TECHNISCHE UNIVERSITÄT
CHEMNITZ

Observations of Knowledge Transfer in Novice Programmers Learning Java as Their Second Programming Language

Master Thesis

Submitted in Fulfilment of the
Requirements for the Academic Degree
M.Sc.

Dept. of Computer Science
Chair of Software Engineering

Submitted by: Yifan Du
Student ID:
Date: 07.02.2024

Supervising tutor: Prof. Dr. Janet Siegmund
Dominik Gorgosch

Acknowledgement

I would like to express my heartfelt gratitude to my supervisors Prof. Dr. Janet Siegmund and Dominik Gorgosch, and my former supervisor Elisa Hartmann, for their steadfast support, invaluable guidance and great patience throughout my master thesis.
I am also deeply thankful to my friend in the lab, Belinda, for her precious support, which has significantly improved the quality of my thesis. Special thanks are extended to Susan Köhler, for providing the helpful facilities during the process of my thesis.
I would like to thank my family for always giving me huge encouragement and opportunities to pursue my dream. Without them, I wouldn't be where I am today.
Lastly, I would like to acknowledge all the participants of my questionnaire. Their contributions were vital in the completion of my thesis.
This master thesis would not have been possible without the collective support of all these individuals. Thank you for being part of this journey.

Abstract

Background: Many studies have been done to discover the problems of programmers when they learn second and subsequent programming languages. The difficulty is even greater for novice programmers. There is a process of knowledge transfer when switching from one programming language to another. Analyzing the thoughts of novices during code comprehension provides an insight into the knowledge transfer that occurs.
Objective: We wanted to find out what kind of transfers could be observed from novices in the process of code comprehension in a new programming language. Especially what kind of transfer could happen intuitively on them. With this information, we hope to contribute to the development of programming pedagogy to mitigate the difficulty experienced by novices when going through the learning transfer phase.
Method: A questionnaire containing Java snippets was used to collect responses from novices. Novices were asked to comprehend the snippets and write the outputs. The number of novices participating in the questionnaire was 104.
Results: For novices with different programming background: positive transfer was observed more often when they comprehended snippets containing concepts that shared similar syntax and the same semantics in Java and their learned programming languages. Negative transfer was often observed in comprehension of snippets containing concepts that had similar syntax but distinct semantics in Java and the learned programming languages. No transfer happened to novices when they comprehended snippets containing concepts with different syntax but the same semantics in Java and the learned programming languages.
Conclusion: Novice programmers have the ability to transfer acquired programming knowledge to similar new scenarios, however, sometimes they still face difficulty in doing so. Without learning, they can barely predict the meaning of snippets that include unfamiliar syntax, even though some snippets have the same underlying semantics as the snippets written in their learned programming languages.
Future Work: Interviews or post-questionnaires can be done to have a deeper understanding of the thoughts of novices during code comprehension. Another code comprehension questionnaire can also be delivered to these novices after they learning Java for some time to find out the changes in learning transfer. Additional instrumentation, such as EEG devices and eye trackers, can also be used to discover more about the behavior of novices as they read code snippets.

Keywords: programming knowledge transfer, program comprehension, novice programmers, Java, syntax and semantics

Contents

Contents **4**

List of Figures **7**

List of Tables **8**

List of Abbreviations **9**

1 Introduction **10**
1.1 Problem Statement 10
1.2 Research Goal 11
1.3 Structure 11

2 Theoretical Background **13**
2.1 Novice Programmers 13
2.2 Syntax and Semantics 14
2.2.1 Syntax in Programming Languages 14
2.2.2 Semantics in Programming Languages 14
2.2.3 Relationship of Syntax and Semantics 15
2.3 Object-Oriented Programming and Procedural Programming 16
2.3.1 Objected-Oriented Programming 16
2.3.2 Difficulty in Teaching OOP 16
2.3.3 Procedural Programming 17
2.4 Program Comprehension 17
2.5 Learning Transfer 18
2.6 Transfer in Programming Languages 19
2.6.1 What Is Programming Language Transfer 19
2.6.2 Difficulty for Novices 20
2.7 The Model of Programming Language Transfer 20
2.7.1 MPLT Presented in A Figure 20
2.7.2 Three Categories of Concept 21

3 Related Work **22**
3.1 Programming Learning of Novice Programmers 22
3.1.1 Programming Environments and Languages 22
3.1.2 Mistakes Made by Novices 22
3.1.3 Overall Performance of Novices 24

3.1.4 Confusion of Novices and Other Studies . . . 24
3.2 Learning Transfer Study . . . 25
3.2.1 Transfer of Learning in CS . . . 25
3.2.2 Programming Language Transfer of Experienced Programmers 25
3.2.3 Programming Language Transfer of Novices . . . 26

4 Methodology . . . 29
4.1 Research Objects . . . 29
4.1.1 Variables . . . 29
4.1.2 Research Questions . . . 30
4.1.3 Hypotheses . . . 30
4.2 Participants . . . 31
4.3 Materials and Tasks . . . 31
4.3.1 Experience Questionnaire . . . 31
4.3.2 Java Questionnaire . . . 32
4.4 Experimental Design . . . 38

5 Results . . . 41
5.1 Data Preprocessing . . . 41
5.2 Descriptive Analysis . . . 42
5.2.1 Description of the Subjects . . . 42
5.2.2 Java Questionnaire Results . . . 42
5.3 Categorization of Results . . . 50
5.3.1 Categories of Question 1 - Variable Type . . . 50
5.3.2 Categories of Question 2 - Variable Scope . . . 52
5.3.3 Categories of Question 3 - While Loop . . . 55
5.3.4 Categories of Question 4 - String Comparison . . . 56
5.3.5 Categories of Question 5 - Method Calling . . . 59
5.3.6 Categories of Question 6 - Object Reference Assignment . . . 62
5.3.7 Categories of Question 7 - Memory Allocation . . . 65
5.3.8 Categories of Question 8 - String Concatenation . . . 69
5.3.9 Categories of Question 9 - Array Length . . . 71
5.3.10 Categories of Question 10 - OOP Related . . . 74
5.4 Individual Analysis . . . 78
5.4.1 Special Case of Three Students . . . 78
5.4.2 Confusion During Interpretation . . . 80

6 Discussion . . . 81
6.1 Assessment of Results . . . 81
6.1.1 Transfer Process . . . 81
6.1.2 Answers to the Research Questions . . . 90
6.1.3 Other Findings . . . 91
6.2 Threat to Validity . . . 92
6.2.1 Construct Validity . . . 92

6.2.2 Internal Validity 93
6.2.3 External Validity 94
6.3 Comparison to Related Work 94

7 Conclusion and Future Work 96
7.1 Conclusion 96
7.2 Future Work 97

Bibliography 98

List of Figures

2.1 Syntax Example of C 15
2.2 Example of an Inheritance Hierarchy [51] 17
2.3 A Flow Chart Description of Procedural Programming [44] 18
2.4 The Model of Programming Language Transfer [48] 20

3.1 Example of Transfer Tutor [37] 26

4.1 Screenshot of Question 1 - Variable Type 33
4.2 Snippet of Question 2 - Variable Scope 33
4.3 Snippet of Question 3 - While Loop 34
4.4 Snippet of Question 4 - String Comparison 34
4.5 Snippet of Question 5 - Method Calling 34
4.6 Snippet of Question 6 - Object Reference Assignment 35
4.7 Snippet of Question 7 - Memory Allocation 36
4.8 Snippet of Question 8 - String Concatenation 37
4.9 Snippet of Question 9 - Array Length 38
4.10 A Structure Example in C 38
4.11 An Example Accessing Length of Structure in C 39
4.13 Process of Experiment for C Background Participants 39
4.12 Snippet of Question 10 - OOP Related 40

5.1 Bar Chart of Answers of Question 1 - Variable Type 44
5.2 Bar Chart of Answers of Question 2 - Variable Scope 44
5.3 Bar Chart of Answers of Question 3 - While Loop 45
5.4 Bar Chart of Answers of Question 4 - String Comparison 45
5.5 Bar Chart of Answers of Question 5 - Method Calling 46
5.6 Bar Chart of Answers of Question 6 - Object Reference Assignment . 46
5.7 Bar Chart of Answers of Question 7 - Memory Allocation 47
5.8 Bar Chart of Answers of Question 8 - String Concatenation 48
5.9 Bar Chart of Answers of Question 9 - Array Length 48
5.10 Bar Chart of Answers of Question 10 - OOP Related 49

List of Tables

5.1 Programming Experience Distribution of Participants 42
5.2 Categorization of Results in Question 1 - Variable Type 51
5.3 Categorization of Results in Question 2 - Variable Scope 53
5.4 Categorization of Results in Question 3 - While Loop 55
5.5 Categorization of Results in Question 4 - String Comparison 57
5.6 Categorization of Results in Question 5 - Method Calling 59
5.7 Categorization of Results in Question 6 - Object Reference Assignment 63
5.8 Categorization of Results in Question 7 - Memory Allocation 67
5.9 Categorization of Results in Question 8 - String Concatenation 69
5.10 Categorization of Results in Question 9 - Array Length 72
5.11 Categorization of Results in Question 10 - OOP Related 76

List of Abbreviations

PL Programming Language

OOP Object-Oriented Programming

RQ Research Question

TUC Technical University of Chemnitz

EEG Electroencephalogram

MPLT Model of Programming Language Transfer

TCC True Carryover Construct

FCC False Carryover Construct

ATCC Abstract True Carryover Construct

CS Computer Science

2D Two-Dimensional

1 Introduction

The evolution of computer science is closely related to the development of programming languages. Originating as a discipline rooted in mathematical theory and machine-specific coding, computer science has evolved into a complicated domain characterized by a diverse array of programming languages designed to address different computational challenges.

The development of programming languages begins with early machine-specific languages and assembly languages. The field has witnessed a transformative shift with the advent of high-level languages catering to distinct programming paradigms and application domains. Under this circumstances, the demand for programmers have grown rapidly since two decades ago [32].

1.1 Problem Statement

Due to the increasing popularity of computer science and programming, the number of people interested in programming has grown swiftly [32]. People always start their coding journey by learning one programming language (PL). Whereas, when they are in different situations, they may be required to solve problems using various PLs. For example, programmers working in distinct fields need to use multiple PLs to implement software with several functions, use different tools, or program hardware and software at the same time. Students need to learn PLs because of taking different courses. As a result, everyone needs to learn another PL, and then another, and has to experience the process of transferring from one PL to another.

However, it is a hard task to learn second and subsequent programming languages [35]. A research shows that problems can be met in syntactic, semantic, or planning level of programming knowledge [35]. Experienced programmers pay a lot of attention to syntax of the new PL because they know it is important to make the program compile at least. Programmers' past syntax knowledge of learned PL can also lead to wrong assumptions. The new PL may have constructs that have no similarity to the learned PLs, which does not provide any shortcut for learning it. Besides, semantic errors are not easy to deal with by reading the error messages generated at run time. Programmers also face trouble in strategic planning, tactical planning, and implementation planning phase [35].

The difficulty of learning subsequent PLs is even greater for novice programmers who are not so familiar with programming languages themselves. There are five areas of issues when novices learning to program [10]. These are general problem of orientation, the notional machine, notation, structures, and pragmatics of pro-

gramming. General problem of orientation means the purpose of programs, and the advantages that one can get after learning programming. The notional machine is related to understanding the general properties of the machine used to execute programs. Notation stands for understanding the syntax and underlying semantics of PLs. Structures have the meaning of acquiring plans to achieve a small scale goal. Pragmatics of programming represents the skill of specifying, developing, testing, and debugging a program [10]. Among these problems, syntax and semantics mistakes are commonly seen in novices. A large number of novice programmers write code that does not compile, and may not be able to solve syntax problems [8]. According to a study, semantic errors occur more frequently than syntax errors [3].

Due to the large difference between programming paradigms, it is problematic for programmers and novices to transfer from languages supporting one paradigm to another. For instance, moving from procedural programming languages to objected-oriented programming languages is difficult [27]. This means transferring from C to Java can be hard for novices. Since Java is one of the most popular programming languages [13], and C is still a common choice as the first programming language in many university computer science and engineering programs, this transfer problem is common among novices.

1.2 Research Goal

To have a deeper look into this situation, this study about programming knowledge transfer, program comprehension, Java, C, Python, syntax, and semantics was carried out. During the experiment, a questionnaire containing 10 Java snippets was distributed to students starting to learn Java in Data Structures course at Technical University of Chemnitz (TUC), and 104 responses were received.

By conducting this study, how novices programmers naturally link their learned programming knowledge to a new PL before they start learning it can be observed. What kind of errors they make can be found. Afterwards, it is helpful for improvement of pedagogy on how to make it easier for students to go through the learning transfer phase.

Therefore, the research questions are set as follows.

- **Research Question 1 (RQ1): What kind of transfer can be observed in novice programmers during Java code comprehension if they only have programming knowledge in C?**
- **Research Question 2 (RQ2): What kind of transfer can be observed in novice programmers during Java code comprehension if they have programming knowledge in various programming languages?**

1.3 Structure

This thesis is organized as follows.

In chapter 2, theoretical knowledge used in the thesis is described, which includes the introduction of novices, the description of syntax and semantics, differences in programming paradigms, program comprehension, learning transfer, programming language transfer, and a model to describe programming language transfer.

In chapter 3, some related work is outlined in two categories, studies of programming learning of novice programmers, and programming language transfer studies.

In chapter 4, the methods used in this thesis, the design of questionnaire, the participants, and the process of carrying out the experiment are presented in detail.

In chapter 5, the results of this thesis are shown in charts. These charts present the correctness of the answers to questions in the questionnaire from participants. Afterwards, categories of results of each question are elaborated and presented in tables. In the end, analysis of some special cases is described.

In chapter 6, discussion about the results of the experiment is introduced by assessing the results of the Java questionnaire, discussing the threats to validity in three types (construct, internal, and external validity), and comparison to related work.

In chapter 7, conclusion of this thesis and future work are presented.

2 Theoretical Background

In this chapter, theoretical background of this thesis will be introduced, which includes novice programmers (section 2.1), the definition of syntax and semantics of programming languages (section 2.2), the difference between object-oriented programming (OOP) languages and procedural programming languages (section 2.3), what program comprehension is (section 2.4), the process of learning transfer (section 2.5), transfer in programming languages (section 2.6), and a model to describe the programming language transfer (section 2.7).

2.1 Novice Programmers

Novice programmers refer to individuals who are beginners to the field of programming. They are typically at the early stages of learning how to write, understand, and design computer programs. They may have limited experience with coding, and may be in the process of acquiring foundational knowledge and skills related to programming languages, algorithms, problem-solving, and software development.

Research focusing on novice programmers has been ongoing for several decades. Corritore and Wiedenbeck discovered that novice programmers were creating a detailed, step-by-step mental model of programs to enhance their understanding no matter the program was short or not [6]. Various studies have concluded that novices possess an insufficient understanding of the area, have only a limited surface knowledge of subject, use generic problem solving approach instead of specific strategies dependent on the given problem, tend to program using control structures, and write code using a line-by-line method to solve problems [32, 42, 55]. Lahtinen et al. stated that programming is difficult to novices not only because of the abstract concepts, but also because they have problem understanding program construction [21]. Bonar and Soloway found that novices' understanding of programming are influenced by incompatibilities between natural and programming languages, which may cause trouble for understanding semantics of programming languages [1]. Robins et al. presented distributions of different PL related problems novices met in an introductory programming course [31]. According to their data, trivial mechanics is most frequently seen type of problems, which supports the conclusion that novices' issues with basic design can be more notable than specific PL construct issues [42, 55].

When educating a novice programmer into an expert, the novice will experience a continuous process. One of the most famous statements that break this process into stages is created by Dreyfus and Dreyfus. These stages are Novice, Advanced

beginner, Competent, Proficient, and Expert [9]. Because of the large amount of problems novices face when learning to program, it is hard to train a novice into an expert. A widely accepted time frame is about 10 years [55].

2.2 Syntax and Semantics

Syntax and semantics are crucial for effective coding when learning a programming language, because they form the basis of communication between programmers and computers. In this section, syntax and semantics of programming languages, and the relationship between them are introduced.

2.2.1 Syntax in Programming Languages

To a natural language, syntax refers to the set of rules that dictate the proper arrangements of symbols, words, or elements to form grammatically correct sentences. To a programming language, the definition is similar. Syntax in PLs defines the formal relations between the element of a language, therefore, it provides a structural description of the different expressions that form legal strings in the language [40]. Syntax only handles the form and structure of elements in a language without considering the meanings of them [40].

Here is an example of syntax of C programming language. See at Figure 2.1.

In this example, `"#include <stdio.h>"` is used to include the standard input/output library for related functions. `"printf"` function used in the main function belongs to this library. `"int addNumbers(int a, int b)"` defines a function that returns an integer after execution, and takes two integers a and b as input parameters. `"int main()"` defines the main function, which is the entry of the program. `"int num1 = 5"` declares a integer variable named num1, and initializes it with number 5. After this, the function addNumbers is called, and conditional statements (if) are used.

2.2.2 Semantics in Programming Languages

Semantics discloses the meaning of syntactically correct sentences in a language [40]. For PLs, it presents the behavior of a machine when executing a program in the language [40]. Natural languages can be used to describe this behavior by pionting out the relationship between the input and output of a program, or explaining line by line what the machine does during execution [40].

There are three types of semantics of programming languages [54]. Operational semantics describes how a PL executes on an abstract machine. It focuses on the transitions from one program state to another as individual operations or steps are performed. For example, in operational semantics, how an expression like `"x = x + 1"` leads to changes in the value of the variable x and the program state will be specified. Denotational semantics used to be called mathematical semantics. It

```c
#include <stdio.h>

int addNumbers(int a, int b) {
    int sum = a + b;
    return sum;
}
int main() {
    int num1 = 5;
    int num2 = 7;
    int result = addNumbers(num1, num2);
    printf("The sum of %d and %d is: %d\n", num1, num2, result);
    if (result > 10) {
        printf("The result is greater than 10.\n");
    } else {
        printf("The result is not greater than 10.\n");
    }
    return 0;
}
```

Figure 2.1: Syntax Example of C

uses abstract mathematical objects or functions to define the meaning of programs. For instance, in denotational semantics, the meaning of a loop construct will be defined by showing a mathematical function that represents the behavior of the loop. Axiomatic semantics attempts to determine the meaning of programming constructs by giving proof rules in program logic. Among these types of semantics, operational semantics is mostly used to teach novice programmers.

2.2.3 Relationship of Syntax and Semantics

According to the definition of syntax and semantics of PLs, they are strongly connected. Correct syntax is a necessary condition for meaningful semantics [40]. If the syntax of a program is incorrect, it may not be executable, and the semantics of it becomes useless. Syntax provides rules for expressing program structures, and makes programs clear and readable. On the other hand, semantics defines the actual meaning and machine behavior translated from these structures. If the semantics of a program is not clear, the program is definitely incomprehensible even if the syntax is correct.

2.3 Object-Oriented Programming and Procedural Programming

Objected-oriented programming (OOP) and procedural programming are two types of programming paradigms, and represent different approachs to structuring and organizing code, as well as managing the state and behavior of programs. In this section, definition and features of these programming paradigms are discussed.

2.3.1 Objected-Oriented Programming

Objected-oriented programming is centered around the concept named "objects". "Objects are collections of operations that share a state" [51], and these operations are used to manipulate data which is also part of objects [56]. For example, a Bird object may consist of data such as number of legs, color, gender, living area, and age. The Bird object may have operations for changing the age value, changing the living place, adding more data, and deleting some values. Classes are also a key concept in OOP. They are templates from which objects can be created [51], and include rules of what objects can and cannot do [56]. An object is called an instance of a class, and can only be an instance of exactly one class [56]. If there are two or more distinct entities that share many common features, inheritance is used to design them [56]. Inheritance makes it possible to reuse the behavior of a class when defining new classes [51]. The class with all the common features is called the superclass, and all the calsses that inherit from superclass are called subcalsses [56]. Subclasses are allowed to add new operations and new instance variables [51]. Below is a figure showing an inheritance hierarchy. See at Figure 2.2. Mammal is the superclass of Person and Elephant, and has Person and Elephant as its subclasses. Person has Student and Female as its subclasses. John, Joan, Bill, and so on are instances.

Encapsulation and polymorphism are also important concepts of OOP. Encapsulation is a technique to minimize interdependencies among modules that are written separately [41], which makes it easier to modify programs [56]. Polymorphism allows treating objects of different types as objects of a common type. It "enables programmers to manipulate subclass objects using superclass references" [18]. Popular languages that support OOP are Java, Python, C++, C#, and so on.

2.3.2 Difficulty in Teaching OOP

OOP is a good tool for teaching important programming methodologies, however, teaching OOP is still difficult [19]. According to Gutiérrez et al. [14], some problems are found related to learning and teaching process of OOP. Novices may have difficulty in understanding object and its dynamic nature, related to understanding classes, understanding the concept of method, understanding object-oriented relationships, understanding encapsulation, and understanding polymorphism and

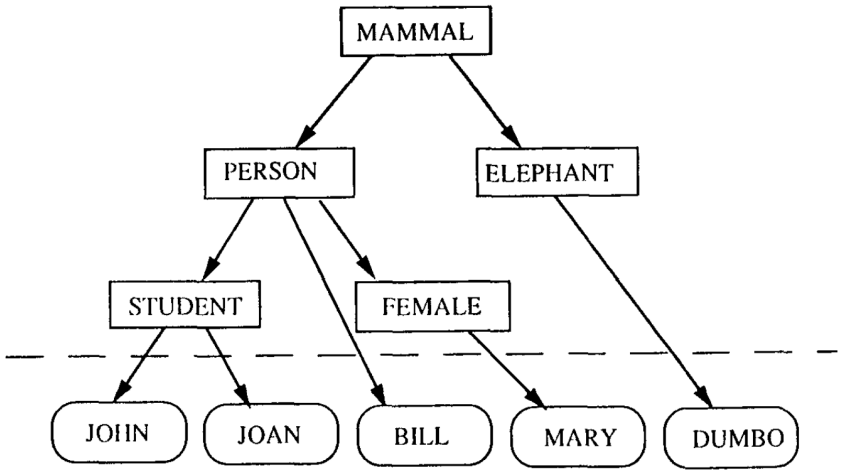

Figure 2.2: Example of an Inheritance Hierarchy [51]

overload.

2.3.3 Procedural Programming

Procedural programming was the de facto approach in the early days of programming [29]. It emphasizes the use of procedures, routines, or subroutines to structure a program. Usually, a program is divided into different functions, each implementing a specific operation. These functions are called procedures or routines. Below is a visualization of procedural programming example in flow chart (Figure 2.3). In this flow chart, the starting point is the black filled circle on the top. By following the direction of arrows, and selecting the result of a conditional statement, operations are executed. In the end, the machine running this program will find an object, grasp it with its empty hand, go to home, and hand over the object.

Kölling stated that many textbooks introduce procedural programming as a starting point to object concepts [19]. This means that maybe many people learn a procedural programming language first, for example, C, Fortran, Pascal, and so on. They are familiar with creating functions, calling functions, and solving problems with them. Due to the growing popularity of OOP, these people may need to learn a OOP language at a certain time. However, transferring from procedural programming to OOP is harder than transferring backwards [19]. They need to pay more efforts getting used to the features OOP has.

2.4 Program Comprehension

Program comprehension stands for the process of understanding how a software system or part of it works [24]. It involves mental activities and cognitive processes of programmers. During software maintenance, programmers spend around half

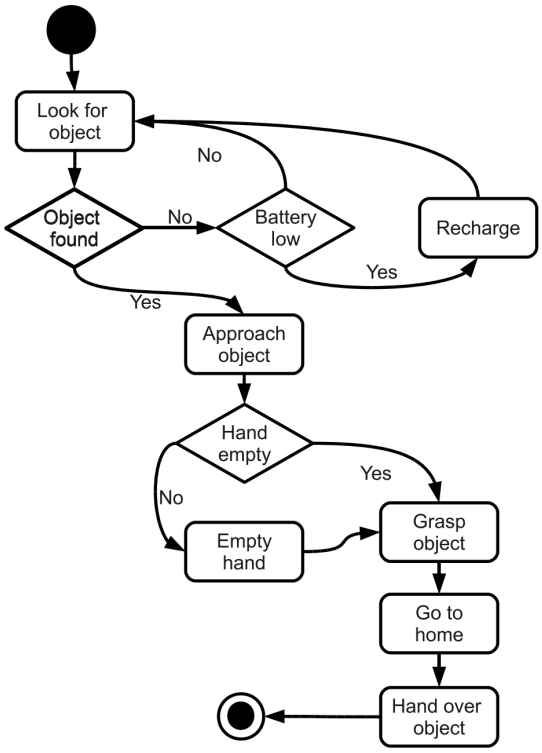

Figure 2.3: A Flow Chart Description of Procedural Programming [44]

of their time understanding programs [11]. Obviously, comprehension of codes is unavoidable for people learning programming.

Research about program comprehension has started since four decades ago [2]. In the past time, there are three common approaches for measuring program comprehension: think-aloud protocols, memorization, and comprehension tasks [38]. Some of these approaches are inappropriate for modern research, but they are the inspirations of methods commonly used today. New emerging methods and techniques are questionnaires and surveys, eye tracking methods, cognitive load assessments, and so on. Think-aloud protocols are still used. Through analyzing the participants' answers to the questionnaire or survey, data collected from eye tracking machines or EEG devices, the thought processes of participants can be detected. When using think-aloud protocols, participants will speak their thoughts out, so that these thoughts can be audio or videotaped. Afterwards, the tapes are transcribed first, then the transcriptions are representing the thinking process of the participants.

2.5 Learning Transfer

A definition of learning transfer is the ability of a person to expand what has been learned in one situation to new situations [5]. There are positive transfer and negative transfer [28].

Positive transfer occurs when the knowledge learned in one case have positive effects on performance in other cases [28]. For example, if someone learns how to drive a car, it is easy for him or her to learn driving a bus. If someone is a native English speaker, he or she may be able to learn French fast. This positive effect on learning reduces the efforts and time taken by both students and teachers, increase the performance of learning, and increased confidence for taking challenges. Therefore, it is significant to leverage positive transfer [28].

Negative transfer appears if the knowledge learned in one context causes negative influence on performance in other context. For instance, there are also negative transfer occurring in the example of a person learning to drive a car mentioned above. Even though driving a car or a bus share similarities, like starting the engine, controlling the steering wheel, paying attention to rear-view mirrors, there are also many difference between them. The person gets distinct perspectives sitting in a car or in a bus. The sizes of a car and a bus are different. These variances influence the learning of this person, and may cause mistakes. Fortunately, the trouble caused by negative transfer typically occurs only in the beginning stages of learning [28]. With the growth of experience in the new domain, learners will avoid the influence of negative transfer [28].

2.6 Transfer in Programming Languages

In the last section, learning transfer was introduced. It also occurs in learning process of different programming languages. In this section, the transfer in programming languages is discussed.

2.6.1 What Is Programming Language Transfer

Programming language transfer means the transfer of learning happening in the context of learning different programming languages. This transfer can happen for various reasons, for example, learning a new language for a specific project, exploring new technologies, or seeking for self improvements.

When people learn a PL, they start with learning fundamental concepts of a PL, such as variables, data types, control structures (if statements, loops), and functions. By learning these, they know the syntax, semantics and structure of simple programs. Afterwards, they need to practice what they have learned repeatedly through writing code and solving problems in the new PL, and gradually move onto more complex projects. In this learning process, both positive and negative transfers are observed in novice programmers and experienced programmers [35, 57]. For experienced programmers, positive transfer occurred when the new PL has similar elements as the former PL. Negative transfer took place when the new PL has distinct constructs from the former PL.

2.6.2 Difficulty for Novices

Novice programmers have not reached a certain level of proficiency in PL itself, so their problem-solving abilities are relatively low. Their lack of programming knowledge is likely to affect their judgment on similar or different problems.

Positive transfer may not occur as expected on novices. In research of Teague and Lister, novices were commonly seen not be able to transfer the concepts taught on a prior problem to the next programming problem [45]. In the study of Izu and Mirolo, around half of the novices tried to reuse their knowledge in the exam, but only about half of these novices were successful [15]. Based on these facts, negative transfer may cause them insurmountable trouble.

2.7 The Model of Programming Language Transfer

To describe the transfer of learning programming language of novices, a model was proposed by Tshukudu and Cutts [48]. In this section, the model of programming language transfer (MPLT) is introduced.

2.7.1 MPLT Presented in A Figure

The model of programming language transfer can be presented in a figure (See at Figure 2.4). According to Tshukudu and Cutts [48], as shown in the figure, the

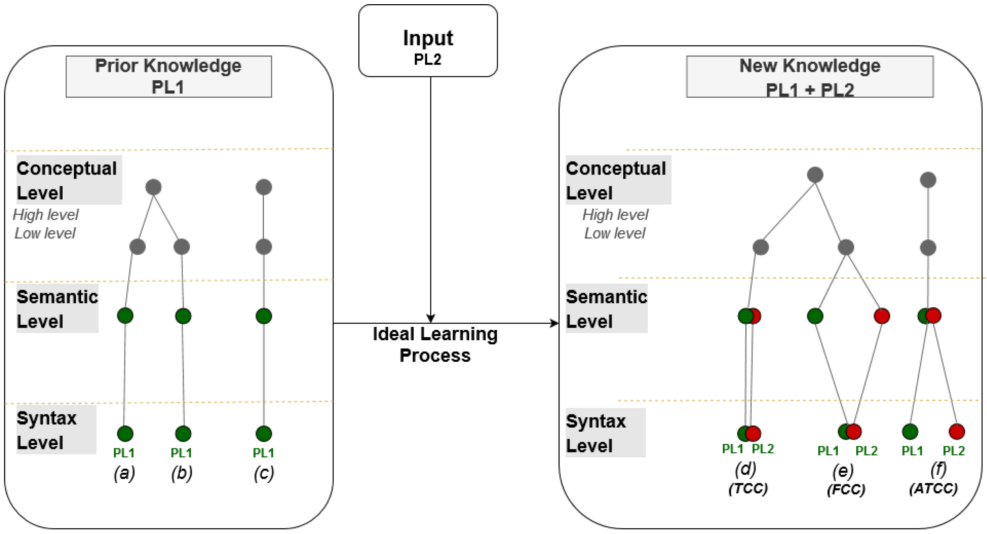

Figure 2.4: The Model of Programming Language Transfer [48]

knowledge of programmers is represented as a network of nodes. These nodes are connected at different levels: conceptual level, semantic level, and syntax level. The conceptual level contains the underlying concepts of PLs. The semantic level is related with the semantics of programs. The syntax level stands for the syntax of a

PL. A route starting from the conceptual level to the syntax level stands for a concept in a PL, the semantics of this concept, and the syntax of this concept. Therefore, for one concept, there could be different sub-concepts. For each sub-concept, there is one semantics related to it. For one semantics, there could be different syntax to implement this semantics. Whereas, before the novice programmer learn other PLs, there is only one syntax connected with one semantics. When the novice programmer starts to learn his or her first PL, the nodes at conceptual level are created, and the links between nodes are formed. For example, the nodes (a) and (b) could be the knowledge of two types of repetition concept: iteration and recursion. Node (c) could be the concept of function return. Afterwards, when the novice learns the second PL, more nodes and connections are appearing. New concepts will add nodes to the conceptual level, and other knowledge also adds nodes or connections to the route. On the right hand of the figure, the relationships that nodes (d), (e), and (f) presented stand for true carryover construct (TCC), false carryover construct (FCC), and abstract true carryover construct (ATCC), respectively.

2.7.2 Three Categories of Concept

As mentioned above, the nodes (d), (e), and (f) represent three types of ideal relationships between different concepts in these two PL. TCC, FCC, and ATCC are three concept categories that show this concept relationship.

TCC is a construct that has similar syntax and the same underlying semantics in both PLs. For example, declaration of an integer is contained in both C and Java, which means the same semantics. The expression, `int number = 2;`, is also syntactically correct in both C and Java. FCC is a construct that has similar syntax but distinct semantics in the two PLs. For instance, the handling of pointers and references in C and Java. The syntax for declaring and initializing a pointer or a reference is similar in both languages, however, there is no no pointer in Java as in C. ATCC is a construct with different syntax but the same semantics in the two PLs. The example for this construct is defined structures in and objects in Java. In C, there is no concept of objects, nevertheless, defined structures can be used to present simple objects. Other examples for this construct are those with hidden implementation details, such as data abstraction in Java, which "at a low level are data structures like Python dictionaries" [48].

3 Related Work

In this chapter, some related work is presented. They are categorized into two main parts: research about novices learning to program (section 3.1), and learning transfer studies (section 3.2). In the first part, programming environments and PLs invented for novices, mistakes, performance, and confusion of novices are mentioned. In the second part, transfer studies are presented in general related to CS first. Then, the programming learning transfer is explored considering experienced programmers and novices.

3.1 Programming Learning of Novice Programmers

Prior to looking at the programming learning procedure of novice programmers, it is necessary to first have an idea on the programming environments and PLs that novices may use.

3.1.1 Programming Environments and Languages

Kelleher and Pausch created a taxonomy of programming environments and languages [17]. Researchers have been putting in efforts in building different PLs and environments to make programming easier for more people [17]. The taxonomy of Kelleher and Pausch [17] divides systems first into two large groups: systems attempting to teach programming, and systems attempting to support the use of programming in other contexts. Example systems in the first group are: BASIC, LogoBlocks, Magic Forest, Pascal, Cleogo, and Robocode. Systems belong to the second groups are: Pygmalion, AgentSheets, COBOL, AutoHAN, Hypercard, and so on. These systems tried to make programming accessible by simplifying the mechanism of programming, providing support for novice programmers, and giving them motivation of learning to program [17].

3.1.2 Mistakes Made by Novices

With the help of these systems and special design PLs, the programming learning should be easier for novices. Whereas, there are still a lot mistakes faced by novice programmers during their learning. The mistakes are introduced in this section in two different ways.

3.1.2.1 General Programming Mistakes

Many mistakes of novices were found by Brown and Altadmri through analyzing the Blackbox data set of over 100,000 students [3]. They categorized the mistakes into three main categories: misunderstanding (or forgetting) syntax, type errors, and other semantic errors. The frequency of mistakes of their study showed that among the 10 most frequently detected mistakes, 5 kinds of mistakes are semantic errors, 4 types of mistakes are syntax errors, and only 1 mistake belongs to the type errors [3].

McCall and Kölling also did research on novice programmers' errors [25, 26], and an error category hierarchy was developed. In their first study, the amount of error categories identified was 80, which can be divided into three main categories: syntactic, semantic, and logic errors [25]. They also performed an analysis about frequency of error categories, and the most frequently found error is "Variable not declared". McCall and Kölling did another study on error categorization in 2019 [26]. They updated a way of defining the severity of a type of error (`severity = frequency × difficulty`) to using error frequency and time-to-fix statistics. As a result, 90 error categories were identified. In general, the three broad categories are: syntactical, semantic, and logical. They continued to make a finer-grained selection of top-level categories, and 11 categories were chosen. Some examples of their final top-level categories of mistakes are: "Variable: Incorrect attempt to use variable", "Variable: Incorrect variable declaration", "Method: Incorrect method call", "Method: Incorrect method declaration", and so on [26].

3.1.2.2 Java Programming Mistakes

Brown and Altadmri [4] analyzed the Blackbox dataset again in their later research, and fond that the most frequently appearing mistake of novices learning Java was related to brackets. Novices would write unbalanced parentheses, curly brackets, square brackets, and quotation marks, and so on. The second frequently met mistakes in Java was about methods. Novices tended to call methods with wrong arguments. For instance, they might write wrong types of arguments. The type of mistakes with the third highest frequency was connected with control flow. Novices could write control flows that would reach an end of a method without encountering a return statement, but the method should return a value. Among the top 10 constantly found mistakes in Java, 50% of them were semantics problems [4]. Two of the semantics mistakes both took the longest time for novices to solve. Brown and Altadmri [4] also found that the estimation of educators about the programming mistakes that students were likely to make was not consistent with the data of students.

Jackson et al. [16] stated that the top ten Java errors were: "cannot resolve symbol", "; expected", "illegal start of expression", "class or interface expected", "¡identifier¿ expected", and ") expected". Their results supported the findings of Brown and Altadmri [4] that mistakes related to brackets were commonly seen.

According to the research of Rodrigo et al. [33], a large amount of students had difficulty understanding event-driven programming in Java. For example, they had a hard time understanding threads, exception, event handling, and so on. The concepts in Java related to OOP were also difficult for students to comprehend. The instances were polymorphism, inheritance, and encapsulation. Some students met trouble when learning general data structures, such as declaring and work with multidimensional arrays.

These findings of mistakes all present the difficulty novices met during learning Java programming. However, Java is one of the most prevalent PLs [13], which means a lot of novices will choose to learn Java. Study about novices learning is still important for making it easier for novices.

3.1.3 Overall Performance of Novices

Besides the research about programming errors of novices, a study about the overall performance of novices had been done by Lopez et al. [23]. They analyzed the responses of students during an examination, after these students completed one semester of learning to program. The questions used in their exam were categorized into different groups: Basics, Sequence, Non-iterative tracing, Iteration tracing, Exceptions, Data, Writing, Explain, and General questions. By analyzing the data using stepwise regression, they found that code tracing and code writing skills of novices are strongly related, and code reading and code writing skills are also associated [23].

Venables et al. [50] did similar research to Lopez et al. [23]. Their findings supported the earlier findings of Lopez et al., which means there is "statistically significant relationships between tracing code, explaining code, and writing code". However, the results of Venables et al. uncovered that the relationship between tracing, explaining, and writing code differ significantly according to the difference of tasks. They also made improvements to the conduction process of the experiment by not using a Rasch model.

3.1.4 Confusion of Novices and Other Studies

The confusion of novice programmers has also been researched by Lee et al. [22]. Through analyzing the grades of students' midterm examination, they studied the relationship between novices' confusion and achievement when learning Java. During their experiment, compilation logs of students in four practical lab exercises were collected. Afterwards, part of the data was labeled manually as Confused, Not Confused, or Bad Clip. A model was built using this labeled data to label the rest of the data automatically. As a result, Lee et al. found that prolonged confusion has a negative influence on students' achievement. However, resolved confusion has a positive effect on students' achievement.

There are other researchers who have tried to find the most problematic concepts for novices [36]. The relationship between boredom and learning achievements has

been done [7, 34].

3.2 Learning Transfer Study

Learning transfer has been a popular research topic for years. Various studies have been carried out discovering details of the knowledge transfer procedure. In this section, transfer of learning in computer science (CS) and PL transfer of novices are the main focus of discussion.

3.2.1 Transfer of Learning in CS

Garcia-Martinez and Zingaro did research on why CS students often fail to transfer learning between context [12]. The most frequent teaching styles in CS can be divided into five categories: "teacher as the isolated authority delivering a subject, teacher as the authority delivering a course, teacher as a member of a learning community, teacher as the facilitator of students' learning, and teacher as a facilitator of a learner centered environment" [49]. Garcia-Martinez and Zingaro found that learning transfer exists in all these categories, and these teaching styles can all be improved if learning transfer is paid special attention to [12].

Izu and Mirolo explored the learning transfer of novice programmers by analyzing the results of two related coding tasks in C PL [15]. Among their participants, 255 CS1 students, 36.5% transferred their knowledge from practices to the task solving successfully, while 56% tried to make the transition. 13% of students partially did the transfer, and 6% of them failed to transfer. 38% of students failed to reuse the valid strategy in task one or implement a better strategy, and were unsuccessful answering task two. What's more, 9% of students thought of a different strategy, which indicated that there were additional learning happening in between the two tasks. During analyzing the data, Izu and Mirolo identified four types of transfer (Extended transfer, Consolidation, Partial transfer, and Failed transfer) and two types of non-transfer (No transfer, and New insight) [15]. They also found that by carrying out peer review of important coding tasks, weaker students were able to read distinct design strategies and were easier to make the transfer.

3.2.2 Programming Language Transfer of Experienced Programmers

Experienced programmers have more knowledge and experience in programming, know more PLs, and are more familiar with concepts in PLs than novices. They are believed to learn new PLs much faster than novices. Nevertheless, the PL transfer procedure can still be hard because of large amount of differences between PLs [37]. The problems or misunderstandings of experienced programmers are also likely to be difficult to answer or explain. Whereas, due to the negative influence of transfer

decreases as the experience of learners grows [28], not so many studies were done recently about experienced programmers.

Transfer Tutor [37] is a research tool created by Shrestha that teaches programmers R using Python and the data analysis library named Pandas. It is used by Shrestha in investigating PL teaching considering learning transfer. The figure below (Figure 3.1) presents details of Transfer Tutor. Two lines of code are shown at

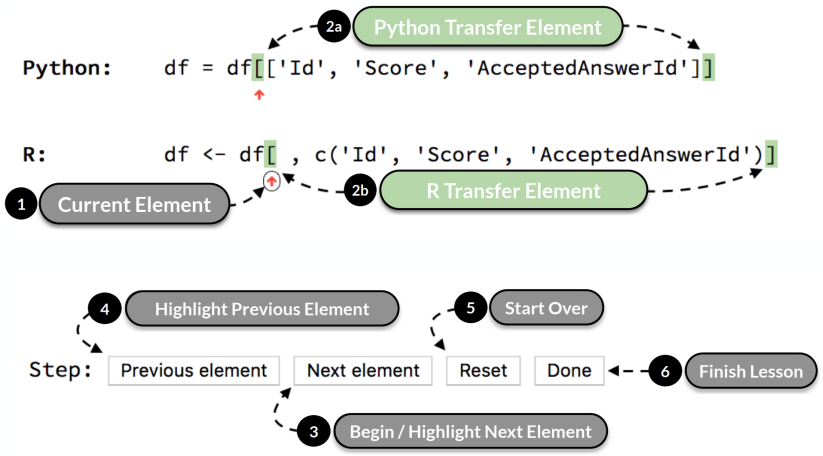

Figure 3.1: Example of Transfer Tutor [37]

the same time with Python code on the top of R code. The red arrow (1) points at the currently highlighted syntax element of both lines, which are (2a) and (2b) in this case. Steps at the bottom are buttons to control the tutorial. "Next element" will highlight the next syntax element, and "Previous element" does similar things but for the previous direction. "Reset" and "Done" mean resetting and finishing the tutorial respectively. Through Transfer Tutor, similarities between syntax elements are highlighted, and potential misconceptions are explained. The results of Shrestha showed that experienced programmers used PL transfer even without guidance, and the knowledge they already had usually contributed to their learning [37].

3.2.3 Programming Language Transfer of Novices

Because of trouble caused by negative transfer typically occurs only in the beginning stages of learning [28], PL transfer has a bigger impact on novices. In recent years, many studies has been done in this area.

Block-based programming is a programming paradigm that utilizes visual elements, often represented as blocks, to create code. Users connect these blocks to program instead of typing out lines of text-based code. This paradigm is particularly popular in teaching coding to beginners. However, learning to program in text-based code is still significant for novice programmers. Transfer from block-based programming to text-based programming can be difficult to novices. The most basic

problems faced by them during this transfer are: "readability, memorisation of commands, memorisation of syntax, typing or spelling, number of commands, prototype versus definition, matching identifiers, grouping, writing expressions, understanding types, interpreting error messages, managing layout, and changing programming paradigm [20]. To solve the aforementioned problems, frame-based editing was proposed by Kölling et al. [20]. It combines many helpful features of block-based and text-based systems into one interface. The aim of it is to save the small-scale readability, discoverability, and some of the error avoidance of blocks, at the same time, keeping the large-scale readability, flexibility, operation efficiency, and keyboard control of texts [20]. By using frame-based editing as an intermediate step, the most fundamental problems are overcame [20].

There are other studies exploring the transfer process from block-based to text-based programming. Weintrop et al. investigated the situation when students divided into three groups were using a block-based, a text-based, or a hybrid blocks/text programming environment [52]. After the introductory course using the environments mentioned above for five weeks, the performance of students in traditional Java course was examined. They found that students using text-based programming environment for introductory course often had the lowest average number of successful compilations. Students in the hybrid blocks/text environment had the highest number of successful compilations. Besides, students in blocks environment had least errors in half of the ten most frequently seen errors because they paid more attention to the differences between syntax of Java programs and block-based programs. This was caused by students finding huge difference between the two programming environments. Weintrop and Wilensky developed commutative assessments to understand how block-based differ from text-based programming in regarding to conceptual understanding [53]. Powers et al. explored a block-based programming system, Alice, and found that the object model in it leads to misunderstanding easily [30]. Even though students didn't make much syntax errors when programming in Alice, however, it can have negative influence when they transfer to Java [30].

Since syntax is still a significant problem to novice programmers when learning programming, an empirical study was done by Stefik and Siebert about syntax of PLs [43]. Two surveys were conducted to discover what words and symbols could be easy for novices to understand. Afterwards, two other studies on six PLs (Java, Perl, Ruby, Python, Randomo, and Quorum) were done to analyze the accuracy rates of novices. As a result, they found that PLs (Java and Perl) share more syntactic constructs as C caused more trouble for novices. These languages did not improve the performance of students comparing to other PLs, while a language with randomly generated keywords, such as Python, Ruby, and Quorum, contributed to higher accuracy rates to novices [43].

Tshukudu and Cutts did research on four novices and one experts from five universities transferring from procedural Python to object-oriented Java in ten weeks [47]. Participants were interviewed individually during the experiment. According to their paper, session 1 was carried out before participants learning Java, session 2

took place in the second week of learning Java, and session 4 was in the sixth week of Java. Tshukudu and Cutts found that participants faced little difficulty with carryover concepts that provide participants with positive effect on the semantic transfer. For example, variables, conditional-statements, methods, and parameter passing are all carryover concepts. Participants understood these concepts correctly in Java. However, they had trouble mapping changed concepts, which occurred when syntax were matching but corresponding semantic were distinct. For those concepts that do not exist in Python, participants could not transfer their knowledge. Their findings reflected that novices "relied mostly on their syntactic matching between Python and Java and subsequent semantic transfer" [47].

4 Methodology

In this thesis, a qualitative study was done to explore the learning transfer of novice programmers when they transfer from C and other PLs to Java. The experiment took place at the very first data structure class of the semester before novices starting to learn Java. The questionnaire used only contained Java comprehension tasks to observe the intuitive behavior of participants when they facing a completely new PL. In this chapter, details about the methodologies used in the thesis are described, which include research objects (section 4.1), participants (section 4.2), materials used in the experiment (section 4.3), and experiment process (section 4.4).

4.1 Research Objects

Before reaching to contents directly related to the experiment, variables of RQ, RQs, and hypotheses of the thesis are introduced.

4.1.1 Variables

Variables are key elements in formulating research questions and hypotheses, designing experiments, and analyzing data. Independent variables and dependent variables are two types of important variables in empirical studies. Independent variables are those controlled by researchers, while dependent variables are measured by researchers. Dependent variables are the outcomes or responses that are influenced by the independent variables.

The independent variable of this thesis is the knowledge background of novices. There are mainly two categories of background: having only knowledge in C, and having knowledge in other PLs. The first category means that novices only have programming knowledge in C. The second category stands for having programming knowledge in various PLs. The reason why this category was not specified clearly is that the detailed information of participants remained unknown when designing the research questions. Participants are novices, but they could have learned Python, C, Java, or the combination of these languages. When carrying out the study, the experience of participants would be collected through a questionnaire about self-assessed experience created by a supervisor.

The dependent variable is the transfer observed during Java code comprehension. Different types of transfer were expected to be observed, which can be divided into mainly three categories. The ideas of these categories were derived from the work of Tshukudu and Cutts [48], and also were effected by the research of Perkins et al.

[28]. For questions contain concepts in TCC, FCC, ATCC category, positive transfer, negative transfer, no transfer were expected respectively. During the experiment, a Java questionnaire was handed to participants. All the questions in the questionnaire asked participants to write the results of the given code snippets or write the possible error of the snippets if they believed it was not correct. Afterwards, the results were collected, and the transfer was observed through analyzing the results.

4.1.2 Research Questions

To investigate what transfer would be observed in novices of different programming backgrounds, the research questions were designed as follows. The first RQ is

- **RQ1: "What kind of transfer can be observed in novice programmers during Java code comprehension if they only have programming knowledge in C?"**

Described in this RQ, the aim of the thesis was to find out phenomena of transfers of novice programmers who only have experience learning C when comprehending Java code snippets. The second RQ is

- **RQ2: "What kind of transfer can be observed in novice programmers during Java code comprehension if they have programming knowledge in various programming languages?"**

The other research goal of the thesis is to find what kind of transfer would be experienced by novices with other programming backgrounds.

4.1.3 Hypotheses

Hypotheses are predictions about the outcomes of a research study. They are the basis of empirical studies and guide the research procedure by providing a clear and testable framework for investing relationships, or effects. The hypotheses of this thesis are listed here.

Considering both RQs, there are three hypotheses related to the three categories of concepts [48].

- **H1: For novice programmers having different programming knowledge, positive transfer is expected to be observed when they comprehend Java code snippets containing TCC of their learned PL.**
- **H2: For novice programmers having different programming knowledge, negative transfer is expected to be observed when they comprehend Java code snippets containing FCC of their learned PL.**
- **H3: For novice programmers having different programming knowledge, no transfer is expected to be observed when they comprehend Java code snippets containing ATCC of their learned PL.**

Because moving from procedural programming to object-oriented programming can be difficult [27], a hypothesis was obtained related only to the first RQ (RQ1).

- **H4: For novice programmers who have only programming knowledge in C, the most frequently observed transfer is negative transfer due to the differences in program paradigms and syntax.**

Students in a programming class often may have very different experience levels [21]. This may lead to different performances during language transferring.

- **H5: Novice programmers who exposed to more PLs are more likely to transfer their previous knowledge successfully to new enarios.**

4.2 Participants

The participants of this thesis were 104 university students in TUC that took the course Data Structures in summer semester 2023. Most of the students only have learned C in their first semester, while some students have learned Python before. They were taking the Data Structures course to learn coding in Java and data structures of Java. All of the students are novice programmers who are new to programming. The details of the programming experience of them were examined through self-assessment questionnaire on programming proficiency.

4.3 Materials and Tasks

In this section, two questionnaire used in the experiment are introduced, which include contents of the experience questionnaire and details of the design of Java code comprehension questions.

4.3.1 Experience Questionnaire

The experience questionnaire used in this thesis was created by a supervisor in professorship of software engineering in TUC. The questionnaire used self estimation to measure programming experience, which was proved to be a reliable way by Siegmund et al. [39].

In the questionnaire, a UID was first generated by a set of questions: last digit of student number, first letter of first name, first letter of last name, last two letters of place of birth, first digit of date of birth, last digit of date of birth, and last digit of height. This UID was unique to each participant, and was used in both questionnaires to protect the safety and privacy of user data, and the user data was anonymous to researchers. After generating the UID, it was sent to participants via emails. Following UID related questions, there were a section of questions about personal information, such as the age, the gender, and the degree of participants. Then a group of programming experience questions were presented.

The parts in this experience questionnaire related to the thesis are UID questions, and three questions about the experience in C, Java, and Python. For example, the question of Java experience was "How experienced are you in using the following programming languages? [Java]" There were five options that participants could choose from: very inexperienced, inexperienced, medium, experienced, very experienced.

4.3.2 Java Questionnaire

To explore the intuitive transfer phenomena happening in novices when they comprehend Java code snippets without starting to learn it, a questionnaire containing basic Java concepts was created. Some questions used in the thesis were adapted from research of Tshukudu [46].

At the beginning of the questionnaire, an introduction to the experiment was included. The introduction briefly listed the goal and contents of the study, and what participants needed to do during the experiment. Participants were told that they did not need to have knowledge about programming in Java, and the goal was to understand their initial expectations about how the provided Java snippets work. They were asked to try as much as they could to link their previous knowledge of C to comprehending Java code snippets.

There were 11 questions in the questionnaire. The first question asked students to fill in the UID generated in the experience questionnaire, and the next ten of them were Java comprehension questions. For each question, participants were required to read the given code snippets carefully and gave the first guess of what the code did. They were also informed that there would be errors in the snippets. If a participant could not make a guess, he or she was allowed to write simply "I do not know." to skip the question.

The ten code comprehension questions contain basic Java concepts, such as declaration of variables, declaration of objects, usage of methods. Below is the figure (see at Figure 4.1) of the first question. In this snippet, an integer variable named "a" was first declared. Then, it was assigned twice with value 1 and 10.5. The last line printed the value of the variable "a". The result of this question should be error when compiling, because a float number was assigned to an integer. Comparing to C, it contains concepts in the category FCC, since it would only result in a warning when compiling and output 10 in C. Comparing to Python, the snippets contains concepts in FCC. Students in C and Python background were expected to have wrong answer.

The second question (see at Figure 4.2) was about for-loop in Java. The variable "var" in question 2 was defined and initialised inside the loop. The last line temped to print the value of "var" outside the loop, therefore, the actual result of this snippet is error when compiling. In C, the for-loop and the scope of variables concepts have the same syntax and semantics as in Java, so the question contains TCC. In Python, the last line is able to print the value of "var" again as "Hello", thus the question contains FCC. Students have knowledge in C would point out the compiling error, but students have Python background would say that there was no error and three

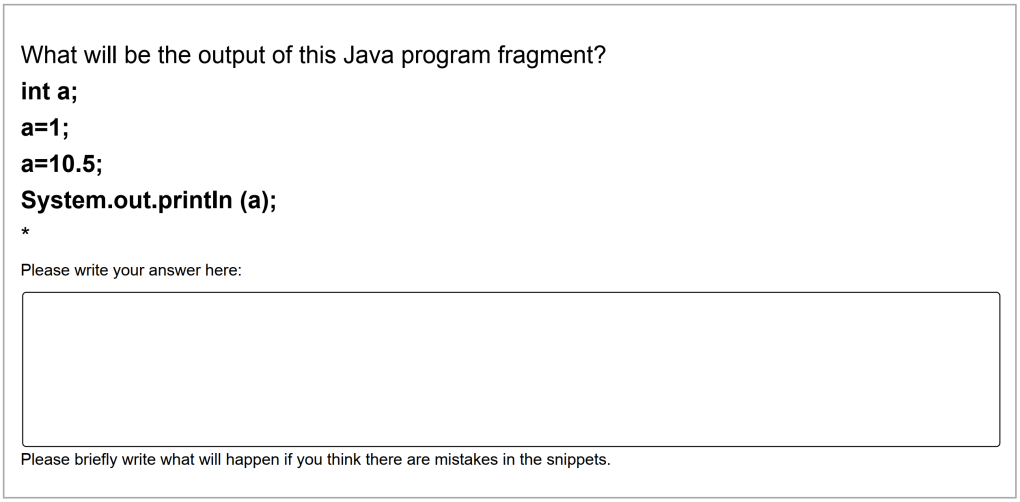

Figure 4.1: Screenshot of Question 1 - Variable Type

```
for (int i = 0; i < 2; i++) {
  String var = "Hello";
  System.out.println(var);
}
System.out.println(var);
```

Figure 4.2: Snippet of Question 2 - Variable Scope

”Hello” would be outputted.

The third question presented a simple while-loop (see at Figure 4.3). In this snippet, the value of variable ”i” was printed in each iteration. The correct answer of the output is ”0 1”. In both C and Python, the concept of while-loop has similar syntax and shares the same semantics, for that reason, this question contains TCC comparing to both PLs. The expected results from participants are also ”0 1”.

In the forth question, the concepts of object and ”==” were covered (see at Figure 4.4). In the first line, a String type object was created and was assigned with value ”lab”. In the second line, another String type object was defined and has the same value as the first object. In the next line, the two objects were compared using ”==”, and the result was printed out. In this example the == operator compared object references, not the content of the strings. Therefore, the result was supposed to be ”false”. If strcmp() function in C is used to compare two strings, ”0” will be the output when two strings share the same content. ”True” is also commonly used even though C does not support boolean data type directly. For that reason, students with C experience would say that the output of this program is ”0” or ”True”. In this case, the question is assumed to have concepts in FCC category. In

```
int i = 0;
while (i < 2) {
  System.out.println(i);
  i++;
}
```

Figure 4.3: Snippet of Question 3 - While Loop

```
String a=new String ("lab");
String b=new String ("lab");
System.out.println(a==b);
```

Figure 4.4: Snippet of Question 4 - String Comparison

the situation of Python, if two string variables, a and b, are declared and assigned by the same string, the output of `print(a == b)` is "True". Thus, the question also contain concepts in FCC. For students with Python knowledge, they would answer "True" to this question.

Concept of methods is included in the fifth question (see at Figure 4.5). This

```
public class Main{
  public static int gen(int g, int s) {
    int a = g * 2;
    return a;
  }
  public static void main(String[] args) {
    System.out.println(gen(7,3));
  }
}
```

Figure 4.5: Snippet of Question 5 - Method Calling

was a program that includes two methods named "gen" and "main". Students with programming experience in C and Python might get confused at first because of keywords such as "public, class, static", etc. Whereas, if they read the code ignoring the parts they were not familiar with, students would find that there was a "function" defined first named "gen". In the "main function", "gen" was called, and two parameters were passed to "gen". The result of this snippet was 14. The

concept of methods has similar syntax as functions in C and Python, and has the same semantics in these PLs. Thus, in this question, the concepts contained belong to category TCC. The answers from all the students were expected to be 14 which was correct.

The sixth question presented concepts of classes, methods, and objects (see at Figure 4.6). This Java snippet defined a Robot class with a constructor and a

```java
public class Robot {
  String label;
  int num;
  public Robot(String n, int w) {
    this.label = n;
    this.num = w;
  }
  public void agga() {
    num = num + 2;
    System.out.println(num);
  }

  public static void main(String []args) {
    Robot n1 = new Robot("Nori", 51);
    Robot n2 = new Robot("Alen", 22);
    System.out.println(n1.num);
    n2 = n1;
    n1.agga();
    System.out.println(n2.num);
  }
}
```

Figure 4.6: Snippet of Question 6 - Object Reference Assignment

method named ”agga”. In the main method, two instances of the Robot class, n1 and n2, were created, and their properties were manipulated. The line ”n2 = n1” made n2 point directly to the same location as n1. Afterwards, when n1 was changed later, the value of n2 was changed as well. So the correct output of this snippet is ”51 53 53”. However, to students only with knowledge in C, they believed that ”n2 = n1” only copied the value of n1 to n2. Variables n1 and n2 were still stored in different locations in memory. In this case, when the value of n1 was changed by calling the ”agga function”, the value stored in n2 would not be affected. The answer from these students would be ”51 53 51”. Comparing to C, this question contains concepts in FCC category. To students with Python experience, the behavior is

similar to Java in terms of object references. The n1 and n2 variables reference the same object after the assignment n2 = n1. When modifying the object through n1, the changes are reflected when accessing the object through n2. Therefore, the concepts in this question are in TCC category, and these students were expected to answer it successfully.

The seventh question was about memory allocation mainly (see at Figure 4.7). The first line of method twoSum contained a property of arrays and strings that

```
public class Solution {
  public int[] twoSum(int[] nums, int target) {
    int n = nums.length;
    for (int i = 0; i < n - 1; ++i) {
      for (int j = i + 1; j < n; ++j) {
        if (nums[i] == target - nums[j]) {
          return new int[]{i, j};
        }
      }
    }
    return new int[0];
  }

  public static void main(String[] args) {
    int[] numbers = {1,2,3,4,5,6};
    int target1 = 11;
    Solution s1 = new Solution();
    System.out.println(s1.twoSum(numbers, target1)[0]);
    System.out.println(s1.twoSum(numbers, target1)[1]);
  }
}
```

Figure 4.7: Snippet of Question 7 - Memory Allocation

returns the length of the array or the string in Java called ".length". There is similar usage in C that accesses a length property, but it can only be used when there is a structure that contains a member named "length". Hence, the first problem students with C programming experience met was the understanding of ".length". In the following part of the questionnaire, there was another question focusing on the usage of ".length". In method "twoSum" of this program, a new array is return in the end. In Java, keyword "new" is used to allocate memory for this array. Whereas, in C the allocation of memory needed to be managed manually by using functions like "malloc". Students have background in C would assume that there was no

memory allocated to the returned array and led to a segmentation fault. However, the actual result is "4 5" that stands for the index of "5 and 6". Students who did not know that Java manages memory through automatic garbage collection would answer it unsuccessfully. In this case, no knowledge transfer was expected, and the concepts belong to ATCC category. In python, automatic memory management is also used, meaning that it is not necessary to explicitly allocate and deallocate memory. However, there is no key words in Python similar to "new" in Java. In this case, no knowledge transfer was expected again, and the concepts also belong to ATCC category.

The eighth question was about the print line function in Java (see at Figure 4.8). "+" is allowed in Java to connect two strings, therefore, the outputs would be

```
System.out.println("hello" + "there");
System.out.println("exec" + 3);
```

Figure 4.8: Snippet of Question 8 - String Concatenation

"hellothere" and "exec3". In C, one possible way to connect two strings is to use "strcat()" function. Students with prior knowledge in C were expected to answer that this program would meet compiling error because "+" is not supported when connecting two strings. This concept belongs to the category FCC. In Python, connecting two string using "+" is allowed. Nevertheless, there would be a compiler error for running `"print("exec"+3)"`. Considering this scenario, this concept is also in the FCC category.

The ninth question was the one mentioned earlier about the usage of ".length" (see at Figure 4.9). The correct output of this program was a 3x3 array. As mentioned before, the only way that ".length" works is to have a structure that contains a member named "length" in C. For instance, there is a structure as follows (see at Figure 4.10) in C. The value of length can be accessed using "." operator or `"->"` operator. In the above example (see at Figure 4.11), we use ".length" and `"->length"` to access the length member of the structure. Looking back to the snippet of the ninth question, if students with C experience thought that ".length" only works for structures, they would answer that this program would meet a compiling error. So the concepts of this question can be classified in the FCC type, and students got wrong answers. The len() function is used in Python to get the length of a sequence, such as a list. Therefore, for students having Python experience, this concept is new.

The last question was about the class and method concepts in Java (see at Figure 4.12). This snippets presented a simple example of class and method in Java, but there is no such concept in C. Firstly, an object named student1 was created and initialized. Then, value of some attributes accessed through using methods was printed. The age attibute of the object was changed by setAge method. In the end, the changed age attribute was printed. So the correct answer was "001, Max, 25

```java
class PrintArray {
  public static void main(String[] args) {
    int[][] arr = {{1,2,3},{4,5,6},{7,8,9}};
    printArr1(arr);
  }
  public static void printArr1(int[][] arr) {
    for(int x=0; x<arr.length; x++) {
      for(int y=0; y<arr[x].length; y++) {
        System.out.print(arr[x][y]+" ");
      }
      System.out.println();
    }
  }
}
```

Figure 4.9: Snippet of Question 9 - Array Length

```c
struct SampleStruct {
  int length;
  char name[];
};
```

Figure 4.10: A Structure Example in C

23". For students with experience in C, they had no idea what classes and methods were. As a result, they might have answered it unsuccessfully, such as "I do not know.", "Compiling error", and so on. However, they might also have correct answer, because the names of methods and variables made it easier to guess the function of them. The concepts in this question were classified in ATCC category, and no transfer would be observed. In Python, there are concepts related to OOP, therefore, the syntax would look similar in python. The concepts were in TCC category, and students with Python knowledge were expected to answer it successfully.

4.4 Experimental Design

The experiment of this thesis was carried out in the first lecture of Data Structures held in summer semester 2023. Experience questionnaires and Java questionnaire offered participants both paper form and online form. For online questionnaire,

```c
struct SampleStruct structOne;
structOne.length = 10;

struct SampleStruct *ptr = &structOne;
printf("Length is: %d\n", ptr->length);
```

Figure 4.11: An Example Accessing Length of Structure in C

participants used LimeSurvey, an open-source online survey tool, to enter their responses to the questions. For the Java questionnaire, participants were allowed to enter anything with no word limitation as their answer so that they could express their thoughts thoroughly. The process of the experiment can be shown in a figure (see at Figure 4.13). This figure shows an example of the experiment process to participants who have C programming knowledge.

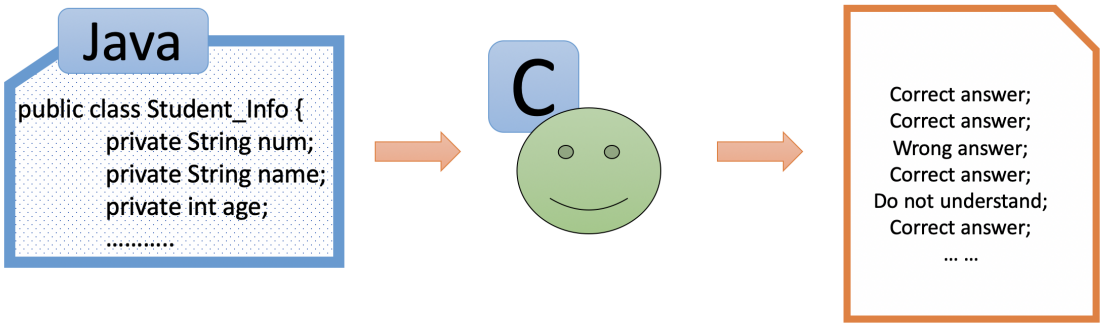

Figure 4.13: Process of Experiment for C Background Participants

During the experiment, the experience questionnaire and the Java questionnaire were distributed to students, and they were asked to fill in the questionnaire with no time limitation. Participants may open the questionnaire using their laptops, or read the questionnaire in paper form. Afterwards, participants were using their previous knowledge of programming and PLs to answer the Java questionnaire. Their answers were collected through LimeSurvey, and an excel sheet presenting the data was downloaded. This answer sheet is important for analysis of the thesis. The analysis of the correctness of the answers would be done firstly followed by categorization of mistakes made by students in the Java questionnaire. In the end, some individual analysis of special cases found among the answers would be done. The unique UID generated from the experience questionnaire were used again in Java questionnaire. This UID was later mapped to the emails of participants for

```
public class Student_Info {
  private String num;
  private String name;
  private int age;

  public Student_Info (String num,String name,int age) {
    this.num = num;
    this.name = name;
    this.age = age;
  }
  String getNum() {
    return num; }
  String getName() {
    return name; }
  int getAge() {
    return age; }
  void setAge(int age) {
    this.age = age; }
  public static void main(String[] args)  {
    Student_Info student1 = new Student_Info("001","Max",25);
    System.out.println(student1.getNum()+","
                       +student1.getName()+","+student1.getAge();
    student1.setAge(23);
    System.out.println(student1.getAge());
  }}
```

Figure 4.12: Snippet of Question 10 - OOP Related

prospective contacting purposes. The usage of UID makes it possible to collect and analyze data while maintaining the anonymity of participants, to track individual performances, to make cross-referencing with other data of future work.

5 Results

In this chapter, the results of the experiment are presented. The data preprocessing is introduced in section 5.1. In section 5.2, the results of the experiment are shown in charts. Section 5.3 is mainly about the categorization of mistakes that students made in the questionnaire, and the hypotheses are tested here.

5.1 Data Preprocessing

After participants finishing the questionnaires, the online questionnaire was collected automatically by LimeSurvey, and the paper questionnaires were collected manually. In the end, 97 online responses were collected in total, and 7 answers in paper were collected, which means a total of 104 responses were collected.

However, among these responses, not all of them can be used for analysis. There are some cases that make the response not available for leveraging. The first one is empty response. There are some responses that were collected online, with no answer at all from the very first question to the last question. These responses were considered invalid. The number of responses in this kind is 9. There is another response also be treated as invalid ones. In this response, only the first question asking to fill in the UID was answered, but the answer was 1 which made no sense. Therefore, 10 responses were regarded as invalid response, and were removed from the valid responses sheet. The second case is about duplicated answers. There were some participants who filled in more than one questionnaire. This case was found by the repetition of UID. The goal of this experiment is to find the intuitive transfer of PL knowledge when participants comprehend a new PL, nevertheless, responses from one participant should be single. For this reason, repeated responses from one participants were considered invalid as well. In this analysis, the last response from the participants were kept, and other duplicated responses were removed. In the answer sheet, 2 responses were duplicated. As a result, 92 valid responses were used to do further analysis.

The responses of the self-assessed experience questionnaire were mapped to the responses of the Java questionnaire by the unique UID. However, not all answers to the questions were used in analyzing Java questionnaire, and the used ones were answers to the three questions about programming experience in Java, C, and Python. The answers of the three questions were mapped and added as columns to the answer sheet of Java questionnaire. Among the participants of Java questionnaires, there was one participant who did not take part in answering the experience questionnaire. The answer of this participant could not be fully analyzed. Whereas, it

Java	C	Python	J&C	C&P	J&P	J&C&P	None	Sum
1	44	4	4	19	1	1	17	91

Table 5.1: Programming Experience Distribution of Participants

is still useful to look into details of this response, so it was retained for it is maximum use. There were five levels of programming experience provided as answers to the three experience questions: very inexperienced, inexperienced, medium, experienced, very experienced. As long as a participant has experience above or equal to medium level, he or she was considered have programming experience in this PL. If the experience level is very inexperienced or inexperienced, this participant was considered not having experience in this PL. A participant could have programming experience of one PL, have experience in some PLs, or have no knowledge of programming. Therefore, there are 8 types of programming experience in this research: experience in C, experience in Java, experience in Python, experience in Java and C (J & C), experience in C and Python (C & P), experience in Java and Python (J & P), experience in Java, C and Python (J & C & P), and no experience in programming (None).

5.2 Descriptive Analysis

In this section, content related to descriptive analysis is introduced. The subjects of this experiment is displayed statistically in section 5.2.1. The analysis of answers of Java questionnaire is presented in charts in section 5.2.2.

5.2.1 Description of the Subjects

The participants took part in the experiment were novice programmers having various programming background. The different types of programming background of participants are Java, C, Python, Java and C, C and Python, Java and Python, Java, C and Python, and no programming background. Except for the one participant mentioned earlier whose programming experience was unknown, the statistics of programming experience is shown in the Table 5.1. There were 68 students who have knowledge programming in C PL, 25 students who have knowledge programming in Python, and 7 students who have knowledge programming in Java.

5.2.2 Java Questionnaire Results

After receiving the answer sheet of Java questionnaire, the first thing being done was to perform the preprocessing. Afterwards, a answer sheet with all the valid data was created.

Aiming for checking the correctness of the answers and having a complete look at all the data, the answers of participants were categorized firstly into 7 types:

"correct answers", "wrong answers", "not sure", "I do not know", "not finished", "abort", and "none". "Correct answers" means that the answer was correct, but may not in exactly the same form as the actual output of the snippet. These correct answers showed that participants understood the snippets and may experience positive transfer. "Wrong answers" stands for answers that were not correct, but they sometimes contained information of how participants comprehended the code snippets. The participants wrote a wrong answer may fail to transfer their previous knowledge, or the question contained concepts in FCC or ATCC category. "Not sure" means that the participants clearly stated in their answers that they were not sure about the answer they gave. This type of answer is different from the wrong answers because the thoughts of the participants were different. The students who wrote a wrong answer may make a mistake, but he or she did not have doubts about coming up with the answer. These students who said that they were not sure about the answer were struggling from making a decision. They noticed that there was something wrong, and part of their answers could be correct. "I do not know" were answered by participants when they did not know the answer and could not make a guess. "not finished" refers to answers that specifically stated that participants were not able to finish the question. The reason why students were not able to finish the questionnaire was unknown, since there was no limitation of time used answering all the questions. "abort" stands for the answers that students wrote "Abort" or something like that. These answers show that students might not have time to finish the questionnaire similar to the "not finished" ones, the students might have no patience to answer the questionnaire anymore, or the students might not know the answer similar to the "I do not know" ones. The last type of answer is "none", which refers to blank answers that participants did not write anything for those questions.

Bar charts are a popular and effective way to present data in different categories. They are straightforward and easy to understand, and present data in a clear and simple manner. Therefore, they were chosen to conduct a comprehensive review of the valid data.

For each question, a bar chart was created to display the distribution of different types of answers. The chart of the first question is shown in Figure 5.1. The first question contained concepts in FCC category for students with C and Python programming backgrounds. According to the figure (see at Figure 5.1), the most answers to the first question are "wrong answers", which was the same as the expectation for these participants. This may show that these participants experienced negative transfer processes during comprehending the snippet. 13 participants had correct answers. One participant was not sure about his or her answer, and also one participant did not understand the snippet. Two participants left the question blank.

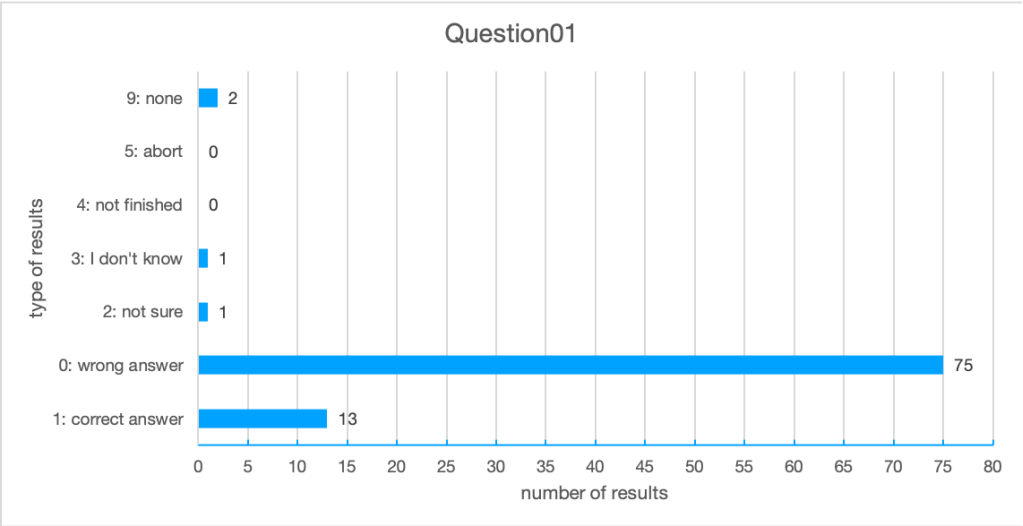

Figure 5.1: Bar Chart of Answers of Question 1 - Variable Type

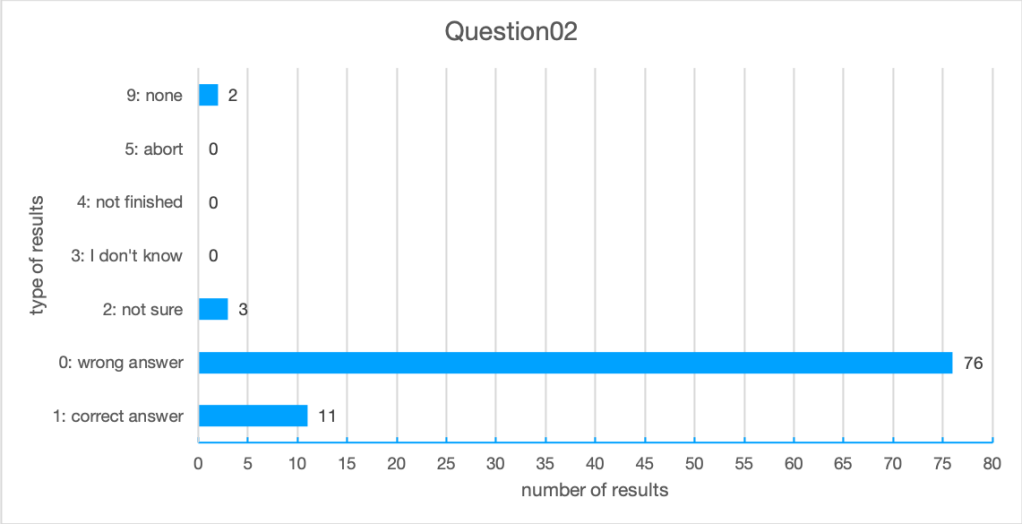

Figure 5.2: Bar Chart of Answers of Question 2 - Variable Scope

The Figure 5.2 shows the results of the second question. This question contained TCC concepts considering C PL, and FCC concepts considering Python. However, there were 76 wrong answers, and only 11 answers are correct. This is contradictory to the assumptions for students having knowledge in C. 3 students were not sure about their answers, and 2 students did not write anything for this question.

In the third question, there was only a while-loop in the snippet. The concepts are in TCC type to C and Python, therefore, more correct answers were expected. According to Figure 5.3, the most frequently seen answer is "correct answer", which is consistent with the expectation. However, there are still 29 wrong answers. Expect for 2 students did not answer the question, other types of answers were not found

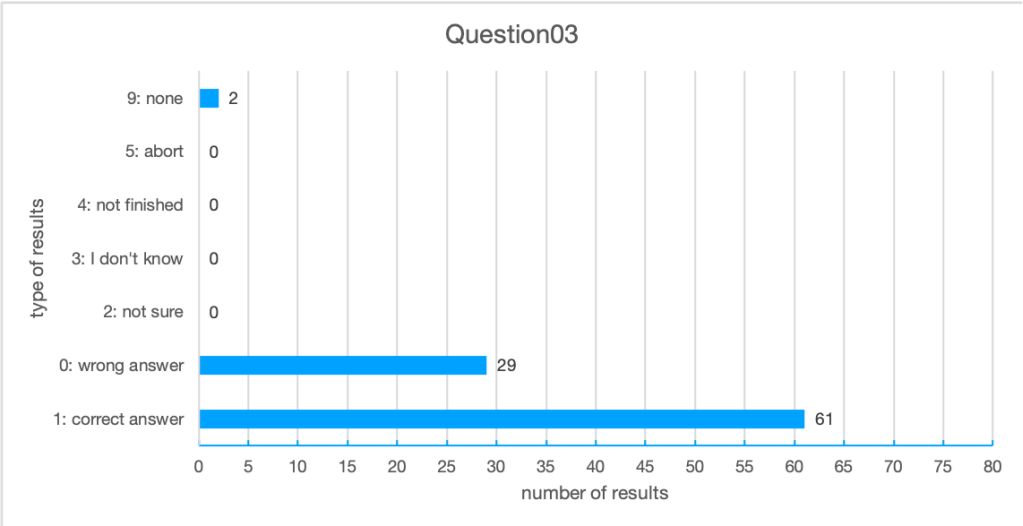

Figure 5.3: Bar Chart of Answers of Question 3 - While Loop

in responses of this question.

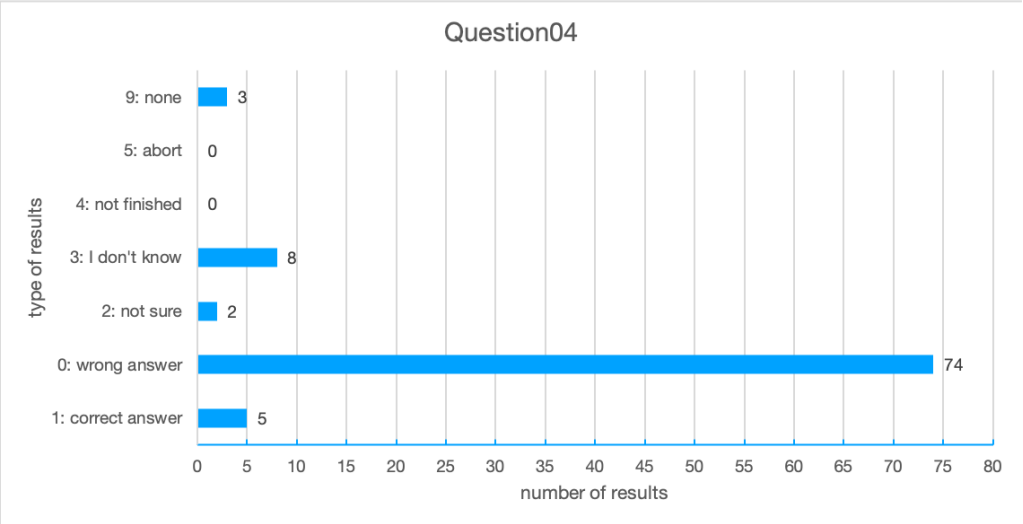

Figure 5.4: Bar Chart of Answers of Question 4 - String Comparison

There are 74 wrong answers, 5 correct answers, and 3 blank answers in the fourth question (see at Figure 5.4). 2 students were not sure about their answers. 8 students chose to answer "I do not know". This question was focus on the usage of "==", hence the concept is in the FCC category to both C and Python. Students having knowledge of C or Python might answer it wrongly. The results of this question supports this assumption.

According to the result of fifth question shown in Figure 5.5, there are 66 correct answers, 12 wrong answers, 3 "not sure" answers, 5 "I do not know" answers, and

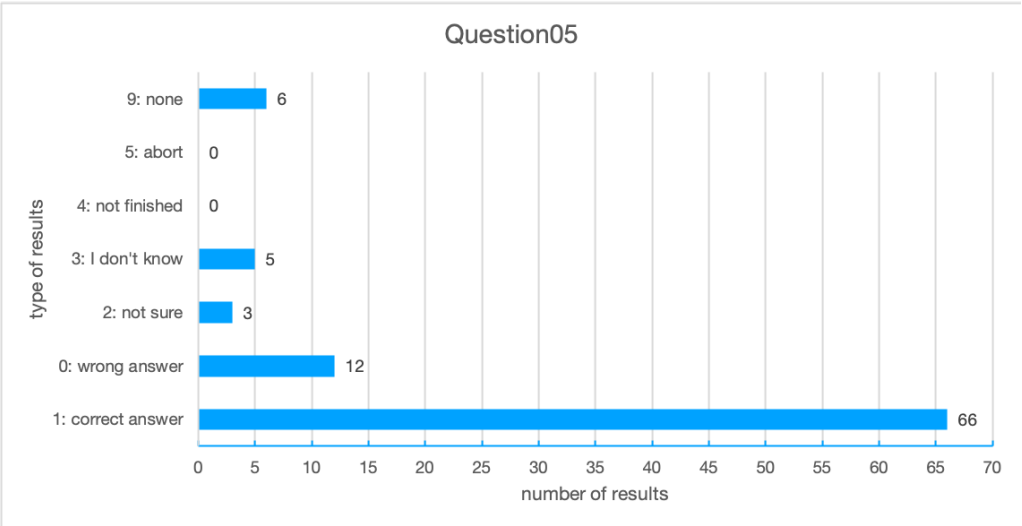

Figure 5.5: Bar Chart of Answers of Question 5 - Method Calling

6 blank answers. This question was about methods in Java, and students having different programming backgrounds were expected to have positive transfer and answer it correctly. The result supports this expectation. There is one thing that needed to be noticed: the number of students writing nothing for the result has increased from 2 to 6. This may show the decrease of students' patience for the questionnaire, or show that there are problems in the design of the questionnaire (number of questions in the questionnaire, or the difficulty of questions).

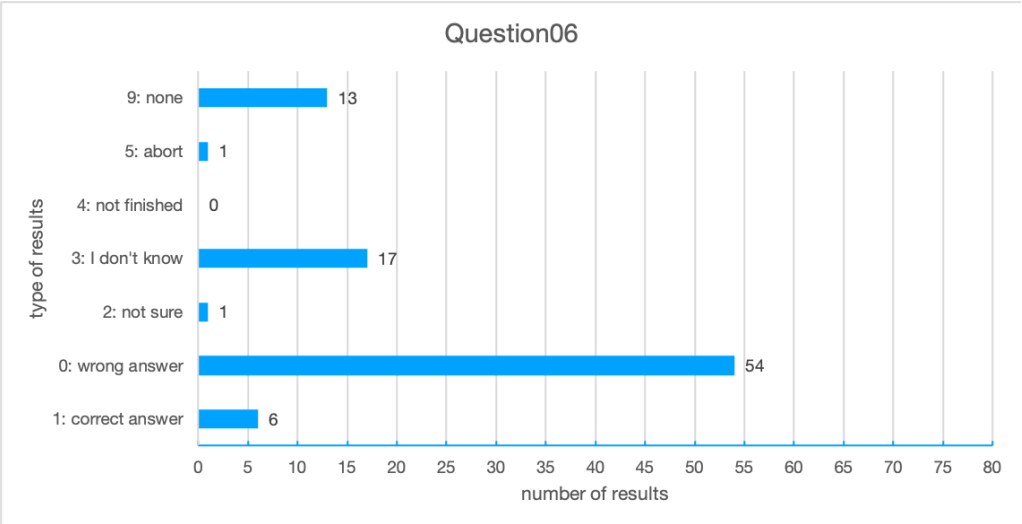

Figure 5.6: Bar Chart of Answers of Question 6 - Object Reference Assignment

The sixth question was about concepts of classes, methods, and objects. These concepts are FCC-type to C, but TCC-type to Python. Shown in Figure 5.6, there were 54 students answering it unsuccessfully, and only 6 students had correct answer.

1 student was not sure about the answer. 17 students said that they did not know the answer. 1 student clearly wrote that he or she aborted this question. 13 students did not write anything. This shows that at least 71 students faced negative transfer in comprehending this snippet, which supports the assumption for C-background students. However, for Python-background students, more analysis needed to be done.

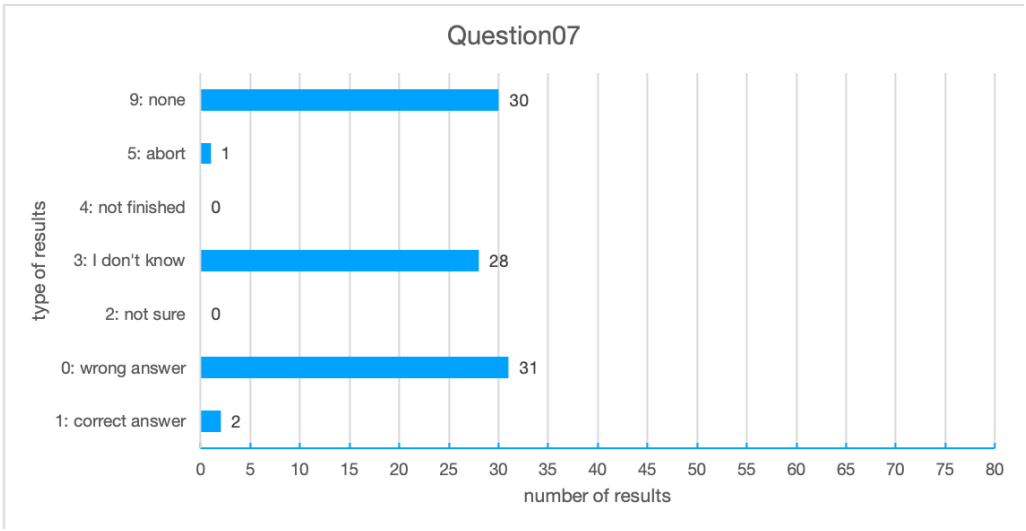

Figure 5.7: Bar Chart of Answers of Question 7 - Memory Allocation

There was an obvious increase in number of students who did not know the answer in results of seventh question (see at Figure 5.7) comparing to previous questions. The maximum number of students with "I do not know" answers was 17 for the first six questions, and for this question, the number is 28. This may be caused by the increase in difficulty of the question to novice programmers. Besides, the concept about memory allocation contained in this question belongs to ATCC category for C and Python. Therefore, students would not have any related knowledge, and no transfer was expected. The answers of "I do not know" in a way support the expectation. There are still 2 correct answers, 31 wrong answers, 1 abort answer, among the responses. The number of empty answers (30 in this case) had increased since this question as well.

There were two lines of code in the eighth question each calling the print line function in Java and outputting the connection of strings or the connection of a string and a number. The concept contained in this question is the usage of "+" when concatenating strings, or concatenating strings and numbers, which is in FCC category for C and Python. Hence, students were expected to have mostly wrong answers. According to the bar chart of the eighth question (see at Figure 5.8), 36 students answered it unsuccessfully, while 25 students had correct answers. 4 students said that they did not know the answer, and 2 students aborted to answer this question. This result supports the assumption that more students came up

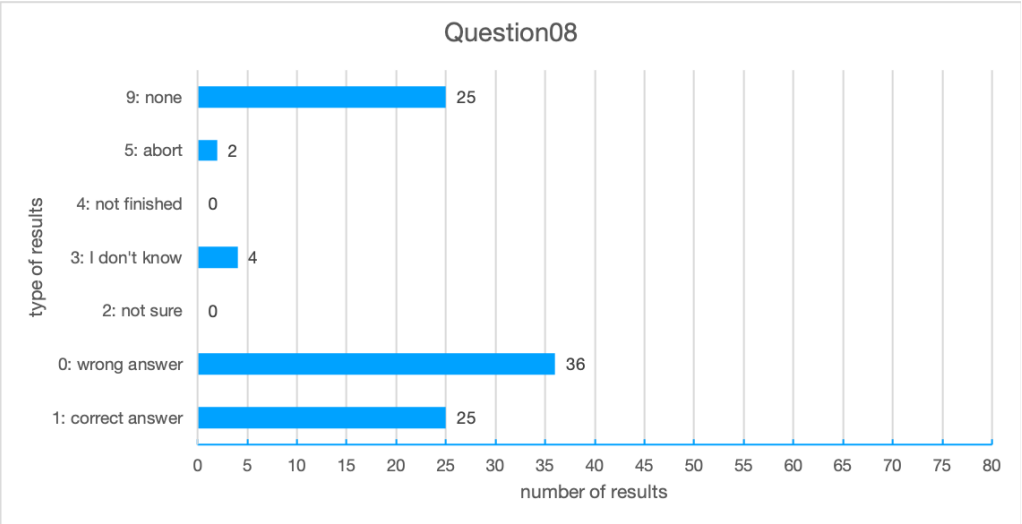

Figure 5.8: Bar Chart of Answers of Question 8 - String Concatenation

with wrong answers, and they were likely to experience negative transfers when comprehending this snippet.

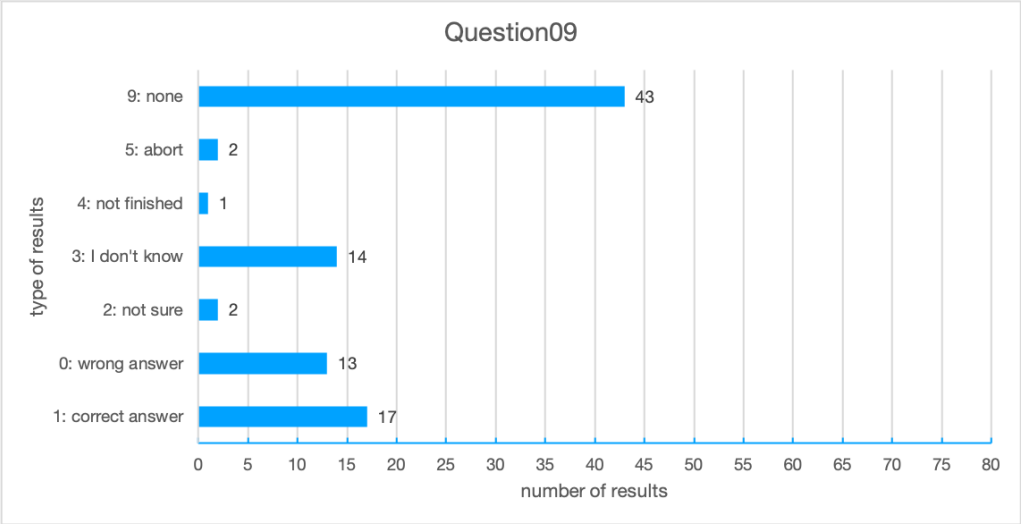

Figure 5.9: Bar Chart of Answers of Question 9 - Array Length

From the bar chart of this question (see at Figure 5.9), the number of empty answers had increased again. There was 43 blank answers. Students might loose patience to the questionnaire due to some difficult questions. This suggests that the questions in the questionnaire should present the concepts using snippets as simple as possible. There are 1 student who clearly expressed that he or she did not finish the question, while 2 students wrote "Abbruch" as the answer. Apart from these invalid answers, there are 17 correct answers, 13 wrong answers, 2 "not sure" answers, 14 "I do not know" answers. To students who have programming experience

in C, the concepts covered in this question belong to the TCC category, so more correct answers were expected. They might go through the positive transfer process when understanding the snippet. For Python-background students, the concepts are in ATCC category, and no special influence on the results was expected. They experience no transfer during comprehension.

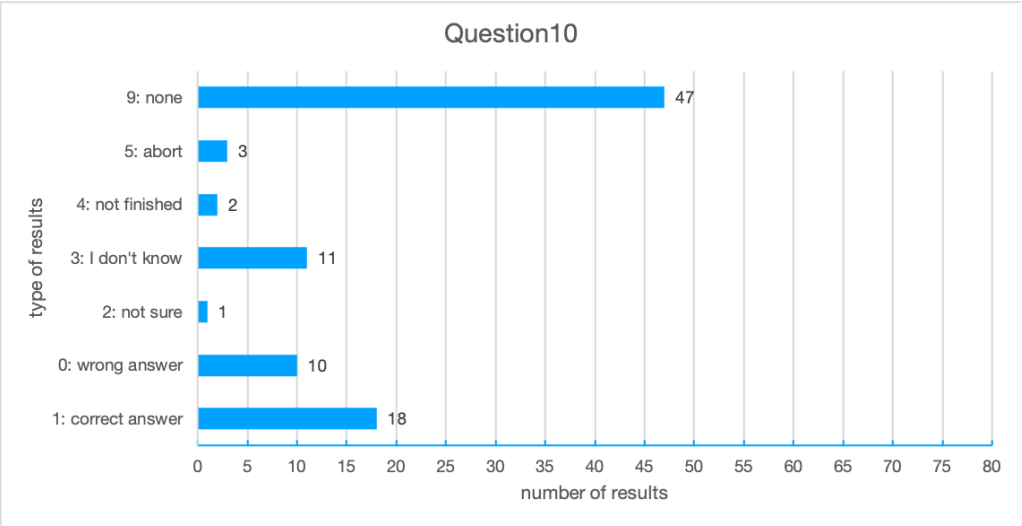

Figure 5.10: Bar Chart of Answers of Question 10 - OOP Related

The invalid answers increased again in the result of the tenth question, and almost half of the students did not answer it (see at Figure 5.10). Among the rest responses, 18 answers are correct, and 10 answers are wrong. Considering the concepts contained in the snippet, they are in ATCC category for C, and TCC category for Python. So more correct answers were expected in responses of Python-background students, and no influence was expected in responses of C-background students. The correct answers of this question is more than incorrect ones, which supports the assumption of Python-background students. However, there were only 25 students having background in Python, and not all of them had correct answers, so further analysis needed to be done. Besides correct and wrong answers, there are 1 student who was not sure about the answer, 11 students did not know the answer, 2 students stated that they did not finish the question, and 3 students aborted the question.

Taken as a whole, all the answers initially answered the research questions and basically supported the hypotheses. To novice programmers having only programming knowledge in C and those having knowledge in other PLs, positive transfer, negative transfer, and no transfer were all observed. When comprehending Java snippets containing concepts in TCC category of their learned PLs, students tended to have correct answers which could be explained as signs of positive transfer. This assumption can be verified through the results of the third and fifth question. The second question was expected to have more correct answers, however, the result showed that most students with C or Python background answered it wrongly. The

reason behind this will be discussed in the following sections. When comprehending Java snippets containing concepts in FCC category of their learned PLs, students were more likely to come up with wrong answers which could be explained as signs of negative transfer. This assumption can be proved in the first, fourth, sixth, eighth question. For Java snippets containing concepts in ATCC category of their learned PLs, students might comprehend them correctly or wrongly. This could mean that no transfer happened in this comprehension process.

The results showed that the correctness of novice programmers is strongly related to the syntactic and semantic relationship between the concept in two PLs. Even though C and Java share a lot of syntactic similarities, it is still hard for novice programmers to positively transfer their previous knowledge. The situation is the same to Python. The syntactic similarities is not as helpful as expected to novice programmers. Details of the discussion will be introduced in the following section and in the Discussion chapter.

5.3 Categorization of Results

The former section presented the results of the experiment by showing a broad way of categorizing the answers. A bar chart was created for each question to better present the statistics. In this section, a more detailed analysis was done. For each question, a series of categories were created focusing on the type of mistakes novice programmers made. By analyzing these mistakes, the intuitive knowledge transfer of novices is more thoroughly observed.

Before creating the tables, the data is processed again to remove the useless responses. As mentioned above, there are blank answers for all of the questions. They are not helpful for analyzing the relationship between the intuitive knowledge transfer and the programming knowledge background of novice programmers. Hence, all the blank answers needs to be removed from the answer sheet. In the actual data processing procedure, the removal was done by marking the related cells of answer sheet to grey color. Afterwards, responses left in the answer sheet were reviewed one by one and categorized into different groups. By analyzing the similarity between answers in the same group, a name for this type of answer was created. If the number of responses included in a group of answers is relatively small, this group would be included in the category named "other". During the categorization process, one answer from a student is classified into only one category.

5.3.1 Categories of Question 1 - Variable Type

By following the procedure described in the last paragraph, the table of the first question can be found at Table 5.2. As shown in the table, there are four categories created from the answers of participants. There were two blank responses removed, and the valid responses are 91 for this question. There are 23 answers in "Type Incompatibility" category, 50 answers in "Type Casting Misconception" category,

	Type Incompatibility	**Type Casting Misconception**	**Correct Answers**	**Other**	**Sum**
Java	0	0	1	0	1
C	8	27	5	2	42
Python	2	1	1	0	4
J & C	2	1	1	0	4
C & P	5	11	3	0	19
J & P	0	0	1	0	1
J&C&P	1	0	0	0	1
None	5	10	1	1	17
Sum	23	50	13	3	89

Table 5.2: Categorization of Results in Question 1 - Variable Type

13 correct answers, and 3 answers in "Other" category.

5.3.1.1 Review of Question

As a reminder, the first question is about variable declaration and assignment in Java. The concepts contained belong to FCC-type comparing to C, and also FCC-type comparing to Python. The snippets of question-1 can be found at Figure 4.1.

5.3.1.2 Categories Elaboration

Shown in the Table 5.2, the first category is named "Type Incompatibility". The answers included in this category are 10 and 11. Students who wrote these two answers simply did a rounding to the float number 10.5 or truncated it. They guessed that a float number can be assigned to an integer variable as long as the float number was changed into the type of integer. However, the variable "a" was declared as an integer, which usually does not change in Java. Therefore, assigning 10.5 to this integer variable "a" will result in "error when compiling". In C, if a float number is assigned to an integer variable, and if the result is printed using `"printf("%d", varName)"`, there will be no compilation error. Even though there will also be warning messages, the code still works, and the output will be 10. The float number is truncated, and only the integer part remains. Therefore, the students who answered 10 or 11 were believed to have background in C programming. The distribution of participants in this category does show that most participants have C background (69.6%).

The second category is "Type Casting Misconception", and contains the answers "10.5" and "1". The students who answered "10.5" had misconception about the type of the variable in a obvious way, while the students with the answer "1" also had doubts about the type of the variable. Those students who wrote "10.5" changed the type of variable "a" from an integer to a float number. They might believe that

the type of a variable varies depending on the value it has. The students who wrote "1" did not execute the second assignment of "a". They could have noticed that there were some problems with assigning an integer variable with a float number, but they still did not think the code would not work because of it. If a float number is assigned to a variable in Python, the type of the variable will change according to the number. This is because variables in Python are dynamically typed, which means it is not necessary to explicitly declare the type of a variable when creating it. So the answer "10.5" was believed to be the answers of Python background students. Among 25 students with Python background, 12 of them had answers in this category, which supports this assumption.

The third category is "Correct Answers". There are 13 students (14.6%) correctly answered this question without the influence of concepts in FCC type. There are 7 student with Java knowledge, however, only 3 of them answered correctly. Because they are still novices in programming, the other 4 students might be influenced by their knowledge in C. Even though half of the students only having C programming knowledge answered the question wrong, 5 of them were correct.

The last category is "Other" containing three different answers. A student said that there would be an error message because the variable "a" was assigned two values. Besides, this student had programming background only in C. The answer showed that the student still really lacked the experience in programming. One student wrote "I do not know", which means that perhaps the student also noticed the problem in the snippet, but was struggling to make a decision. The last answer in this category was poorly formed in grammar and was hard to understand. Hence, it could only be given up during analysis.

From the students' solutions to this question, the majority of them are wrong answers, which indicates the difficult students had when comprehending the code snippet. This result can be considered to support the idea in Hypothesis 2 and 4 that negative transfer would be experienced by novices when they facing FCC-type concepts.

5.3.2 Categories of Question 2 - Variable Scope

The categories of the second question can be found in the Table 5.3. Five categories were created, which include "For-loop Concept Misconception", "Code Execution Misconception", "Variable Scope Misconception", "Correct Answers", and "Influenced by C".

5.3.2.1 Review of Question

This questions contained a for-loop, and a string type variable was declared inside the loop. Outside the loop, the variable was printed again without declaration, which caused an error when compiling. This concept is in the TCC category when comparing to C, and in the FCC category when comparing to Python.

	For-Loop Concept Misconception	Code Execution Misconception	Variable Scope Misconception	Correct Answers	Coding Knowledge Missing	Sum
Java	0	0	0	1	0	1
C	4	14	16	7	1	42
Python	0	2	2	0	0	4
J & C	0	1	1	2	0	4
C & P	2	4	9	3	1	19
J & P	0	1	0	0	0	1
J&C&P	0	0	1	0	0	1
None	1	5	7	0	4	17
Sum	7	27	36	13	6	89

Table 5.3: Categorization of Results in Question 2 - Variable Scope

Considering the large number of participants having programming experience in C, there should be more correct answers. However, the results are not as expected. There are only 13 people in the category of "Correct Answers".

5.3.2.2 Categories Elaboration

The first category of mistakes is called "For-loop Concept Misconception". It contains answers such as "Hello" and "Hello Hello Hello Hello". Even though the actual output is nothing due to compilation error, there should be only two "Hello" strings printed from the loop. Therefore, students who wrote these answers did not correctly understand the condition of the for-loop. They might miscalculate the increase of variable "i", and thought only one string would be printed. They might also have problems with the "i++" part, and thought that this would cause the string to be printed for twice in one loop. This type of mistake is caused by the lack of programming knowledge, and is not related to a specific PL closely. The number of responses in this category is 7 which shows that most of the participants of this questionnaire have basic knowledge of programming and fit the definition of a novice programmer.

There are 27 participants having the "Code Execution Misconception" mistake. They wrote two "Hello" strings as the answer to this question. These participants understood the for-loop correctly, and also knew that the string variable declared inside the for-loop would not be printed outside the loop. However, they have misconception that a code can still run even if there are compilation errors. This misconception is likely to be caused again by the lack of programming knowledge. Maybe at the beginning of their learning phase, novices tend to read books or websites, instead of writing and running codes.

In the third category, "Variable Scope Misconception", there are 36 responses. The answer in this category is three "Hello" strings. These students with the answer

were believed to have Python programming background. They might think that there were two "Hello" strings printed from executing the for-loop, and one "Hello" string printed by the last line of code outside the loop. This is actually the result of running the code in Python. Python uses a "lexical scoping", where the scope of a variable is determined by its location in the source code rather than the specific block in which it is defined. Hence, in Python, the snippet will have three "Hello" strings as the output. Whereas, in Java, "block-level scoping" is used, which refers to the concept that the scope of a variable is limited to the block (a set of statements surrounded by curly braces) in which it is defined. So, only two strings will be printed if the last line of code causing the compilation error does not exist.

The fourth group is "Correct Answers". Because the concepts of For-loop and variable scoping in Java are in TCC category to C, more correct answers from students having experience in C were expected. However, there were only 7 students with C experience in this category, 2 students with Java and C experience, and 3 students with C and Python experience. The correct rate was 14.6%. The result shows the difficulty students have transferring from C to Java. There was no student with only experience in Python in this category, which shows that negative transfer was very likely to happen in the comprehension process of these students.

The last group is named "Coding Knowledge Missing" because the answers in this category could be assumed to present the influence of missing basic programming knowledge. The answers included in this group are "He" and "Hel". At first, the reason why students got these two answers was not clear. There was no way that a for-loop presented in this question (see at Figure 4.2) could lead to these strings. The possible results should be different times of "Hello" string. Then, it was found that these two strings were both part of the string "Hello". Hence, the students who wrote these answers might think that the condition of this for-loop stood for going into each letter of the string. As long as a student has basic knowledge of programming, the way to access each position of an array will be learned. Therefore, these two answers were caused by missing basic coding knowledge. The distribution of students proves this assumption, since 4 out of 6 students did not have any programming experience. However, there were still 2 students who had programming knowledge in C. This could be explained that these 2 students had seen code snippets dealing with strings in C before but did not remember clearly the way of implementing it.

The responses support the hypothesis 4 (H4) that novice programmers have difficulty transferring from C to Java due to the differences between programming paradigms. It also supports the hypothesis 2 (H2). Four students with only Python programming experience were unsuccessful when solving this problem. Eighteen other students with Python experience also answered it wrong. This shows that these students were having difficulty solving this problem, and negative transfer was observed when they comprehending code snippet containing concepts in FCC category.

5.3.3 Categories of Question 3 - While Loop

The Table 5.4 shows the categorization of mistakes in the third question. Shown

	While-Loop Concept Miscon-ception	**i++ Miscon-ception**	**While-Loop Condition Misconception**	**Correct Answers**	**Sum**
Java	0	0	0	1	1
C	12	1	2	27	42
Python	1	0	0	3	4
J & C	0	0	0	4	4
C & P	4	0	0	15	19
J & P	0	0	0	1	1
J&C&P	0	0	0	1	1
None	8	1	0	8	17
Sum	25	2	2	60	89

Table 5.4: Categorization of Results in Question 3 - While Loop

in this figure, there are four categories created from the mistakes that students made: "While-Loop Concept Misconception", "i++ Misconception", "While-Loop Condition Misconception", and "Correct Answers".

5.3.3.1 Review of Question

The code snippet in this question was about while-loop in Java. Firstly, an integer variable "i" was declared at the first line, and it was used for counting the iterations. The condition of the while-loop asked if the variable was smaller than 2. If so, the body of the loop was executed, and variable "i" was printed as well as increased by 1. The snippet can be found at Figure 4.3. When executing this snippet in Java, the result will be "0 1" without compilation errors. The concept of while-loop contained in this snippet belongs to TCC category to both C and Python, therefore, more correct answers were expected.

The actual results of this question are consistent with the expectation. Among 89 valid responses, 60 of them are correct answers.

5.3.3.2 Categories Elaboration

In the first category, there are 25 responses. The programming experience participants had with largest number of responses belong to this category is experience in C programming (48.0%). Besides these 12 participants, there are 4 other participants with programming background in C. Only one participant had knowledge in

Python. 8 of them have no knowledge of programming. The answers these participants provided were "0", "1", "2", and "3". Since the snippet contains a while-loop, the answer should at least represent a series of numbers that form outputs of a loop. However, it is clear that these answers are just single numbers which do not have the features mentioned. Therefore, these participants were assumed to not have the complete knowledge of while-loop. The result "0" could also be categorized into the second kind of mistake, because the students writing this answer might think that "i++" increased the value of variable "i" for twice. In this case, after printing the initial value of "i" (0), the value was changed to 2, and 2 did not fulfill the requirement of the condition of while-loop. The loop ended with only one output 0. However, this answer was still categorized into "While-Loop Concept Misconception" because it is hard to decide what the participants were thinking without asking them further questions. Besides, the mistakes they had comparing to those of the participants currently separated in the second category are distinct. This type of mistake, "While-Loop Concept Misconception", is assumed to be caused by the lack of programming knowledge, so it is not related to specific PLs.

In the second category named "i++ Misconception", there are 2 different responses: "0 0", and "infinite loop 0". The student with answer as "0 0" seemed to know that the loop body was only executed for twice. Whereas, due to unknown reasons, the printed value of variable "i" did not change. It is more obvious that the student who wrote "infinite loop 0" had trouble understanding "i++". The value of "i" was supposed to increase each time the loop body was executed. According to the understanding of this student, the value of "i" did not change, and the loop body was run continuously. The responses in this category were believed to be influenced by the lack of programming knowledge again, and are not related to any PL.

The third category is about condition of while-loop. There are 2 identical responses included: "0 1 2". The answer shows that the students knew about how to execute a while-loop. Nevertheless, the loop body should be executed only for twice according to the condition of the loop. In this case, after the value of "i" was changed to 2, the students still ran the loop body for once and printed the value 2. This situation demonstrates that these students did not understand the condition of the while-loop accurately. They might be less experienced in programming.

The majority of students (67.4%) answered this question correctly, which met the expectation of answers of this question. Since the concept included in this question belongs to the category TCC comparing to C and Python, positive transfer is more likely to be experienced by participants. The results obviously support this assumption and also support the hypothesis 1 (H1).

5.3.4 Categories of Question 4 - String Comparison

The Table 5.5 presented the categories created from results of the fourth question. The categories of different answers of this question are: "String Comparison Unavailable" , "Value Comparison", "== Return Value Misconception", "Usage of == Missing", "Correct Answers", and "Other". One thing to note is that the total

	String Comparison Unavailable	**Value Comparison**	**"==" Return Value Misconception**	**Usage of "==" Missing**	**Correct Answers**	**Other**	**Sum**
Java	0	0	0	0	1	0	1
C	5	20	7	3	2	5	42
Python	0	2	1	0	0	1	4
J & C	0	2	0	1	1	0	4
C & P	0	14	3	0	1	1	19
J & P	1	0	0	0	0	0	1
J&C&P	0	1	0	0	0	0	1
None	2	10	0	2	0	2	16
Sum	8	49	11	6	6	9	88

Table 5.5: Categorization of Results in Question 4 - String Comparison

number of valid answers to this question has decreased compared to the previous questions. This results from removing blank answers.

5.3.4.1 Review of Question

The fourth question focused on using "==" to compare two objects. The code snippet of it can be seen at Figure 4.4. Two string type objects were created with the same value "lab". Then, these two objects were compared using "==", and the result of the comparison was printed. In this example, the references of the two objects were compared, hence, "false" was printed in the end.

In C programming, if the content of two strings need to be compared, strcmp() function is often used, and the output of the function is 0 when two strings have the same content. The == operator is also used in C for equality comparison, and compares the values of two variables to determine if they are equal. If two integers are compared using "==", the result is "True" which is a boolean-type value. In Python programming, if two variables are assigned with the same string, the result of "a == b" is "True".

Looking at the cases in C and Python, the concept contained in this question is in FCC category for both languages. Thus, more incorrect answers might be came up with by participants. They were also more possible to meet negative transfer. To confirm the assumption, different categories are discussed here.

5.3.4.2 Categories Elaboration

The first category is named "String Comparison Unavailable", which includes 8 responses from participants. In this category, the types of response included are: "No output", "No output and compiler error", and "Error (cannot compare strings

with ==)". These participants thought that there was something wrong using "==" to compare two strings. This action would lead to compilation error, and no output would be created. The thought of them was assumed to be influenced by knowledge in C programming, because the content of two strings cannot be compared directly using this method. However, this form of comparing two strings is allowed in C, and the addresses of strings will be compared. Hence, these participants have misunderstanding about using "==" to compare two strings, but they were partially correct. The number of participants with C programming experience (5) was the highest among all programming backgrounds. This result supports the assumption of hypothesis 2 (H2).

The second category, "Value Comparison", contains the highest number of responses (55.7%). "True", "1", and "True or 1" are included in this category. "True" is a boolean value, while "1" are often used to represent positive feedback in the learning process of novice programmers. Students who wrote these answers thought that "==" was used to compare the content of the objects in this snippet. They might get this thought from their experience in Python programming. They might also have the same thought because they always used this method to compare two integers in C coding. The distribution of responses shows that a lot of students with C programming experience and Python programming experience came up with the same responses, which proves that the hypothesis 2 (H2) and hypothesis 4 (H4) are very likely to be accepted.

The third category, "== Return Value Misconception", contains answers such as "lab", and "Print both strings". The students whose answer was "lab" might think that the content of the string object would be printed if the result of comparison is equal. They also thought that this method compared the actual content of the objects, but the new problem in this case is misunderstanding of the return value of "==" comparison. For students whose answer was "Print both strings", it is possible that they believed the last line of code (`"System.out.println(a==b);"`) meant printing the content for both string objects. There is another potential explanation that they might think both string objects would be printed as long as the result of comparison was equality. These students all had mistakes comprehending the return value of "==" comparison. This case was believed to be caused by lack of programming knowledge or lack of Java programming knowledge, and is not related to a specific PL.

The next category is named "Usage of == Missing". The answers included in this category are: "Wrong expression for printout", `"new String("lab")==new String("lab")`, and "lab==lab". The students with "Wrong expression for printout" answer thought that an error existed in the format of "println" function. The students whose answer is `"new String("lab")==new String("lab")`, or "lab==lab" might have similar understanding of the snippet. They believed that the two objects on either side of the comparison operator would be replaced by the value of the objects, and the whole expression was printed by the "println" function. The difference between these two group of students is that the first group of students did not have the knowledge of creating a new object. These answers all indicate that

students whose answer is classified into this category did not have the knowledge of using "==" operator to compare something. This situation is not related to a PL, and is owing to the students' inadequate understanding of programming.

Students with correct answers (6.8%) had various programming background. There are one student with Java programming experience only, two students with C programming experience only, one student with knowledge in Java and C, and one student with knowledge in C and Python. Among these six participants, two of them had learned Java. This case provides another perspective that most participants faced problems when transferring their previous knowledge to the new context.

The last category of the results is "Other", and only answers did not provide an opinion to the question were classified into this category. These students said that they did not know the answer, or they had no idea about the answer, which indicates that they might have less programming experience. However, there are 5 students with C programming background also stated that they did not know the answer. This circumstance could be a sign that these C-background students had trouble comprehending the Java snippet.

Considering all the results, the first two categories of responses support the hypothesis (H2) that novice programmers are more likely to face difficulty when transferring their previous knowledge to a context that contains FCC-type concepts. The results also present that negative transfer is commonly seen when novices transferring from C to Java (H4).

5.3.5 Categories of Question 5 - Method Calling

The classification of results of the fifth question is shown in Table 5.6.

	Correct Answers	**Method Parameters Misconception**	**Method Creation Misconception**	**Class Knowledge Missing**	**Method Access Knowledge Missing**	**Other**	**Sum**
Java	1	0	0	0	0	0	1
C	31	3	3	0	0	5	42
Python	4	0	0	0	0	0	4
J & C	3	0	0	1	0	0	4
C & P	14	3	0	0	1	0	18
J & P	1	0	0	0	0	0	1
J&C&P	1	0	0	0	0	0	1
None	12	0	0	0	0	2	14
Sum	67	6	3	1	1	7	85

Table 5.6: Categorization of Results in Question 5 - Method Calling

5.3.5.1 Review of Question

This question is about methods in Java. The code snippet of this question can be found at Figure 4.5. Inside the main class, a method named "gen" taking in two parameters was first created. Then, in the main method, this "gen" method was called, and the return value of the method was printed using "println" method.

Even though the methods in Java have different syntactic features comparing to functions in C and Python, the overall syntax is similar. Therefore, this concept is in TCC-type for both C and Python. Participants should be able to answer this question correctly and had less difficulty understanding it.

5.3.5.2 Categories Elaboration

The number of answers (67) classified into "Correct Answers" category supports the assumption mentioned above, because 67 out of 85 students (78.8%) understood the snippet correctly. One kind of answer in this type is "14". These students passed the values of the parameters to the "gen" method, and multiplied the value of "g" by 2. The result was then returned to the "println" method, and 14 was printed. Another kind of answer is "14, (warning) s not used". These students not only calculated the correct output, but also noted that there should be a warning message. They were likely to have more programming experience, or grasped the programming knowledge thoroughly. When similar code snippets in C and Python are executed, the same process will be performed. This thinking process shows that reason why a lot of students answered it correctly is that they might have positive transfer during the comprehension process. Consequently, the students whose answer is in this category were believed to have programming experience in C and Python. This category supports the idea in hypothesis 1 (H1) that it is easier for novices to transfer their knowledge to contexts containing concepts in TCC category.

The second category is named "Methods Parameters Misconception". There are 6 students in this group, and all of them have C programming knowledge. The answers included in this category expressed that there was an error caused by not using the other parameter of the method. For example, some students said that "Error, s not used". These students thought both parameters of the method should be used. There are a lot of IDEs, such as Eclipse, suggest that all the variables declared or passed to a function should be useful to part of the following code. If a variable has no effect after being declared, a warning may appear pointing at the variable. This point has been pointed out by students in the category of correct answers. In some other IDEs, unused variables will be colored in grey. So it is considered a good practice to avoid useless parameters as intended to maintain code clarity and readability. However, in PLs like C, Python, C++, and Java, it is not strictly necessary to use all the parameters passed to a function within the function body. Besides, the decision to use or ignore specific parameters depends on the requirements and logic of the function. Hence, the students had misunderstanding about the usage of parameters of a method.

The third category, "Method Creation Misconception", contains 3 answers. The three participants in this group only had programming knowledge in C. Their answers to the question was: "Error, no gen". These students thought that there would be an error when calling the method "gen" in the main method because it was not declared. However, the method "gen" was the first thing created in the main class. Thus, these students might not know the way of creating a method in Java. They could not find the declaration of the "gen" method, even if the declaration of the method in Java has similarities to the declaration of the function in C. Even though this question was expected to be successfully solved, the syntactical details were still difficult to comprehend for novice programmers. This situation supports the hypothesis 4 (H4), and reveals that even little difference could lead to misunderstanding of novices.

In "Class Knowledge Missing" category, there are one student with Java and C programming background. The student expressed the doubts in understanding the code by saying that there were issues with how the Main Class was being created and referenced in the snippet. However, there should be no problem with the creation of Main class since this code snippet can be directly executed in a Java IDE. The answer shows that this student had learned Java programming, but maybe some knowledge of class was still missing.

The next category is named "Method Access Knowledge Missing", which has one response. The student wrote only "a" for the answer. This is the return value of the "gen" method, nevertheless, there were numbers as the value of the parameter passing to the method when it was called. Consequently, the output of the snippet should at least be a number. The answer "a" indicates that the student did not understand or did not understand correctly the access of "gen" method in the main method. This result was believed to be caused by lack of programming knowledge, whereas, the student had programming background in both C and Python. More analysis need to be done to explain this situation.

The last category is "Other". The answers classified into this category are: "I do not know", "56", and "creates a new string". The students who said that they did not know the answer had trouble comprehending the snippet because they might lack Java programming experience and cannot find out the solution. One possible way to understand the answer 56 is that the student passed the number 7 to variable "g", the number 3 to variable "a", and multiplied 7 by 2 for three times. This student is highly possible not to have programming experience, and was influenced by the two questions with loop earlier. The student with the last type of answer might also lack experience in programming.

The responses of this question are good examples of positive transfer when novice programmers dealing with concepts in TCC-type. 78.8% of the participants answered it successfully. This result increases the chance of accepting the hypothesis 1 (H1). Nevertheless, the rest part of participants still had problems understanding the snippet, which represents the influence of differences in syntax of Java, C, and Python.

5.3.6 Categories of Question 6 - Object Reference Assignment

There were 7 categories formed from the responses of the sixth question (see at Table 5.7). There are 12 out of 78 (15.4%) participants answering the question correctly, and 66 participants met different types of problems.

5.3.6.1 Review of Question

The snippet was about concepts of classes, methods, and objects, and can be found at Figure 4.6. In the snippet, firstly two variables were declared inside the class named "Robot". Then, a constructor method of Robot and a method named "agga" were created. In the main method, two objects named n1 and n2 were declared and initialized. Then, "println" method printed the "num" attribute of the object n1. After the assigning n1 to n2 and increasing the "num" attribute of n1 by calling "agga" method, the "num" attibute of n2 was printed. The actual result of this Java snippet would be "51 53 53", because the assignment made n2 point to the same address as n1. There is one thing to be noticed that the second number being printed was produced by the "agga" method.

In C programming, if similar data structures and functions were created, only the value of n1 will be passed to n2 through the assignment "n2=n1". When the "num" attribute was changed afterwards, the value stored in n2 will not be affected and remain the same. The result in this case will be "51 53 51". Therefore, the concepts contained in this snippet are in FCC category comparing to C programming language.

Python supports OOP just like Java does. Hence, similar things in the snippet can also be achieved in Python codes. The assignment has the same semantic meaning as in Java. As a result, exactly the same result will be outputted when executing the code, which is "51 53 53". The concepts are in TCC category in this case.

5.3.6.2 Categories Elaboration

The first category is "Correct Answers". The students whose answer was correct were expected to have background in Python programming. As shown in the Table 5.7, 50% of these students had Python programming knowledge, which supports the expectation and indicates that they had positive transfer during solving this question. The result also supports the hypothesis 1 (H1).

The second category is named "Assignment Misconception", which includes answers that stated "51 53 51" as the output. The output is the same as executing the similar snippet in C, hence, it was believed that more students in this group have background in C programming. The actual statistics shows that there are 8 students with only knowledge in C, 1 student with knowledge in Java and C, 3 students with knowledge in Python and C, and 1 student with knowledge in these three PLs. Therefore, the total number of students having experience in C is 13, which occupies 68.4% of the students in this group. This result supports the assumption.

	Correct Answers	Assignment Misconception	No Method Knowledge	Method Misconception	Code Execution Misconception	Assignment Not Noticed	Other	Sum
Java	1	0	0	0	0	0	0	1
C	3	8	8	1	6	0	15	41
Python	0	2	0	0	1	0	0	3
J & C	1	1	1	0	0	0	1	4
C & P	5	3	3	0	0	2	2	15
J & P	1	0	0	0	0	0	0	1
J&C&P	0	1	0	0	0	0	0	1
None	1	4	0	0	1	1	5	12
Sum	12	19	12	1	8	3	23	78

Table 5.7: Categorization of Results in Question 6 - Object Reference Assignment

These students thought that "n2=n1" only copied the values of n1 to n2, thus, the value of n2 remained the same after the value of n1 being changed. Their thoughts present the misunderstanding of the assignment between two objects, and are instances of negative transfer from C to Java.

In the third category, "No Method Knowledge", 12 responses were included, such as "51 51", "51 51 (I do not know what ".agga" does)", "prints out two objects n1 and n2", "n1=51 n2=", "Nothing, because the functions are not called". Students with "51 51" and "51 51 (I do not know what ".agga" does)" did not know what ".agga" means and only wrote the first printed number and the last printed number. If the students had learned the basic knowledge related to methods in Java, they would not have doubts about ".agga". Students who wrote "prints out two objects n1 and n2" as the answer seemed to skip the line of code calling the method. They were likely not to understand this line similar to the students with the former two answers. Besides, they did not know that "n1.num" means accessing the "num" attribute of the object n1. "n1=51 n2=" was written by students who might also skip the assignment because they could not comprehend it. Though the format of output was inconsistent with the actual result, the first output number was correct. Whereas, the second output of this student was missing. This shows that he or she had trouble from the line with the "agga" method. This student thought there was something going on with "n1.agga()", however, the last output was built upon this result and was hard to guess without knowing the meaning of the last line. "Nothing, because the functions are not called" shows that this student did not know ".agga" is used to access the method created above. Since the student called the methods in Java "functions", it is very possible that this student had background in C programming. Since methods is a unique feature of Java, this category of answers was assumed to be answers of students with C programming experience. The statics in Table 5.7 supports this assumption, because there are 8 out of 12 students with only knowledge in C. There are other 3 students with knowledge in C and Python.

The next category is "Method Misconception". One student wrote "The code is incorrect, objects Robot don't have an attribute agga". This student considered the class Robot as an object and the method "agga" as an attribute by mistake. The thought shows that this student had only knowledge in C programming, which is verified by the statistical result. The student might have heard of some features in Java, but have not learned it systematically.

The "Code Execution Misconception" category refers to the misconception in code execution order. It contains responses such as "53", "53 53", "num+2 51 51", "53 55", and "53 51". The answers whose first number is 53 are obvious examples of executing the code in a wrong order. The reason for this is that the first printed number is supposed to be the original number initializing the object n1 (51). 53 is the value of "num" of n1 after executing the method that took place after n1.num was printed. However, in the snippet, the creation of "agga" method was written before the main method, which might cause students to think that it is executed before the main method. Through analyzing the answer, "num+2 51 51", the thought of this student is clearly to see. "num+2" is the return value of "agga" method. "51"

is the value of n1.num. After assigning the value of n1 to n2, the value of n2.num becomes "51". So this student also have misunderstanding about the code execution process. Since the code execution order is a basic knowledge in programming, this category presents the lack of programming experience of novices and is not related to a specific PL.

In the sixth category, "Assignment Not Noticed", 3 responses were classified into it. These students all had the same answer: "51 53 22". They understood correctly about "n1.num" and the use of the method. Whereas, the last printed number should at least be 51 as a result of the assignment. 22 showed that these students did not notice there is one assignment before calling the method. Maybe they were not careful enough.

In the last category, "Other", there are 23 answers included. Among them, 17 students said that they did not know the answer, and they had either C programming background or no programming background. This presents the difficulty met by them when comprehending the snippet. Except for these answers, there are answers like "Robot n1=new Robot("Nori", 51); Robot n2=new Robot("Alen", 22);", "0", "22 53 51", and "51 22 53". The student whose answer is printing the first two lines of code inside the main method might not understand the whole snippet. "22 53 51" and "51 22 53" are two confusing answers. The student with the second answer might have doubts in the code execution order as well as the assignment. The first answer is similar to the answers in "Assignment Misconception" group, however, the reason why the first printed number was 22 remained unknown unfortunately. Why a student wrote "0" as the answer was unknown, but a possible explanation is that this student had huge difficulty understanding this snippet and decided to give up.

Looking at all the responses of this question in general, they demonstrate the difficulty that novices with C programming experience had during comprehending snippets containing FCC-type concepts. This situation supports the hypothesis 2 (H2) and 4 (H4). For novices with programming knowledge in Python, their answers were expected to be mostly correct due to TCC-type of concepts. Even though in the correct answer category there are 6 responses from participants with Python background, none of them is from participants with only Python background.

5.3.7 Categories of Question 7 - Memory Allocation

Table 5.8 gives a demonstration of results of the seventh question. There are 9 categories for this question. One thing should be mentioned is that the number of valid answers decreased from this question. After removing blank responses, there are 61 of them left. The category containing the highest number of answers is "Lack of Programming Knowledge" (30 out of 61 responses).

5.3.7.1 Review of Question

In the snippet of this question (see at Figure 4.7), a method named "twoSum" was created before the main method. It took an array and a integer as parameters, and

returned an array as the return value. In the main method, "twoSum" was called to find two numbers in an array that add up to the target integer. The target number in this case is 11, and the array is `"[1, 2, 3, 4, 5, 6]"`. Obviously, the only two number in the array that fulfill the requirement are 5 and 6. The array returned by "twoSum" consisted of the indexes of the found number, so `"[4, 5]"` would be returned. The two lines of "println" method at the end, printed 4 and 5 separately.

The Java concepts in this snippet are presented in `"nums.length"` and `"new int[]i, j"`. `".length"` is a technique used to access the length of an array in Java. However, there is no similar things in C. In python, the way of getting the length of an array is `"len()"`. This concept is unknown by the participants, whereas, it was not the main focus of this question. `"new int[]i, j"` allocates a space in memory for this array. Whereas, no keyword "new" is included in C and Python. In C, "malloc" function is often used to manage allocation of memory manually, which is different from and more complicated than Java. In Python, although there is automatic memory management similar to Java, the keyword "new" does not have a match.

As a consequence, the concepts contained in this snippet are in ATCC category for both C and Python, and no transfer was expected to be observed from responses of participants.

5.3.7.2 Categories Elaboration

The first category shows the correct answers. There are only 2 students (3.3%) came up with the correct answer "4 5". One of them had background in Java programming, and another had background in C and Python.

The second category is named "Array Index Misconception". The answer included in this category is "3 4". Students with this answer might understand the whole snippet, but made mistakes when dealing with the index of arrays. One of these two student wrote clearly what every step of the code snippet did, and only the result was incorrect. This indicates that these two students could have answered the question correctly.

The third category, "Array Index and Elements Confusion", consists of "5 6" and "6 5". These two answers are the numbers themselves that met the requirement. However, the correct answer should be the index of these numbers. Hence, the students in this group had a misconception about the indexes and elements of arrays, but they understood the snippet.

The next category is "For-loop Misconception". Students whose answers are in this category had problems understanding the nesting of for-loops. Their answers are "1 1", "0 0", "6 6", and "error". These answers reveal that students did not understand the for-loops inside the "twoSum" method, because the second condition of the for-loop made sure that variable j would not reach the same value as variable i (`"int j = i + 1; j < n; ++j"`). The student with answer "6 6" might understand the snippet better than the remaining students in this group.

	Correct Answers	Array Index Misconception	Array Index and Elements Confusion	For-loop Misconception	Method Access Knowledge Missing	Arrays as Parameter of Method Misconception	Lack of Programming Knowledge	++i Misunderstanding	Other	Sum
Java	1	0	0	0	0	0	0	0	0	1
C	0	0	4	2	2	4	17	1	3	33
Python	0	0	0	0	0	0	0	0	1	1
J & C	0	0	1	0	0	0	0	0	1	2
C & P	1	0	0	3	1	0	7	0	1	13
J & P	0	1	0	0	0	0	0	0	0	1
J&C&P	0	0	0	0	0	0	1	0	0	1
None	0	1	0	0	0	0	5	0	3	9
Sum	2	2	5	5	3	4	30	1	9	61

Table 5.8: Categorization of Results in Question 7 - Memory Allocation

The students who wrote "1 1" and "0 0" as the results might not know that these two for-loops traversed the elements in the array nearly twice and not understand that the goal was to find two numbers that met the demand. The student who thought there was an error stated that the second loop ran to n instead of n-2 and caused an error. The thought shows that this student did not understand the condition of the second loop as well, however, the student also explained that the first loop ran to n-1 and the second loop started 1 later, which was correct. This situation can be explained that the student did not grasp the learned programming knowledge.

The fifth category is named "Method Access Knowledge Missing", and consists of 3 responses that have the same answer: "1 2". The first line that printed the result is `"System.out.println(s1.twoSum(numbers, target1)[0]);"`. In it, s1 was the name of the object of the class Solution. In the parameter of "println" method, the method "twoSum" was called, and the array named "numbers" and target number were passed to the method. Then the first element of the returned array was printed, which should be 4. For students in this group, they had trouble understanding the access of the method "twoSum" and printed the first and second element of the array "numbers" instead.

In the "Arrays as Parameter of Method Misconception" category, responses were "no output", "No output because arrays cannot be passed without a length to a function (in C anyway).", "error because n is not defined", and "There could be a problem because the first for loop starts always with 0, I think it should start with the number in the array instead". The common part of these answers is that they all express a thought of this snippet having an error passing the array as a parameter of the method. The first two students stated that there would be no output, and the second student mentioned the length of the array was missing. The second student also stated that this conclusion was derived from the knowledge in C programming, but the misunderstanding was mostly caused by not knowing the use of ".length" in Java which was not the focus of this question. The third student said that there was an error since n was not defined. This problem is also a result of missing knowledge of accessing the length attribute of an array in Java, because the declaration of n (`"int n = nums.length;"`) was in the first line of the "twoSum" method. The last student stated that the loop should start with the number in the array instead of 0. This student seemed not notice that the array was passed as the parameter and the variable i and j in the nesting loop stood for just indexes of the array.

The seventh category is named "Lack of Programming Knowledge" because the students whose answer was classified in this group all said that they did not know the answer, or had no idea about the solution. These results demonstrate that new programming knowledge of Java was missed by students, and they had no experience of knowledge transfer during code comprehension.

The eighth category is "++i Misunderstanding". Only one student's response was included in this category. The student described that ++i and ++j were wrong in their notation, and the new string should have i = 5 and j = 6. Although i++ is often used in C programming, ++i is also allowed in the syntax of C. In the

expression i++, the current value of i is used, and then i is incremented. In the expression ++i, i is incremented first, and then the updated value of i is used. So i++ is post-increment, and ++i is pre-increment. But the description from the student indicated that he or she understood what the snippet was doing.

The last category is "Other" consisting of 9 responses. These responses were unfortunately hard to understand without asking students further questions. One answer included in this category is "123" which is the first three number of the array, but it is not related to the function of the snippet. Another answer stated that "n=6 target=11", which is correct but has nothing to do with the output. One other answer contained string "654321 11" for twice. These answers could only demonstrate that the students who wrote these answers did not understand the snippet even if they did not write "I do not know" directly.

All the mistakes participants met in this question do not have special relationship to any PL, instead, they are all because of lacking knowledge in Java or lacking programming experience. Besides, the category with the highest number of responses is "Lack of Programming Knowledge", which supports the opinion again. Hence, the statistical results present that participants had no transfer during comprehending the snippet. This case contributes to the possibility of accepting the hypothesis 3 (H3).

5.3.8 Categories of Question 8 - String Concatenation

Table 5.9 presented the categories created for the eighth question. There are 5 cate-

	Correct Answers	**"+" Concatenation Knowledge Missing**	**Strings and Numbers Connection Misconception**	**"+3" Misunderstanding**	**Other**	**Sum**
Java	1	0	0	0	0	1
C	12	1	9	10	3	35
Python	2	0	2	0	0	4
J & C	0	0	2	0	1	3
C & P	6	0	6	0	1	13
J & P	1	0	0	0	0	1
J&C&P	1	0	0	0	0	1
None	3	0	2	2	1	8
Sum	26	1	21	12	6	66

Table 5.9: Categorization of Results in Question 8 - String Concatenation

gories in the Table, and 26 out of 66 responses (39.4%) are correct. Other categories

are "No Method Knowledge", "String and Number Connection Misconception", "+3 Misunderstanding", and "Other".

5.3.8.1 Review of Question

The snippet of this question is simple. The first line printed `"hello" + "there"` using the "println" method. The second printed `"exec" + 3`. In Java, the addition symbol can be used to concatenate two strings, or a string and a number. Therefore, the actual results of running this snippet are `"hellothere"` and `"exec3"`.

Whereas, there is no such usage of the addition symbol in C programming. One of the common methods to connect two strings is using "strcat()" function. If similar snippet written in C PL is executed, compiling error will be produced. In Python, using the addition symbol to concatenate two strings is allowed, and only concatenation of strings is available. This means that connecting "exec" to the number 3 will lead to an error.

Consequently, in both C and Python, the concept of using "+" in this case belongs to the FCC category. Negative transfer is assumed to happen during participants' comprehension phase.

5.3.8.2 Categories Elaboration

In the "Correct Answers" category, there are 26 responses. The correct rate of participants with only Python background is 50%, and 34.3% for only C-background participants. For Python participants, it is easier to correctly solve the first problem. Hence, they had high correct rate.

The category named "+ Concatenation Knowledge Missing" consists of one answer that said there would be an error. This is consistent with the situation in C programming, therefore, it is believed that the student who wrote this answer have learned programming knowledge in C. The statistical result proves this assumption.

The third category, "Strings and Numbers Connection Misconception", contains 21 responses. The content of responses is different from each other, such as "hellothere no output", "hellothere error", "Will not compile, as 3 is not a string", "The program crashes because System.out.println cannot take strings + integers in its input", "Error Because we can not add numbers to strings", and "hello there Do not know what the second one does (maybe error)". Most answers contained the correct answer for the first output, but they all showed doubts on the second line of code when a number was connected to a string. Some students thought there would be no output, while some of them believed that there would be an error, the program would not compile, or it would crash. There was one student thinking that numbers could not be added to strings in calculations, which is correct, but it was not the meaning of the addition symbol here. One student said that "tring + integers" cannot be the input for "println" method, which is incorrect. The opinion presented a misunderstanding of the "println" method, but it also showed the misconception of connecting a number to a string. There were also students stating

that they did not know what the second line of code did, and a guess of the result was made saying that there would be an error. In general, students whose answers are in this category all expressed their doubts about the second line of code, and experienced negative transfer during comprehension.

The fourth category is named "+3 Misunderstanding". The answers included in this category are "hellothere exe", "hellothere exec", and "hellothere c". The students in this group did not have problem with the first output, and seemed to think that "+3" was related to printing a certain position of the string "exec". Therefore, string "exec" with different lengths were written as the output. Even though no one in this group answered the first output wrong, they were still believed to have background in C. The reason for this is that according to the categories of the answers to the previous questions, students with a C language background are more inclined to process strings in units of each character. The misunderstanding of these students also reflects the negative transfer they experienced.

The last category is "Other", and contained 6 responses. 2 students in this group said that they did not know the answer, and 2 students gave up answering the question. One student wrote "hellothere3", which is hard to interpret. Perhaps it was due to lack of programming knowledge, but more analysis needs to be done. One student described that the one in each parenthesis was reproduced. This student is correct in a way, but without explanation of the answer, it is not clear whether or not the student understood the snippet.

The answers to this question are mostly examples of negative transfer (60.6%). They support the opinion in hypothesis 2 (H2) and hypothesis 4 (H4) that novices will experience negative transfer when comprehending Java snippets containing FCC-type concepts, and it is hard to transfer from C to Java for novices.

5.3.9 Categories of Question 9 - Array Length

The categories of results of the ninth question is demonstrated in Table 5.10. The number of valid responses dropped in this question with 47 not blank ones. There are 6 categories. The number of correct answers is 15, which comprises 31.9% of the total answers.

5.3.9.1 Review of Question

The key concept in this snippet was the usage of ".length" to access the length attribute of an array. In the snippet (see at Figure 4.9), firstly, the main method was declared. Inside the main method, a two-dimensional (2D) array was declared and initialized. Then, this array was printed using a method named "printArr1". After the main method, the method, "printArr1", was created. It leveraged two nested for-loop to traverse through all elements in the array, and to print the elements row by row. The elements in one row were separated from each other by blank spaces. After running the snippet the output should be exact the same as this array:

	Correct Answers	**".length" Knowledge Missing**	**For-Loop Nesting Misconception**	**"+" Connection Knowledge Missing**	**Lack of Other Programming Knowledge**	**Sum**
Java	1	0	0	0	0	1
C	5	0	4	2	10	3	24
Python	1	0	1	0	0	0	2
J & C	0	0	0	0	0	1	1
C & P	4	2	2	0	1	1	10
J & P	1	0	0	0	0	0	1
J&C&P	1	0	0	0	0	0	1
None	2	0	1	0	3	1	7
Sum	15	2	8	2	14	6	47

Table 5.10: Categorization of Results in Question 9 - Array Length

1 2 3
4 5 6 . The approach of getting the length of the array used in the Java snippet
7 8 9
is ".length". The condition of the first for-loop was `"(int x=0; x < arr.length; x++)"`, and "arr" was the name of the 2D array. According to this condition, the value of x would increase from 0 to 2, and went through all the rows of the array. The condition of the second for-loop was `"(int y=0; y < arr[x].length; y++)"`. This loop traversed to each element in one row of the array.

In C, the for-loop has similar syntax as in Java. However, for the usage of ".length", there is no such keyword existing. The traditional way of accessing the length of an unknown array in C is to go through each element in an array and count the number. Another common approach is to use the "sizeof" operator along with the size of an individual element to calculate the length: `"length = sizeof(arr) / sizeof(arr[0]);"`. The only way that makes ".length" work is to have a data structure that contains a member named "length" in C. The example of this method can be found in section 4.3.2. After having the data structure, the value of length can be accessed using "." operator or `"->"` operator. The instance of using this method can be found at Figure 4.11. `"structOne.length"` or `"ptr->length"` are used. In this case, the the snippet could have similar syntax in C and Java. Therefore, this concept belongs to FCC category. Besides, to students with C programming background, especially to students with only C programming background, negative transfer was more possible to happen during comprehension.

Python has simpler way of getting the length attribute, whereas, length keyword still does not exist. Instead, "len()" function is used: `"length = len(list_example)"`. The parameter of "len()" function is named "list_example", which is the equivalent of an array in Python. Comparing to the ways of getting the length attribute in

C, a totally different function is used in Python. Thus, this concept is in ATCC category.

5.3.9.2 Categories Elaboration

In the "Correct Answers" category, there are still 5 answers from students with only C programming knowledge, and the correct rate for students with the same background is 20.8%. This showed the difficulty they met in this question, and proved the higher probability to accept the hypothesis 2 (H2). For students with only Python background, the correct rate is 50%, however, there were only two students in this group.

In the ".length Knowledge Missing category", two responses are included, which were both provided by students having learned C and Python programming. One of them wrote the correct answer first, but expressed the doubt about the answer by saying that maybe there would be an error because of "arr[x].length" might not be possible in Java. The thought presented that the student did not have related knowledge in Java, but could make a correct guess about the answer. The other student said that there might be problem if arr.length was not automatically determined through the variable name. This answer also means that the student did not have the knowledge of using ".length".

The third category is named "For-loop Nesting Misconception". Answers to this type vary greatly. One student wrote $\begin{matrix} 0 & 1 \\ 1 & 5 \\ 2 & 9 \end{matrix}$. Two students wrote "1 5 9". These two kinds of answers contain similar number, however, the second answer printed the numbers on the diagonal of the array. In the first answer, no regular pattern was found, which indicates that this student might not understand the nesting loop at all. One student's solution was "1,2,3 4,5,6" which only contained the first two rows of the array, and it could be considered that the student did not understand clearly about the nesting of the loop. The same problem was also found on a student printed the array for twice, a student whose result was $\begin{matrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{matrix}$, and a student with answer "11 12 13 24 25 26 37 38 39". The knowledge of nesting of for-loops is not related to any specific PL, and was the result of these students lacking programming experience.

The fourth category, "+ Connection Knowledge Missing", is about the addition symbol used in `"System.out.print(arr[x][y]+" ")"`. The purpose of adding a space after each element was to separate elements in the same row from each other. There were problems found in students' responses that they misunderstood the purpose. One student wrote the solution as "1+2+3+4+5", and another student explained the meaning of the snippet was to add each row and print out the results respectively. They thought that the addition symbol stood for printing out the symbol or adding the numbers together. This situation is likely to be the consequence

of lacking programming experience again.

The fifth category is named "Lack Programming Knowledge", and only responses of "I do not know" are included. These students met trouble during the code comprehension and came up with nothing, which represented that they experienced negative transfer or no transfer. And this case supports the hypothesis 2 and 3. Most students in this group had C programming knowledge, which made it more possible to accept the hypothesis 4.

The last category is "Other". There are 6 answers classified into it. Two students clearly stated that they aborted the question. One student said that this question was not finished, and the next question was not finished as well. One student answered 5 as the result, which did not make sense. It is found that the tenth question was also replied with 5 by the same student. Hence, it is possible that the student also gave up the question. One student with no programming background wrote that the output would be "{1,2,3},{4,5,6},{7,8,9}" which was the same as the initialization value of array arr in the snippet: `"int[][] arr = {{1,2,3},{4,5,6},{7,8,9}};"`. This student might not understand the snippet, however, the answer was correct in a way. Thus, more analysis needs to be done. A student expressed that the code could not work because the object arr and the attribute length were not created. The student might think that the array was an object, and the length was an attribute of it that should be declared before use. The knowledge background of this student is only C, so further analysis is required.

Looking at the results of this question as a whole, the 68.1% error rate proves the point that students had difficulty finding the output of the snippet. Negative transfer and no transfer were observed.

5.3.10 Categories of Question 10 - OOP Related

Table 5.11 presents the categories of the tenth question. In the table, there are 7 categories. Unfortunately, there was no responses from students who only had Python programming knowledge. There were some responses from student having knowledge in Python and other PLs, so that the analysis about students with Python background still could be carried out.

5.3.10.1 Review of Question

This question was about the class and method concepts in Java. In the snippet (see at Figure 4.12), three attributes of the class "Student_Info" (num, name, age) were declared. Then the constructor method of the class was created. Afterwards, four methods were created: "getNum", "getName", "getAge", and "setAge". By understanding the meaning of these method names, their functionality can also be revealed. Then, the main method was created. Inside the main method, an object named "student1" was created with the initialization value `"("001","Max",25)"`. The value of student1 was then printed by calling the methods created above: "student1.getNum()", "student1.getName()", "student1.getAge()". The next line

of code changed the value of the age attribute of student1 by using the "setAge" method. After this, the modified age attribute of student1 object was printed again. Then the result is "001,Max,25 23".

There is no OOP related concepts such as class and method in C programming. Therefore, students with experience in C had no idea what class and method mean. As a result, they could not think of learned knowledge when comprehending the snippet, and no knowledge transfer would happen. They were more likely to make mistakes and draw wrong assumptions. However, they might also answer the question correctly and made correct guesses, because the names of methods and variables made it easier to understand the meaning of them. No matter what the situation was, the students experienced no transfer phase due to these concepts are classified in ATCC category.

In Python, it supports OOP related concepts. Hence, the syntax of this Java snippet would look similar in Python. The concepts are in TCC category. As a result, students who had knowledge in Python programming would have no difficulty answering them, and positive transfer should take place.

5.3.10.2 Categories Elaboration

The first category is "Correct Answers", and there were 17 students in this group. The total correct rate is 38.7%. There were 7 students with only C programming background, and their correct rate of this question is 29.2%. The number of students who learned C programming is 12, so the correct rate for all the students with C programming knowledge is 34.3%. The number of students with Python knowledge is 11, and their correct rate is 54.5%. Comparing the result, the students with Python background had more possibility to answer the question correctly, this support the expectation that they would experience the positive transfer.

The second category is "No Method Knowledge", and includes three responses. Two of the students wrote "Max 25", and the other student wrote "25?". They all had some parts of the output missing. One explanation for this is that they might not fully understand the way of accessing the method in Java. The first two students only wrote the output of the first print method, which showed that they might have trouble understanding the "getNum()" and "setAge()" mainly. The problem they had with "setAge()" led to not being able to wrote the last answer. The third student added a question mark to the answer, which means that except for the outputs the student did not write, the student was also not sure about the answer given. The output of the snippet was related to methods created above, so the result of this student demonstrated that he or she had almost no knowledge in methods of Java. The students in this category were assumed to have programming knowledge in C, and the statistics is consistent with the assumption. However, there were 2 out of 3 students in this group had knowledge also in Python. The reason why they did not have positive transfer requires further analysis.

	Correct Answers	No Method Knowledge	Incorrect Output Format	"this" Knowledge Missing	"+" Connection Knowledge Missing	Lack of Programming Knowledge	Other	Sum
Java	1	0	0	0	0	0	0	1
C	7	1	4	0	0	8	4	24
Python	0	0	0	0	0	0	0	0
J & C	0	0	0	0	0	0	1	1
C & P	4	2	0	0	1	1	1	9
J & P	1	0	0	0	0	0	0	1
J&C&P	1	0	0	0	0	0	0	1
None	3	0	0	1	0	2	1	7
Sum	17	3	4	1	1	11	7	44

Table 5.11: Categorization of Results in Question 10 - OOP Related

The third category is named "Incorrect Output Format", because the responses in this category were incorrect with the format of the output but contained correct answer. The examples of responses are "001 Max 25 001 Max 23" and "23". The first answer is closer to the actual output than the second one. There were two students who wrote this result. They might not notice that in the second print method, only the age attributed was printed. The second response, however, only consists of the second output of the snippet. The students with this answer maybe could not focus on the questions anymore and missed the first print method, and the correct answer of the second output had proved that they understood the snippet. In general, students in this group could have answered the question successfully, however, due to not being careful enough, they failed to solve the problem.

""this" Knowledge Missing" is the fourth category. There is only one student in this group, and the result of this student was "001 Max 25 25". The first output was correct, whereas, the age attribute in the second output remained the same as the original value. This indicates that the student did not understand the method "setAge". The content of this method was similar to the others except for the usage of "this". The keyword "this" is a reference to the current instance of the object. "this.age" refers to the instance variable "age" of the current object. The purpose of using it is to distinguish between the instance variable and the local variable with the same name. Due to the student did not have programming background, the knowledge of using the keyword was missing.

The fifth category, ""+" Connection Knowledge Missing", consists of one answer saying that there would be an error in the snippet. The student explained that at the end of the first print method, `"+student1.getAge()"` was used to append an integer to a string without casting it, which led to an error message. The misunderstanding this student had is the same as the problem some students had in question 8. It is allowed in Java to concatenate a number to a string. The student did not have the related knowledge.

The next category is named "Lack Programming Knowledge". The responses included in it are only "I do not know", or responses that expressed the same meaning. The students had not learned Java programming yet when filling out the questionnaire, thus, it was normal for them not having programming knowledge in Java. They were assumed to have programming knowledge mostly in C only. The result in the table presented that there were 8 students having C programming experience, which is consistent with the assumption. There was no transfer observed among these students.

The last category is "Other". Four students said that they gave up on this question. One student said that the question was not finished. One other student expressed that the time ran out. Since there was no time limitation of this questionnaire, the above mentioned students might all give up answering the question. The one student left wrote that the snippet printed out three objects and 23. The answer was correct in a way, because the actual result was that three values were printed and followed by 23. The problem this response had was considering the attributes of a class as objects, which was caused by missing of Java knowledge.

The different mistakes students had in the answers prove that they had trouble understanding the snippet and figuring out the output. This situation represents that most students experienced no transfer and could not rely on the programming knowledge they had acquired, which is the same as the expectation. For students with Python programming experience, the correct rate was obviously higher, which is an example of positive transfer generated by the TCC-type concepts in the snippet.

5.4 Individual Analysis

During analyzing the mistakes students had when answering the questions, there were some special cases found. In this section, these special cases are introduced and analyzed.

5.4.1 Special Case of Three Students

Through looking at all the data collected, there were 3 participants whose responses were analyzed first. Two of them had background only in C programming and the other participant had background in Java and Python programming.

The first student had programming background only in C, and the correct rate was 0 which means that there was no correct answers. However, the student's poor performance in the questionnaire was not because he or she did not grasp the programming knowledge in C. The problem was that the student always wrote narrative sentences as the answers instead of the actual output, which made it hard to decide whether or not the answer was correct. For example, in the first question, the answer from this student was "Wiedergabe von a" which stands for "Playback of a". The actual printed variable was a, however, it was still not clear what the value of a was. The second answer from this student looked similar "Wiedergabe von var, also Hallo" that means "Playback of var, so hello". The snippet worked to print the value of var which was correct, and the string was "hello" which was also correct. However, the point was the number of the string printed, so it was not enough just saying something like that. The answer of the eighth question was "das je in der Klammer stehende wird wiedergegeben" which means "the one in each parenthesis is reproduced". This answer was mentioned earlier in the categorization analysis, and it was in the category "Other" in the eighth question. The same problem remained in this answer: the student said that the content in each "println" method would be printed, however, there was no explanation whether or not the addition symbol would also appear in the output. There were 5 questions that this student answered "I do not know", and these questions were all containing concepts in the category FCC and ATCC, which represents that the student was a ideal participant with only programming knowledge in C.

The second student also had programming background only in C. This was the second time that the student participated in the questionnaire, and the student explained that the questionnaire was not finished in the first time. In the data

pre-processing stage, the first response of this participant was removed. In the second response, thoughts analyzing what the snippet did could be found in almost all answers. This shows that this student treated the questionnaire seriously, and makes the answers from this student valuable for analysis. For instance, in the first answer the student described what the third line did and the result according to his or her knowledge in C: "The decimal value would be lost and only 10 would be outputted." This was exactly the same result that was expected from students with only C programming knowledge. The answer of the seventh question also showed typical thoughts influenced by C programming knowledge: "No output because arrays cannot be passed without a length to a function (in C anyway)." It was anticipated that students with C background would had trouble understanding the part about accessing the length of an array. Whereas, in the answer of question 6 revealed that this student might have touched some basic concept in Java before. The student said that Robot in this snippet was an object, and it did not have the attribute agga. The explanation presented that the student had misunderstanding about the object, method, and class in Java. Even though this snippet contained FCC-type concepts, the focus was about the assignment rather than the method agga. Hence, the basic knowledge in Java influenced the understanding or guessing of this student toward the snippet. The answer of question 5 also reflected the negative influence of basic Java knowledge. In this answer, the student said that "The code is incorrect because no object of the defined class is ever created, so it's attributes cannot be accessed. Besides, no attributes are ever accessed (I'd say it should be objectname.gen, not gen()." The student seemed to believe that there should always be an object of a class created to enable the access of the attributes. The method gen was considered as an attribute again.

The third student had programming background in both Java and Python. The student also wrote detailed explanations about comprehension of the snippets. Because of the Java experience, the student had a high correct rate of 70% comparing to other students. To this students all the concepts in snippets were in the TCC category, therefore, the analysis was focused on the errors this student had. The second question was answered incorrect by this student. The student only explained the part inside the for-loop as if the last line was not seen. The answer to the fourth question was incorrect, because the student thought that there were errors caused by the word "new" being used for both string "lab" and "==" being used to compare the both strings. The explanation showed the student might only have basic Java programming knowledge. The answer of question 7 was incorrect as well. However, the statement of what the snippet did had no problem at all. In the end, the student got to a conclusion that the output was "3 4", which was close to the actual answer "4 5". Whereas, the indexes of the numbers were counted wrong. This might be a result of not being careful enough when counting the indexes.

5.4.2 Confusion During Interpretation

There are some difficult parts to understand when comprehending the answers for each category, and they are analyzed here.

In question 5, there was a student whose answer was classified into "Method Access Knowledge Missing" category. The student had both C and Python programming knowledge, however, the result from this student was "a", which was confusing. After checking all the responses of the student, only one correct answer was found. This suggested that the student might not have good understanding of the acquired knowledge. One student in "Other" group of question 8 wrote "hellothere3" which looked like a combination of both outputs. The student had no programming knowledge, and the correct rate was 20%. Therefore, the reason why this student came up with this answer was not having enough programming experience. In the "Other" category of question 9, there was a student with no programming background writing "1,2,3,4,5,6,7,8,9" as the answer. The correct rate of this student was 10%, which proved that he or she did not have much experience in programming. Hence, maybe for the question 9, the student was just thinking that the snippet printed something of the array and copied the value of the array as the result. In question 10, there were two students in "No Method Knowledge" category answering the question as "Max 25". They both had experience in Python programming, thus, they should not have difficulty understanding the concepts in this question. Nevertheless, their answer was incorrect. Through looking at other results of these students, one of them had 50% correct answers, and another had 10% correct answers. The second student with lower correct rate might not have a good grasp of the learned knowledge, and the mistake was caused probably due to not fully understanding the snippet. As for the student with higher correct rate, he answered the question 5 to question 9 correctly, but the outputs of question 1 to question 4 and question 10 were wrong. This situation might show that the student did not take the questionnaire very seriously, because it did not make sense that this student had no trouble understanding more difficult snippets while failed to comprehend the easy ones.

As a conclusion for this chapter, the results collected through the questionnaire were analyzed in mainly two ways. In the next chapter, discussion over the results will be carried out.

6 Discussion

In the last chapter, the responses of the questionnaire was presented generally in ten bar charts. The classification of different types of mistakes in the responses was demonstrated in ten tables. In this chapter, the discussion over the result of the analysis is done. In section 6.1, the assessment of results is introduced, which includes the discussion about the transfer process observed, some special cases found, and the answers to the research question. In section 6.2, threat to validity is discussed from the construct validity, internal validity, and external validity. In section 6.3, the thesis is compared to some related work.

6.1 Assessment of Results

In this section, assessment of results of the questionnaire is talked about and related to the hypotheses. The main point of carrying out the research is to explore and answer the research question. Before approaching to the discussion about the research question, it is necessary to take a look again at the results first.

6.1.1 Transfer Process

In this thesis, the participants of the questionnaire had various programming experience as shown in the Table 5.1. To demonstrate the assessment of the results, three broad types of programming backgrounds are considered: C, Python and Java. The transfer processes of transferring from C to Java, and Python to Java are the main focus of this thesis.

6.1.1.1 Positive Transfer

The hypotheses of the thesis can be found at Section 4.1.3. The first hypothesis is about concepts in TCC category and is related to positive transfer:

- **H1: For novice programmers having different programming knowledge, positive transfer is expected to be observed when they comprehend Java code snippets containing TCC of their learned PL.**

Since the snippets in the questionnaire were written using Java, the concepts contained in them are all in TCC category comparing to Java itself. Therefore, to students with Java experience, only positive transfer was expected to be observed. The number of participants who have Java experience is 7 out of 91, and there was

only one student who had only experience with programming in Java. According to the analysis of all the responses, most participants only had very basic programming knowledge in Java that they did not even answer the simple questions correctly. The student with only Java background answered all the questions correctly, which indicates that this student experienced positive transfer during the code comprehension. There is one other student with Java background had 70% of correctness, which also means that the student faced mainly positive transfer. However, with the data of two students, we cannot say that we can accept the H1 in this case.

Comparing to C PL, there are three questions consisted of concepts in TCC category: Question 2, Question 3, and Question 5. Among them, Question 3 and 5 have a correct rate of students with only C background at 64.3% and 73.8% respectively. The correct rates of students having C programming experience is 71.2% and 75.4% for Question 3 and 5. These results indicate that most students with C background only or combined with knowledge in other languages experienced positive transfer during Java code comprehension, and proves that we can accept the H1 in this situation.

However, as for Question 2, there were only 16.7% of the students who had only programming experience in C answering it correctly. For all students with C language experience, the correct rate is 18.2%. The snippet contained in this question was not complicated. Shown in Figure 4.2, the snippet included a for-loop, and printed the value of a string some times inside the loop and one time outside the loop. Students with only C programming knowledge should be able to understand the for-loop and guess accurately the meaning of "println" method. Whereas, the actual result showed that they faced difficulty during the comprehension. In Table 5.3, there are 38.1% of the students with only C background having the mistake "Variable Scope Misconception", and 33.3% of them having problem in "Code Execution". This means that 38.1% of the students wrote three "Hello" strings as the answer and did not realize there should be an error when printing the variable declared inside the loop. This misconception is not caused by programming knowledge transfer because the students wrote "Hello" string as the output that showed they guessed successfully the function of "println" method. Besides, the rest part of this snippet shared the same syntax with functionally equivalent C snippet, and the concept of variable scope is taught in basic C programming courses. Therefore, we could say that "Variable Scope Misconception" was caused by lack of programming experience. There are 33.3% of students with only C programming experience having problem with "Code Execution". They thought that there would still be output when compiling error was detected during execution. This is also a basic knowledge that a student would acquire if he or she tried to execute code in a IDE. Hence, we could also say that this misconception was caused by lack of programming experience.

To sum up, the results of Question 3 and Question 5 indicate that we can accept H1 in the case when participants are novices with programming knowledge in C. Whereas, the results of Question 2 revealed that due to lack of programming knowledge, positive transfer may not appear as expected in code comprehension phase of

novices.

The snippets in Question 3, Question 5, Question 6, and Question 10 contain concepts in TCC category comparing to Python. In Question 3, 75% of the answers from students with only Python background were correct, and 80% of the answers from students with Python programming experience were correct. With this data, we could say that most students experienced positive transfer during solving the third question. Under this circumstance, we can accept H1. In Question 5, students with only Python programming experience all answered it correctly, and the correct rate is 100%. The correct rate of students with Python experience is 83.3%. In this situation, we could also say that positive transfer was observed during the code comprehension of novices, and we can accept H1.

In Question 6, the correct rate of students who had only programming experience in Python is 0 because there were no one who answered it correctly, and the correct rate of the students with Python experience is 30%. This result is inconsistent with the expectation. Therefore, more analysis was done to find out the explanation: The snippet of this question can be seen at Figure 4.6. The concept of classes, methods, and objects were included in it. For Python background students, the common mistakes were "Assignment Misconception", "No Method Knowledge", and "Assignment Not Noticed". The first mistake demonstrated that they believed "n2=n1" copied the value of n1 to n2, and the value of n2 would not be effected after the change of value of n1. However, this is what happens in C snippet, and the students were thought to have background in C programming. For the 6 students having this misconception (30% of all students with Python background), 4 of them (66.7%) had background also in C. Hence, we could say that they were effected by their knowledge in C programming. After checking the results of the rest 2 students with only Python background, they both answered incorrectly the sixth and tenth questions. These two questions involved classes and objects concepts, which are related to OOP. They also said that they had medium experience programming in Python. One possible explanation for their misconception is that they were more familiar with procedural programming paradigm, and the concepts in this question were new to them, which led to similar performances as students having C background. The second and third mistakes of these students revealed that they were influenced again by their knowledge in C, might lack programming knowledge, or were not careful enough when reading the code. According to the discussion about Question 6, we can not accept H1. Besides, it was found that students with Python and other PL experience would be affected by knowledge of other languages, which might lead to negative transfer when comprehending snippets contain TCC-type concepts. Novices with only Python programming experience might not have learned the OOP features of Python, and these features could be completely new to them. Thus, these concepts could not always be categorized into TCC category for novices.

Students with only Python programming experience all gave up on Question 10. Fortunately, there were 11 students with Python experience submitted the responses, and the correct rate for them is 54.5%. The correct rates for students with Java and

Python background, and students with Java, C, and Python background, are both 100%. It means that for these students, positive transfer occurred, which was also very likely caused by their Java programming background. For the students with C and Python background, the correct rate is 44.4%, which is lower than the overall correct rate of students with Python experience. The reason why this happened is that the concepts contained in this question are in ATCC category for C, and the students could be influenced by C knowledge.

In general, the results of question 10 imply that most students who had programming experience in Python could understand the snippet successfully and experienced positive transfer of knowledge. Besides, it shows that we could accept H1 in this scenario.

Considering the cases of students with two different programming backgrounds other than Java (C and Python), there were 5 out of 7 cases supporting the assumption in H1. We could say that for these cases of the thesis, we could accept the hypothesis 1. The observation presented that positive transfer was most often observed when novices comprehending Java snippets containing concepts in TCC category. However, due to lack of programming knowledge, positive transfer may not appear as expected for novices. Moreover, OOP concepts in Python may not be classified in TCC category for those students who only learned procedural programming in Python.

6.1.1.2 Negative Transfer

The second hypothesis is:

- **H2: For novice programmers having different programming knowledge, negative transfer is expected to be observed when they comprehend Java code snippets containing FCC of their learned PL.**

It describes the connection between FCC-type concepts and negative transfer. The analysis was done on the results of C and Python background students. The Question 1, 4, 6, 8 and 9 consisted of FCC-type concepts comparing to C, and comparing to Python, the FCC Questions are 1, 2, 4 and 8.

According to the results of Question 1, the correct rate for students with only C programming experience is 11.9%. The correct rate for students who had knowledge in C programming is 13.6%. It revealed that most C background students faced difficulty during the comprehension. The type of mistake that was influenced by novices' experience in C is "Type Incompatibility". In this category, 69.6% of the students had C programming experience. This information infers that these students experienced negative transfer in the comprehension, and supports the Hypothesis 2. The correct rate of Question 4 for students with only C experience is 4.8%, and for students with C experience, it is 6.1%. We could find that students with background in C had much trouble understanding the snippet. The categories of mistake: "String Comparison Unavailable" and "Value Comparison" were assumed to be caused by the negative transfer of knowledge in C programming. The answers of 59.5% of

the students who had only C background were classified in these categories, which means that they did have negative transfer and we could accept H2 in this case. For the responses of Question 6, the correct rate of only C background students is 7.3%, and the correct rate of students who had learned C is 14.8%. "Assignment Misconception", "No Method Knowledge", "Method Misconception" and "I do not know" were assumed to be common mistakes made by C students. From the results, among the students with only C background, 76.2% of them had the above mentioned mistakes. This finding indicates that we can accept H2 in this case. The correct rates of results of Question 8 for students with only C programming experience and students with C and other language experience are 34.3% and 36.5%, respectively. It shows that more students had problems comprehending the snippet. "+ Concatenation Knowledge Missing", "Strings and Numbers Connection Misconception" and "+3 Misunderstanding" were all believed to be problems related with knowledge in C programming. 57.1% of students who only had C programming experience had mistakes in these categories. This result demonstrates that these students had negative transfer during code comprehension, which means that we could accept H2 in this situation. The results of Question 9 also supported the assumption in H2. Firstly, the correct rates of students with only C experience and students with C experience are 20.8% and 27.8%. The low correct rate implies the obstacles met by novices. ".length Knowledge Missing category" and "Lack Programming Knowledge" were assumed to be two major problems that these students met, because they both indicated that students did not have enough knowledge of the usage of ".length" in Java which is used differently in C. There were 41.7% of students with only C experience faced problems in these category, which occupied more than half of the students with wrong answers. It shows that more students faced difficulty when reading the code snippet and experienced negative transfer, which proves that we can accept H2 in this circumstance.

According to the responses of these five questions, more students who had knowledge in C programming had trouble understanding the Java snippets containing concepts in FCC category. They were likely to go through a negative transfer phase. Therefore, we can accept H2 considering the results from students with C programming background.

There were four questions consisting of FCC-type concepts comparing to Python: Question 1, 2, 4 and 8. Looking at the results of Question 1, the correct rate of students who only had Python experience is 25%. The correct rate of students who had Python and other PLs experience is 20%. The category "Type Casting Misconception" contained the largest number of students with Python background (48% of the students). They answered the question with their knowledge of Python and changed the type of the variable dynamically. This situation represents that they were influenced by the Python knowledge, however, the knowledge was transferred negatively and led to a wrong answer. It indicates that we can accept H2 in this case. For Question 2, the correct rate of students with only Python experience is 0, and the correct rate of students with Python experience is 12.0%. There were 50% of the Python students who thought that the variable outside the loop

could be printed. In Python, because of lexical scoping, it is achievable and correct. However, there would be an error for this operation in Java. Hence, for these students they understood the snippet based on Python knowledge and got wrong answer. The situation presented the negative transfer between programming knowledge of different PLs. We could accept H2 again in this case. In Question 4, none of the students with only Python experience answered it correctly. Only 4% of the students with Python experience answered it correctly. There were 56% of the students with Python background made mistakes classified in "Value Comparison" category. They believed that only the value of these two strings would be compared, and the result should stand for equality. However, the memory location of the strings were compared in Java. The difference caused these students failing to correctly understanding the Java snippet. Hence, negative transfer was observed in this situation, and we could accept H2. In Question 8, 50% of the students who had only Python experience answered it correctly, and the correct rate for students who had Python experience is 52.6%. The rise in correct rate could be due to the reduction in code snippet length and decrease in difficulty of snippet. Although the correct rate increased, the average rate was still around 50%, which means that almost half of the students with Python experience faced problems. Among all the categories of mistakes, "Strings and Numbers Connection Misconception" was most commonly seen in Python students with a proportion of 42.1%. These students thought that it was not allowed to concatenate a string with an integer in Java because it leads to an error in Python. Their thoughts showed that they made a guess using their current knowledge, whereas, the guess was incorrect. Therefore, negative transfer was experienced by them, and we could accept H2 in this scenario.

In general, for students with Python background, they derived wrong assumptions from their previous experience when reading Java snippets containing concepts in FCC category, which represented a negative transfer. Hence, we can say that we accept H2 in this case.

6.1.1.3 No Transfer

The Hypothesis 3 is related to the concepts in ATCC category:

- **H3: For novice programmers having different programming knowledge, no transfer is expected to be observed when they comprehend Java code snippets containing ATCC of their learned PL.**

The questions containing ATCC-type concepts are Question 7 and 10 for C, and Question 7 and 9 for Python.

In Question 7, there was no students who had only programming experience in C answering it correctly. The correct rate for students who had background in C is 2.0%. The performance of these students had already shown that they met much difficulty understanding this snippet. The increase of difficulty and length of the snippet contributed to this result at the same time. More than half (51.5%) of the students with C background only and 51.0% of students with C knowledge said

that they did not know the answer, which means that they did not find similar information that they could use to help them comprehend the snippet and failed to comprehend it. There were 10.2% of C background students having "Array Index and Elements Confusion" problem, and also 10.2% of them having "For-loop Misconception" problem. These two categories of mistakes were both related to basic programming knowledge and were not connected to a specific PL. Hence, we could say that these students could not find knowledge that had relationship with the knowledge contained in this snippet, and could not use their previous programming experience during comprehension, which demonstrates that there was no transfer experienced by them. In this situation, we can accept H3. In Question 10, the correct rate for students with C experience only (29.2%) increased comparing to that of Question 7. For all the students with C experience, the correct rate is 34.4%. The number of these students who said they did not know the answer was still the largest among the numbers of students in all the categories. This situation also led us to the same assumption as Question 7 that the students could not find similar knowledge to implement. Therefore, no transfer was observed in the comprehension phase, and we can accept H3 also in this case.

Considering the results of Question 7 and 10, the students with C programming knowledge did not transfer their knowledge when understanding Java snippets containing ATCC-type concepts. They did not know that there were similar features in the language they had learned. For example, in Question 7, the keyword "new" had the same function of allocating a space for storing the variable. In Question 10, the custom data structure in C could be similar to objects in Java. Without realizing these, the learned knowledge was not helpful for the understanding of novices. No transfer was observed, and we accept H3 in the case of students having C programming background.

To analyze the Hypothesis 3 in Python scenario, the results of Question 7 and 9 were checked. For Question 7, the correct rate of students who had only Python knowledge is 0. Unfortunately, there was only one student left in this background. The answer of this student was about complaining the syntax of Java and the wish of learning Python instead. We could tell from this answer that the student had difficulty learning Java, but it was not directly related to the snippet and the question. The correct rate of students who had Python knowledge is 6.3%. It revealed that the snippet was hard for them to understand. There were 50% of them saying that they did not know the answer to this question. It demonstrates that they could not think of something similar to the keyword "new" in Python, even though Python also has automatic memory management. Hence, the students did not do any transfer from their owned knowledge to this new case, which makes it possible for us to accept H3. In Question 9, there were two students who had only Python experience, and one of them had the correct answer (50%). The other student had problem with nesting loops which is not language specific. The answer of this student was "1,2,3 4,5,6" which was the first two lines of the actual output, and it already showed that this student knew the meaning of "arr.length" was to access the length of an array. Therefore, we could say that for these two students, their learned knowledge effected

positively on their understanding. For the students who had learned Python, the correct rate is 50%. Half of them could understand the snippet correctly, which presented the positive effect of their previous programming knowledge. The mistake category that contained the largest number of answers of these students is "For-Loop Nesting Misconception". The students understood correctly that ".length" was about the length of the array, but made mistakes counting the length or had trouble understanding the nesting of loops. In this case, we could observe transfer during their comprehension. They seemed to notice that ".length" was related to the usage of "len()" function in Python. The meaning of the word "length" also contributed to their understanding, but it belonged to the positive transfer from natural language to Java snippets. Considering the mentioned aspects, we could not accept H3 in this case since transfer was observed during the comprehension.

In general, we can only accept H3 in the case of Question 7. For students with Python knowledge, no transfer was observed when they comprehending Java code snippets in Question 7. Whereas, positive effects were observed in Question 9 comprehension, which suggests to reject H3.

Considering both conditions of C-background novices and Python-background novices, we still could say that we accept H3 because the assumption in it was commonly observed. Novices usually will have no idea what knowledge could be similar to the ATCC-type of concepts and could not rely on their previous programming experience during Java code comprehension.

6.1.1.4 Transfer Related to Experience

To discuss the assumption in H4, the performance of all the students with knowledge of C was considered. The Hypothesis 4 was stated as follows:

- **H4: For novice programmers who have only programming knowledge in C, the most frequently observed transfer is negative transfer due to the differences in program paradigms and syntax.**

The participants in this questionnaire were mainly C-background students (74.7% of all participants). None of them answered all the questions correctly. Besides, most of them had less than five correct answers. Their performance presented in correctness indicates the difficulty they had during the code comprehension.

They had misconceptions in FCC-type of concepts. They noticed that the syntax of the snippets in Java, the new PL, looked familiar to them, because it also appeared in the PL they had learned. However, these concepts had different underlying semantics, which resulted in misconceptions.

For the concepts in ATCC category, they could not think of any piece of knowledge to help them understand them. The syntax looked unfamiliar, and the semantics brought more trouble to them.

It was assumed that novices could comprehend successfully the concepts in TCC category, because those concepts shared the same syntax in the new language (Java in this case) and their learned PL (C in this case). In addition, the underlying

semantics were also the same, which means that as long as they could understand the snippet written in C, they had no problem understanding the new snippet written in Java. Whereas, the results showed that novices may not be able to do this because they might make mistakes in basic programming questions. For instance, they might not understand the nesting of loops, the indexes of arrays, the condition of loops, and so on. This indicates that they would make mistakes in their learned PL regarding these basic knowledge, therefore, it was not possible for them to answer the question correctly in the new PL.

As the complexity of snippets increasing, more keywords related to OOP appeared in the snippets. This would also lead to confusion for novices who had not learned the language, or for those who had never been exposed to OOP. For example, to novices who had not learned OOP, "public" and "class" at the beginning of the snippet could look odd, and the meaning of it was unknown to them. "this" could also be ambiguous. They learned the main function in C, whereas, "public static void main" could be confusing. In the responses of the questionnaire, it was found that the number of novices saying "I do not know" increased as the snippets becoming more complex, which supports the assumption mentioned above.

In general, negative transfer was observed in novices comprehending snippets containing FCC-type concepts most frequently. Sometimes, negative transfer was also observed when they comprehending snippets containing TCC-type concepts. Novices were more likely to answer the questions consisting ATCC-type concepts incorrectly due to no transfer happening in this period.

Students in a programming class usually have very different experience levels [21], the participants in this questionnaire were in the same situation. Hypothesis 5 suggested that this phenomenon might lead to different performances during language transferring.

- **H5: Novice programmers who exposed to more PLs are more likely to transfer their previous knowledge successfully to new scenarios.**

In the previous discussion, we could find that the correct rates were usually higher of students with different backgrounds than those of students with only C or only Python background. Although the possibility for novices to transfer their knowledge successfully to new scenarios was connected to the experience growth in the new PL [28], we found that the performance of novices was similar when they had started to learn more PLs. Learning more PLs demonstrates that novices need to read and write more code snippets. They are likely to spend more time on learning programming, and get more comfortable with coding. As the saying goes, practice makes perfect. Novices who learned multiple PLs have more knowledge about at least what codes in different PLs look like. Hence, when they comprehend another new PL, they have more previous knowledge that they might rely on, which will result in better performance in the Java questionnaire. Considering the increase in actual correct rates, we can say that we accept H5.

6.1.2 Answers to the Research Questions

In this section, the research questions are answered based on the results of the questionnaire and the discussion about them. Firstly, we need to look at the research questions again at Section 4.1.2:

- **RQ1: "What kind of transfer can be observed in novice programmers during Java code comprehension if they only have programming knowledge in C?"**
- **RQ2: "What kind of transfer can be observed in novice programmers during Java code comprehension if they have programming knowledge in various programming languages?"**

Then, with the responses and creation of the categories of results, we could answer that positive transfer, negative transfer and no transfer were observed in novices during Java code comprehension when they only have programming knowledge in C. For questions containing TCC-type concepts, more students were able to answer them correctly, which indicated that they had no trouble understanding the code with their previous knowledge. Therefore, positive transfer was observed in their behaviors. Some of them could not answer the question successfully, which meant that they drew wrong conclusion from their programming experience. In this case, negative transfer was observed.

For questions containing FCC-type concepts, more novices had wrong answers that demonstrated the difficulty they had during the code comprehension, and negative transfer was observed. Similar to the TCC-type questions, there were also students who comprehended and answered them correctly. In this case, positive transfer was observed.

For questions containing ATCC-type concepts, no transfer, positive transfer, and negative transfer were all observed. A lot of novices wrote "I do not know", which presented the doubts they had. They could not link the snippet in the question with their learned knowledge and could not make a guess about the output. Thus, no transfer was experienced by them. Some of them could understand the snippet correctly, while some of them made wrong guess. In their responses, the positive transfer and negative transfer were observed separately.

For novice programmers who had programming knowledge in various PLs, positive, negative, and no transfer were also observed during Java code comprehension. In our case, the backgrounds of novices can be found at Table 5.1.

The student who had only experience in Java answered all the questions correctly. In fact, this student was the only one who successfully understood all the snippets. Hence, for this student, only positive transfer was experienced. There were 4 students who had only experience in Python, and there was no special findings in their results. They experienced positive transfer, negative transfer, and no transfer, and all said that they had medium level in Python programming. They made mistakes in basic programming knowledge like nesting of loops, so during the code

comprehension, they would also be influenced by lack of programming experience in general.

Positive, negative, and no transfer were also observed in the students who had programming experience in two or more than two PLs. It was found that for these students, the chance of answering "I do not know" was less than it of students who only learned one PL. The correct rate of these students was also higher. Positive transfer was observed in these cases. Whereas, they still made mistakes or replied "I do not know" in some questions, which indicated the existence of negative transfer and no transfer. Some of them made mistakes such as misconceptions on indexes of arrays, which presented that they might also lack programming experience. Nevertheless, it was normal to find that they did not have enough experience in programming, because they were still novice programmers.

6.1.3 Other Findings

In addition to the discussion mentioned above, there were other findings in the results of the questionnaire.

We found that students with C programming background tend to treat strings or arrays in their smallest unit. For example, in Question 2, there were students with C programming experience writing the answer as "Hel". The output of the loop should be two "Hello" strings if the snippet did not contain compiling error. These students thought that the loop worked to print the first three letters of the string. In Question 8, similar things happened again. There were students saying that the output of the second method was "exe" or "c". They believed that `"println("exec"+3)"` stood for printing the first three letters, or the forth letter of the string "exec". The reason why these C-background students thought in this way could be that there are not as many built-in functions or simple features in C to use directly. C is often considered a foundational or basic programming language. It is an essential language for many programmers to learn, especially low-level programming or understanding computer architecture. Therefore, programmers need to write a lot of functions in C themselves. For novices, they might have learned the way to traverse a string or array, which is usually using loops to go through each unit of the data structure. Besides, after practicing this, they may have developed a fixed mindset to consider these data structures in their smallest unit. When they looked at snippets written in higher-level languages compared to C, they were still likely to think in the same mindset intuitively.

We found another interesting phenomenon in answers of a student with only knowledge in C. The student wrote many details in the answers to describe the way of understanding the snippets. In some answers, "in C" was clearly pointed out, which demonstrated that the student was thinking based on C programming knowledge. However, in some answers, OOP features were mentioned such as "class", "object" and "objectname.gen". These words indicated that the student had some impressions on Java or OOP. The student might also have learned about them. However, instead of reducing the difficulty in understanding of Java snippets, these

impressions influenced negatively on the accuracy of this student in answering the questions. Nowadays, the information spreads fast, and the number of people learning programming has been increasing swiftly [32]. Novices could get new inputs from the website or from their friends. This situation provides various ways for novices to learn programming, and they could easily hear of some information about a PL that they may learn in the future. Whereas, the case of this student told us that the impression novices had on the PL could be incorrect, which might become obstacles when they learn the PL.

There was one student who had experience in Java, C and Python among the participants. This student should have a good performance according to the Hypothesis and discussion, because more coding exercises were assumed to be done by this student. However, the correct rate of this student was 40%. The last three questions were answered correctly by this student, which showed that basic Java programming knowledge was learned. Whereas, the replies of Question 1, 2 and 4 showed that the student was influenced by the Python experience, and made mistakes in understanding the snippets which contained even more basic Java knowledge. Hence, when a novice programmer has leaned different PLs but is not proficient in any of them, he still faces problems comprehending snippets written in a new PL or one of the learned PL. It is also possible to happen that different types of concepts in these PLs would influence the code comprehension negatively.

6.2 Threat to Validity

In this chapter, threats to validity will be discussed, which are factors that may compromise the accuracy, reliability, or generalizability of the findings. Three types of threats to validity are introduced here: construct validity, internal validity, and external validity.

6.2.1 Construct Validity

Construct validity refers to the extent to which the thesis accurately measured the theoretical concepts it intended to measure. In this thesis, whether or not the programming knowledge transfer could be measured by analysing the results of the Java questionnaire needs to be checked.

In the questionnaire, each snippet in questions was designed to contain different types of concepts, and the outputs of the snippet were required to be written by participants. For participants taking part in the experiment in either online or hand-written format, they could write anything they want in the input box. Therefore, they could describe their thoughts about a snippet rather than only provide an answer. By analyzing the answers from participants, where they met problems or where they made mistakes in could be spotted. With this information, the transfer of programming knowledge could be measured. However, not all of the participants explained the reason why they came to this conclusion, the form of using a question-

naire can be improved. For instance, individual interviews after the questionnaire can be used to know more about the thoughts of participants. It is also possible to decrease the number of questions and ask participants to describe what the snippet is doing.

Despite these factors that could be improved, a questionnaire with code comprehension tasks can be used to observe the programming knowledge transfer happening on participants.

6.2.2 Internal Validity

Threats to internal validity relate to the ability to draw accurate conclusions about cause and effect relationships between the independent variable and the dependent variable. The conclusions should not be influenced by other confounding factors as much as possible.

In this thesis, there were some confounding factors. Firstly, the questionnaire only contained simple snippets of Java programming, which means that it was possible for some students to answer correctly the questions containing concepts in FCC and ATCC type. However, considering the participants were still novices in programming and had not learned Java yet, if the snippets were all complicated and long, they would have trouble understanding those questions containing TCC concepts. In order to mitigate the influence of this confounding factor and keep a balance in the complexity of the snippets, short and longer snippets were all used.

Secondly, the participants had various abilities to code and had different numbers of PLs learned, which also influenced the results. When designing the questionnaire, the participants were expected to be students who had only learned C programming or students who had only learned Python programming. Whereas, the actual participants had various background. A few of then had learned Java before, while some of them had learned Python and C, which made it hard to tell whether their results were influenced by C knowledge or Python knowledge. To decrease the influence, the responses of participants with different backgrounds were collected separately when creating categories. Individual analysis about some special responses was also performed.

Thirdly, loss of participants appeared in the latter problems, especially in Question 7, 8, 9, and 10, leading to biased results. This factor might be caused by lack of focus from participants, and it suggests that the design of the questionnaire should be more careful. The number of questions in the questionnaire may need to be reduced. The difficulty of snippets may also need to be modified. Considering the confounding factor mentioned at the beginning, it is hard and significant to choose proper snippets.

There was a possible confounding factor in the order of the questions. The answer for the first two questions were both "error when compiling", which might influence the results of novices. The reason for this is that it might not occur to students that the first two snippets were incorrect. They were likely to try their best in writing outputs rather than pointing out the mistakes in the snippets. Therefore, the order

of the questions should be changed for further studies.

The questionnaire was designed to be filled by novices before they started to learn Java, so that their answers could reflect the intuitive understanding of the questions based on their previous programming knowledge completely. If more pilot studies are done before the actual questionnaire to test the snippets, and a post questionnaire or interview is carried out after the questionnaire, the results will be improved.

6.2.3 External Validity

Threats to external validity are about the generalizability of the study findings to other populations, settings, or conditions. In this thesis, the participants were all university students in the course Data Structures in summer semester 2023. The PLs they had learned were C, Python, and Java. Besides, the questionnaire was designed for novices with programming background in C and Python. Hence, there will be problems if the participants of the questionnaire have different programming backgrounds. The target group of the questionnaire should only be novices because the snippets were not complex. However, the questionnaire can still be used in studies where participants are novices but not university students. The questionnaire was filled by the participants before they started to learn the new PL (Java). The purpose of it was to find the intuitive knowledge transfer in participants. If participants have already learned something in the new PL, the experiment design needs to be changed to observe their knowledge transfer.

6.3 Comparison to Related Work

There are studies done to discover the mistakes made by novices. Brown and Altadmri categorized the mistakes into three categories: misunderstanding syntax, type errors, and other semantic errors [3, 4]. They analyzed the Blackbox data set of over 100,000 students. These mistakes were also found in the results of this thesis, which show that syntax and semantics related knowledge are big obstacles for novices. This assumption can be verified in the work of McCall and Kölling [25].

The studies about mistakes students made during learning Java [3, 4, 16, 33] suggested that students still would meet a lot of problems. Hence, it is important to carry out research on this topic, and this thesis was an attempt to contribute to this topic.

Many studies looked into the performance of students during exams [23, 50, 22]. However, the performance of students during exams might not be the same as it of students in conditions where they feel no pressure. This thesis tried to observe the performance novices have in normal learning situation.

The work of Izu and Mirolo [15] explored the learning transfer of novice programmers when they solving two related coding tasks in C. Four types of transfer were identified that covered positive transfer and negative transfer mentioned in this the-

sis. Also two types of non-transfer were identified by Izu and Mirolo [15] which included the no transfer mentioned in this thesis. In this thesis, knowledge transfer between different PLs was observed, and the focus was to discover how novices intuitively transfer their knowledge.

Some studies focused on the transfer process from block-based to text-based programming [52, 53, 30]. They proved that it is difficult for novices transfer from one PL to another PL that are different in syntax. The findings of this thesis also supported this assumption.

In the research of Tshukudu and Cutts [47], transfer from procedural Python to object-oriented Java was studied. They carried out several interviews with the participants in 10 weeks, and found that participants had little difficulty with carryover concepts, while had trouble mapping changed concepts. These findings are similar to the assumptions in this thesis that novices are more likely to experience positive transfer when comprehending TCC-type concepts, and are more likely to experience negative transfer when understanding FCC-type concepts. In the study of Tshukudu and Cutts [47], the participants were four novices and one experts from five universities. The number of participants is less than it of this thesis (104 students).

7 Conclusion and Future Work

7.1 Conclusion

Research has shown that it is difficult to learn second and subsequent programming languages [35]. For novice programmers who just start their journey of learning to program, the difficulty is even greater. In this thesis, we aim to observe the programming knowledge transfer that novices intuitively have, and discover the problems they meet in the process of code comprehension. In order to achieve this goal, we designed and used a Java questionnaire containing ten snippets covering three types of concepts (TCC, FCC, and ATCC) to collect responses from 104 novices studying in TUC before they started to learn the new PL (Java).

By analyzing the results of the questionnaire, we found that the types of transfer observed had no difference for novices with various programming backgrounds. They all tended to have positive transfer when comprehending Java snippets containing TCC-type concepts, have negative transfer when understanding Java snippets containing FCC-type concepts, and have no transfer when comprehending Java snippets containing ATCC-type concepts. These findings are similar as the study of Tshukudu and Cutts [47]. We also found that for novice programmers who only had experience in C programming, the most frequently observed transfer was negative transfer. For novices with other programming experience, it was possible for them to have misunderstandings in basic programming knowledge such as condition of loops and nesting of loops. In addition, as the number of learned PLs increased, novices were more likely to transfer their previous knowledge successfully to new scenarios. Some special characteristics of students with C PL background were also found in their responses.

We hope that these findings of the thesis could be used to improve pedagogy on how to make it easier for students to go through the learning transfer phase. The knowledge transfer is inevitable if a programmer wants to learn a new PL. Therefore, if some pedagogy could be used to help students or programmers take advantage of the knowledge transfer, they will learn the second and subsequent programming languages faster and with less obstacles.

However, there are some limitations of this thesis that need to be reduced in future research. For instance, the length of the questionnaire needs to be modified to reduce the loss of responses, pilot studies could be carried out before the questionnaire, and the order of the questions should be changed, and so on.

In conclusion, a deep observation of intuitive behavior of novice programmers with different programming background was done in the thesis. The results of the

thesis supported some conclusions of related research, and added new ideas about behavior of novices during code comprehension. The number of participants was large, so the findings of the thesis were built on a relatively general basis.

7.2 Future Work

In this thesis, a questionnaire was distributed to participants before they learn the new PL. In their responses, not all of them wrote detailed explanation about how they got to the answers. Hence, interviews or post-questionnaire can be done before they learn the new PL to have more understanding of their thoughts during the code comprehension. A think aloud protocol can also be useful in this case. Additional instrumentation, such as EEG devices and eye trackers, can be used to discover more about the behavior of novices when they read code snippets.

Another code comprehension questionnaire can be designed with more complicated snippets to collect the responses from these novices after they learned Java for one semester. With this data, we could find out if there are changes in learning transfer. More analysis can be done to check the participants' performance of the final exam of the programming course to see if they still suffer from the transfer problems. This information could help us think about how we can improve the pedagogy to reduce the pressure of novices caused by programming language transfer.

Bibliography

[1] Bonar, J., Soloway, E.: Uncovering principles of novice programming. In: Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. pp. 10–13 (1983)

[2] Boysen, J.P.: Factors affecting computer program comprehension. Iowa State University (1979)

[3] Brown, N.C., Altadmri, A.: Investigating novice programming mistakes: Educator beliefs vs. student data. In: Proceedings of the tenth annual conference on International computing education research. pp. 43–50 (2014)

[4] Brown, N.C., Altadmri, A.: Novice java programming mistakes: Large-scale data vs. educator beliefs. ACM Transactions on Computing Education (TOCE) 17(2), 1–21 (2017)

[5] Byrnes, J.P.: Cognitive development and learning in instructional contexts. (No Title) (1996)

[6] Corritore, C.L., Wiedenbeck, S.: What do novices learn during program comprehension? International Journal of Human-Computer Interaction 3(2), 199–222 (1991)

[7] Craig, S., Graesser, A., Sullins, J., Gholson, B.: Affect and learning: an exploratory look into the role of affect in learning with autotutor. Journal of educational media 29(3), 241–250 (2004)

[8] Denny, P., Luxton-Reilly, A., Tempero, E., Hendrickx, J.: Understanding the syntax barrier for novices. In: Proceedings of the 16th annual joint conference on Innovation and technology in computer science education. pp. 208–212 (2011)

[9] Dreyfus, H.L.: Hubert dreyfus and stuart dreyfus mind over machine: The power of human intuition and expertise in the era of the computer (new york: The free press, 1986), p. 50 table 1-1. five stages of skill acquisition

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

[11] Fjeldstad, R.K.: Application program maintenance study. Report to Our Respondents, Proceedings GUIDE 48 (1983)

[12] Garcia-Martinez, S., Zingaro, D.: Teaching for transfer of learning in computer science education. Journal for Computing Teachers pp. 1–6 (2011)

[13] Gunnarsson, K., Herber, O.: The most popular programming languages of github's trending repositories (2020)

[14] Gutiérrez, L.E., Guerrero, C.A., López-Ospina, H.A.: Ranking of problems and solutions in the teaching and learning of object-oriented programming. Education and Information Technologies 27(5), 7205–7239 (2022)

[15] Izu, C., Mirolo, C.: Learning transfer in novice programmers: A preliminary study. In: Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1. pp. 178–184 (2021)

[16] Jackson, J., Cobb, M., Carver, C.: Identifying top java errors for novice programmers. In: Proceedings frontiers in education 35th annual conference. pp. T4C–T4C. IEEE (2005)

[17] Kelleher, C., Pausch, R.: Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM computing surveys (CSUR) 37(2), 83–137 (2005)

[18] Klump, R.: Understanding object-oriented programming concepts. In: 2001 Power Engineering Society Summer Meeting. Conference Proceedings (Cat. No. 01CH37262). vol. 2, pp. 1070–1074. IEEE (2001)

[19] Kölling, M.: The problem of teaching object-oriented programming, part 1: Languages. Journal of Object-oriented programming 11(8), 8–15 (1999)

[20] Kölling, M., Brown, N.C., Altadmri, A.: Frame-based editing: Easing the transition from blocks to text-based programming. In: Proceedings of the Workshop in Primary and Secondary Computing Education. pp. 29–38 (2015)

[21] Lahtinen, E., Ala-Mutka, K., Järvinen, H.M.: A study of the difficulties of novice programmers. Acm sigcse bulletin 37(3), 14–18 (2005)

[22] Lee, D.M.C., Rodrigo, M.M.T., Baker, R.S.d., Sugay, J.O., Coronel, A.: Exploring the relationship between novice programmer confusion and achievement. In: Affective Computing and Intelligent Interaction: 4th International Conference, ACII 2011, Memphis, TN, USA, October 9–12, 2011, Proceedings, Part I 4. pp. 175–184. Springer (2011)

[23] Lopez, M., Whalley, J., Robbins, P., Lister, R.: Relationships between reading, tracing and writing skills in introductory programming. In: Proceedings of the fourth international workshop on computing education research. pp. 101–112 (2008)

[24] Maalej, W., Tiarks, R., Roehm, T., Koschke, R.: On the comprehension of program comprehension. ACM Transactions on Software Engineering and Methodology (TOSEM) 23(4), 1–37 (2014)

[25] McCall, D., Kölling, M.: Meaningful categorisation of novice programmer errors. In: 2014 IEEE Frontiers in Education Conference (FIE) Proceedings. pp. 1–8. IEEE (2014)

[26] McCall, D., Kölling, M.: A new look at novice programmer errors. ACM Transactions on Computing Education (TOCE) 19(4), 1–30 (2019)

[27] Nelson, H.J., Irwin, G., Monarchi, D.E.: Journeys up the mountain: Different paths to learning object-oriented programming. Accounting, Management and Information Technologies 7(2), 53–85 (1997)

[28] Perkins, D.N., Salomon, G., et al.: Transfer of learning. International encyclopedia of education 2, 6452–6457 (1992)

[29] Poo, D., Kiong, D., Ashok, S.: Object-oriented programming and Java. Springer Science & Business Media (2007)

[30] Powers, K., Ecott, S., Hirshfield, L.M.: Through the looking glass: teaching cs0 with alice. In: Proceedings of the 38th SIGCSE technical symposium on Computer science education. pp. 213–217 (2007)

[31] Robins, A., Haden, P., Garner, S.: Problem distributions in a cs1 course. In: Proceedings of the 8th Australasian Conference on Computing Education-Volume 52. pp. 165–173 (2006)

[32] Robins, A., Rountree, J., Rountree, N.: Learning and teaching programming: A review and discussion. Computer science education 13(2), 137–172 (2003)

[33] Rodrigo, M.M.T., Andallaza, T.C.S., Castro, F.E.V.G., Armenta, M.L.V., Dy, T.T., Jadud, M.C.: An analysis of java programming behaviors, affect, perceptions, and syntax errors among low-achieving, average, and high-achieving novice programmers. Journal of Educational Computing Research 49(3), 293–325 (2013)

[34] Rodrigo, M.M.T., Baker, R.S., Jadud, M.C., Amarra, A.C.M., Dy, T., Espejo-Lahoz, M.B.V., Lim, S.A.L., Pascua, S.A., Sugay, J.O., Tabanao, E.S.: Affective and behavioral predictors of novice programmer achievement. In: Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education. pp. 156–160 (2009)

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

[36] Shinners-Kennedy, D., Fincher, S.A.: Identifying threshold concepts: From dead end to a new direction. In: Proceedings of the ninth annual international ACM conference on International computing education research. pp. 9–18 (2013)

[37] Shrestha, N.: Towards supporting knowledge transfer of programming languages. In: 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). pp. 275–276. IEEE (2018)

[38] Siegmund, J.: Program comprehension: Past, present, and future. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). vol. 5, pp. 13–20. IEEE (2016)

[39] Siegmund, J., Kästner, C., Liebig, J., Apel, S., Hanenberg, S.: Measuring and modeling programming experience. Empirical Software Engineering 19, 1299–1334 (2014)

[40] Slonneger, K., Kurtz, B.L.: Formal syntax and semantics of programming languages, vol. 340. Addison-Wesley Reading (1995)

[41] Snyder, A.: Encapsulation and inheritance in object-oriented programming languages. In: Conference proceedings on Object-oriented programming systems, languages and applications. pp. 38–45 (1986)

[42] Soloway, E., Spohrer, J.C.: Studying the novice programmer. Psychology Press (2013)

[43] Stefik, A., Siebert, S.: An empirical investigation into programming language syntax. ACM Transactions on Computing Education (TOCE) 13(4), 1–40 (2013)

[44] Taipalus, T., et al.: Actionpool: A novel dynamic task scheduling method for service robots (2010)

[45] Teague, D., Lister, R.: Manifestations of preoperational reasoning on similar programming tasks. In: Proceedings of the Sixteenth Australasian Computing Education Conference [Conferences in Research and Practice in Information Technology, Volume 148]. pp. 65–74. Australian Computer Society (2014)

[46] Tshukudu, E.: Understanding conceptual transfer in students learning a new programming language. Ph.D. thesis, University of Glasgow (2022)

[47] Tshukudu, E., Cutts, Q.: 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. pp. 307–313 (2020)

[48] Tshukudu, E., Cutts, Q.: Understanding conceptual transfer for students learning new programming languages. In: Proceedings of the 2020 ACM conference on international computing education research. pp. 227–237 (2020)

[49] Tutty, J., Sheard, J., Avram, C.: Teaching in the current higher education environment: perceptions of it academics. Computer Science Education 18(3), 171–185 (2008)

[50] Venables, A., Tan, G., Lister, R.: A closer look at tracing, explaining and code writing skills in the novice programmer. In: Proceedings of the fifth international workshop on Computing education research workshop. pp. 117–128 (2009)

[51] Wegner, P.: Concepts and paradigms of object-oriented programming. ACM Sigplan Oops Messenger 1(1), 7–87 (1990)

[52] Weintrop, D., Bain, C., Wilensky, U., Education, U.: Blocking progress? transitioning from block-based to text-based programming. Proc. Amer. Educ. Res. Assoc. pp. 1–8 (2017)

[53] Weintrop, D., Wilensky, U.: Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs. In: Proceedings of the eleventh annual international conference on international computing education research. pp. 101–110 (2015)

[54] Winskel, G.: The formal semantics of programming languages: an introduction. MIT press (1993)

[55] Winslow, L.E.: Programming pedagogy—a psychological overview. ACM Sigcse Bulletin 28(3), 17–22 (1996)

[56] Wu, C.T.: An Introduction to object-oriented programming with Java TM. Mcgraw-Hill Incorporated (2006)

[57] Wu, Q., Anderson, J.R.: Problem-solving transfer among programming languages. Tech. Rep. (1990)