted that the Study 2 participants were at a more advanced level of learning Python (week 6) than the Study 1 participants learning Java (week 3) hence it was more likely that their semantic knowledge will switch to

their PL2 (Python) at that stage. But the most important thing to note is that regardless of which direction the transfer is coming from, students still held *one* semantic model of the two languages, they seemed to struggle with having *two* semantic representations of a single concept as elaborated in the model of PL Figure 2, *Box A*.

Lastly, contrary to hypothesis 3, we did not find a significant difference in the ATCC score except for the *object aliasing* concept. Participants did not seem to struggle with ATCC constructs of *lists* and *dictionaries*. These results imply that participants found it easier to transfer their knowledge of ATCC from a language that has an abstract representation (eg. Java array-lists) to a concrete representation (e.g. Python lists). The only challenge they faced was with transferring their objects aliasing knowledge of a Java object to a Python dictionary and of Java array-lists to Python lists. This result was observed in Study 1, when the transfer was from Python to Java.

5.3 Summary Discussions and Teaching Implications

The proposed model of PL transfer provides a plausible explanation of the successes and difficulties of PL transfer. The model predicts that students' perceptions on the three construct categories (TCC, FCC and ATCC) impacts learning. In analyzing both the studies conducted, we find that:

- TCC are the easiest concepts to transfer as reported in Study 1 (Python to Java) and in Study 2 (Java to Python) because they have similar syntax and semantics hence cross language syntax matching and semantic transfer affects their learning in a new language positively. This supports hypothesis 1. Educators can make use of the similarities between languages and the advantages of semantic transfer in this category when teaching. They can take advantage of the knowledge students already have on these types of concepts to accelerate teaching a PL2.
- FCC are the most difficult concepts to transfer between programming languages because they have similar syntax and different semantics hence semantic transfer affects the learning negatively. These studies partially support hypothesis 2, however semantic transfer does not only occur from PL1 to PL2 as initially predicted, it can also occur bidirectionally from PL2 to PL1 as reported for some constructs in Study 2. If the differences between the languages are not explicitly explained, relative novices seem only able to hold one mental representations of the same concept in two languages that behaves differently in each. These concepts are more likely to be helpful than harmful if instructors can point out the differences between languages and take this as an opportunity to teach students a deeper understanding of programming concepts. For example, in explaining to students about the difference between the concept of *for-loop* in Java vs Python, the instructor can teach about deeper concepts such as iterator-based vs index-based *for-loops*.
- ATCC seem to be more challenging for students shifting from procedural Python to object-oriented Java (Study 1), as also confirmed by other studies [2, 28], than for students shifting from object-oriented Java to Python. This could be because moving from a more concrete representation (e.g Python dictionaries) to an abstract representation (e.g Java objects) where students encounter syntax such as class and constructors in Java can be a challenge. This explanation of the challenges in abstract conceptual learning is consistent with psychology research by Gentner [14] and natural language research[33]. To assist semantic transfer in these categories, instructors can point out the corresponding concept representations in PL1 so that students can map them to PL2.

6 LIMITATIONS AND FUTURE RESEARCH

Our study explored a model based on the notion of semantic transfer and the similarities between languages. There is currently no formal way to measure similarity in PLs hence follow-up studies should try to provide a formal guideline for such categorisations. Exploring the model of PL transfer in the context of other languages such as blocks-based to text-based languages can help give educators a clearer picture of the process of PL transfer. Lastly, a longitudinal study which follows the learner throughout the process would facilitate a more detailed exploration of PL transfer and proficiency level.

7 CONCLUSIONS

The motivation for this research was to understand how students transfer conceptual knowledge from their first to subsequent languages as they progress with their CS education. To answer this question, this study proposes a model of PL transfer with three categories of constructs (TCC, FCC and ATCC) that influence the learner to pick up those constructs in different ways. Specifically, the model categories use the notion of semantic transfer based on student' perceptions of syntax similarities. The model is investigated using two quantitative studies. Study 1 investigated near novice students transitioning from procedural Python to object-oriented Java while Study 2 investigated near novice postgraduate students transferring from object-oriented Java to procedural Python. The results of this study confirm that semantic transfer is positive to learning for concepts that share similar syntax and semantics (TCC) in either direction, from Python to Java or from Java to Python. They also show that semantic transfer is negative for concepts that share similar syntax but different semantics (FCC), suggesting that students struggle with holding two different semantic representations of the same concept in two languages such that semantic transfer was observed from Python to Java as well as from Java to Python. Finally, students transfer partial or no semantics on concepts that do not share similar syntax but have similar semantics (ATCC). The model of PL transfer can be used as a basis to explain the learning processes involved in language transfer and hence aid in guiding teachers who teach new programming languages.

ACKNOWLEDGMENTS

We would like to thank Dr Mary Ellen Foster and Dr Gerardo Aragon Camarasa for their support during this research.

REFERENCES

[1] Michal Armoni, Orni Meerbaum-Salant, and Mordechai Ben-Ari. 2015. From scratch to "real" programming. *ACM Transactions on Computing Education (TOCE)* 14, 4 (2015), 25.

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

[3] Alethea Liane Blackler and Jorn Hurtienne. 2007. Towards a unified view of intuitive interaction: definitions, models and tools across the world. *MMI-interaktiv* 13, 2007 (2007), 36–54.

[4] Jeffrey Bonar and Elliot Soloway. 1985. Preprogramming knowledge: A major source of misconceptions in novice programmers. *Human–Computer Interaction* 1, 2 (1985), 133–161.

[5] Wanda Dann, Dennis Cosgrove, Don Slater, Dave Culyba, and Steve Cooper. 2012. Mediated transfer: Alice 3 to Java.. In *SIGCSE*, Vol. 12. Citeseer, 141–146.

[6] Steven Davis. 1992. *Connectionism: Theory and practice*. Number 3. Oxford University Press on Demand.

[7] Tamar Degani, Anat Prior, and Natasha Tokowicz. 2011. Bidirectional transfer: The effect of sharing a translation. *Journal of Cognitive Psychology* 23, 1 (2011), 18–28.

[8] Joachim Diederich. 1988. *Knowledge-intensive recruitment learning*. International Computer Science Institute Berkeley, CA.

[9] Dimitrios Doukakis, Maria Grigoriadou, and Grammatiki Tsaganou. 2007. Understanding the programming variable concept with animated interactive analogies. In *Proceedings of the The 8th Hellenic European Research on Computer Mathematics & Its Applications Conference (HERCMA'07)*.

[10] Joel E Dworin. 2003. Insights into biliteracy development: Toward a bidirectional theory of bilingual pedagogy. *Journal of Hispanic Higher Education* 2, 2 (2003), 171–186.

[11] Sally A Fincher and Anthony V Robins. 2019. *The Cambridge handbook of computing education research*. Cambridge University Press.

[12] Kathi Fisler, Shriram Krishnamurthi, and Preston Tunnell Wilson. 2017. Assessing and teaching scope, mutation, and aliasing in upper-level undergraduates. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. 213–218.

[13] Vikki Fix and Susan Wiedenbeck. 1996. An intelligent tool to aid students in learning second and subsequent programming languages. *Computers & Education* 27, 2 (1996), 71–83.

[14] Dedre Gentner. 2005. The development of relational category knowledge. In *Building object categories in developmental time*. Psychology Press, 263–294.

[15] Mark Guzdial. 2018. Teaching Two Programming Languages in the First CS Course. *On the Internet at https://cacm. acm. org/blogs/blog-cacm/228006-teaching-two-programming-languages-in-the-first-cs-course/fulltext (visited August 2019)* (2018).

[16] Johannes Holvitie, Teemu Rajala, Riku Haavisto, Erkki Kaila, Mikko-Jussi Laakso, and Tapio Salakoski. 2012. Breaking the programming language barrier: Using program visualizations to transfer programming knowledge in one programming language to another. In *2012 IEEE 12th International Conference on Advanced Learning Technologies*. IEEE, 116–120.

[17] Maria Hristova, Ananya Misra, Megan Rutter, and Rebecca Mercuri. 2003. Identifying and correcting Java programming errors for introductory computer science students. *ACM SIGCSE Bulletin* 35, 1 (2003), 153–156.

[18] Christopher D Hundhausen, Sean F Farley, and Jonathan L Brown. 2009. Can direct manipulation lower the barriers to computer programming and promote transfer of training? An experimental study. *ACM Transactions on Computer-Human Interaction (TOCHI)* 16, 3 (2009), 1–40.

[19] Nan Jiang. 2000. Lexical representation and development in a second language. *Applied linguistics* 21, 1 (2000), 47–77.

[20] Nan Jiang. 2002. Form–meaning mapping in vocabulary acquisition in a second language. *Studies in Second Language Acquisition* 24, 4 (2002), 617–637.

[21] Nan Jiang. 2004. Semantic transfer and its implications for vocabulary teaching in a second language. *The modern language journal* 88, 3 (2004), 416–432.

[22] Walter Kintsch and Teun A Van Dijk. 1978. Toward a model of text comprehension and production. *Psychological review* 85, 5 (1978), 363.

[23] Walter Kintsch and CBEMAFRS Walter Kintsch. 1998. *Comprehension: A paradigm for cognition*. Cambridge university press.

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

[25] Shriram Krishnamurthi and Kathi Fisler. 2019. Programming paradigms and beyond. *The Cambridge Handbook of Computing Education Research* 37 (2019).

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

[27] Linxiao Ma, John Ferguson, Marc Roper, and Murray Wood. 2007. Investigating the viability of mental models held by novice programmers. In *Proceedings of the 38th SIGCSE technical symposium on Computer science education*. 499–503.

[28] H James Nelson, Gretchen Irwin, and David E Monarchi. 1997. Journeys up the mountain: Different paths to learning object-oriented programming. *Accounting, Management and Information Technologies* 7, 2 (1997), 53–85.

[29] Aneta Pavlenko and Scott Jarvis. 2002. Bidirectional transfer. *Applied linguistics* 23, 2 (2002), 190–214.

[30] Roy D Pea. 1986. Language-independent conceptual "bugs" in novice programming. *Journal of educational computing research* 2, 1 (1986), 25–36.

[31] Nancy Pennington. 1987. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive psychology* 19, 3 (1987), 295–341.

[32] 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), 1–24.

[33] Håkan Ringbom. 2007. *Cross-linguistic similarity in foreign language learning*. Vol. 21. Multilingual Matters.

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

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

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

[37] Jean Scholtz and Susan Wiedenbeck. 1992. An analysis of novice programmers learning a second language.. In *PPIG (1)*. 9.

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

[39] Carsten Schulte. 2008. Block Model: an educational model of program comprehension as a tool for a scholarly approach to teaching. In *Proceedings of the Fourth international Workshop on Computing Education Research*. ACM, 149–160.

[40] Carsten Schulte, Tony Clear, Ahmad Taherkhani, Teresa Busjahn, and James H Paterson. 2010. An introduction to program comprehension for computer science educators. In *Proceedings of the 2010 ITiCSE working group reports*. ACM, 65–86.

[41] Ravi Sethi. 1989. *Programming languages: concepts and constructs*. Addison-Wesley Longman Publishing Co., Inc.

[42] Ben Shneiderman and Richard Mayer. 1979. Syntactic/semantic interactions in programmer behavior: A model and experimental results. *International Journal of Computer & Information Sciences* 8, 3 (1979), 219–238.

[43] Elliot Soloway, Beth Adelson, and Kate Ehrlich. 1988. Knowledge and processes in the comprehension of computer programs. *The nature of expertise* 129 (1988), 152.

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

[45] David R Thomas. 2003. A general inductive approach for qualitative data analysis. (2003).

[46] Ethel Tshukudu and Quintin Cutts. 2020. 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*. 307–313.

[47] Anneliese Von Mayrhauser and A Marie Vans. 1995. Program comprehension during software maintenance and evolution. *Computer* 28, 8 (1995), 44–55.

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

[49] David A Watt. 2004. *Programming language design concepts*. John Wiley & Sons.

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

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

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