ASSESSMENT OF UNDERSTANDABILITY OF OBJECT ORIENTED PROGRAMMING AMONG LEARNERS: A CASE STUDY OF GRETSA UNIVERSITY

ALI IBRAHIM ALI

ICT-4-1829-16

A RESEARCH PROJECT SUBMITTED TO THE SCHOOL OF COMPUTING AND INFORMATICS IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE IN COMPUTER SCIENCE OF GRETSA UNIVERITY

OCTOBER 2019

Declaration

This research project is my original work and has not been presented for award of a Degree in Computer Science or for any similar purposes in any other institution.

Signature: _____ Date: _____

Ali Ibrahim Ali

ICT-4-182916

This research has been submitted with my approval as University Supervisor.

Signature: _____ Date: _____

Prof./Dr./Mr./Ms.: _____

School of Computing and Informatics

Gretsa University, Thika.

Table of Contents

List of Tables

List of Figures

Abbreviations and Acronyms

C – A high level structured programming language used for computer programming.

C++ – A high level object oriented programming language used for computer programming.

GST – General System Theory

OO – Object Oriented

OOP – Object Oriented Programming

SQL – Server Query Language

Operational Definition of Terms

Attribute – In OOP; a specification that defines a property of an object.

Class – In OOP; a template of blueprint that describes the behavior or states that an object of its type supports.

Code – In computer programming; a series of symbols written in a predetermined fashion used to give instruction to a computer.

Function – A method.

High Level – A language that is understandable by a human being but has to be translated to machine code to be understood by the computer.

Language – In computer programming; a system of conventional written symbols which are used to give instructions to a computer.

Method – In OOP; a container of program logic, data and all actions related to the program that is contained within a class.

Object – In OOP; an instance of a class with states and behaviors.

Pascal –An imperative and procedural computer programming language.

Programming – The process of designing and building an executable computer program.

Abstract

Computer programming is a vital part of the computer science curriculum. This research describes an investigation into some of the academic problems the face new programming students as they learn object oriented programming. Specifically, the researcher looked into the problems that arise during the study of object oriented programming languages. Object oriented programming languages have been proven to be more complex to learn as compared to structural programming languages. The research is focused on two of some of the most popular programming languages, C++ and Java. These languages were selected because they were taught at Gretsa University as well as other Universities across the country. The study involved Gretsa University students who were in the School of Computing and Informatics and pursuing either a Diploma in Computer Science, Information Technology or Software Engineering or a Degree in Computer Science. Questionnaires were handed out to a sample selected using purposive sampling technique. The study found that the students did have difficulty in learning various areas and also recommended some few actions that could be taken to improve understanding of new concepts.

CHAPTER ONE: INTRODUCTION

1.1 Introduction

The background of the study is helpful in giving a foundation of the concepts of research. This chapter has handled the research concepts background, elaborated on the problem statement, given the purpose of the study and discussed the conceptual framework. The chapter has also covered the general and specific objectives, significance of research, scope of the study, and the limitations of the study.

1.2 Background to the Study

Programming languages have been known to give students challenges due to the fact that they are relatively new concepts that had not been covered in their prior years of education. This situation if further aggravated by the fact that in secondary school level, only mathematics, languages and sciences are required as the bare minimum for entry in computer science related courses.

The type of programming languages that have been taught over time may not be the reason why students have foundthem hard rather, it has been the abstract programming concepts and logical reasoning processes. The nature of the programming languages themselves has become the actual hindrance in understanding their concepts. Further, being in group classes has caused the lack of specified instructions for students based on their understanding levels.

To ease on the learning process, new students have been required to learn structural or procedural programming languages and their concepts such as C or Pascal in their first year of study. This method has ensured the once they move on to object oriented programming, they are familiar with programming concepts. More often than not however, the expected learning curve

has not been achieved. This has as a result of having structural programming present challenges to these students and these challenges being inherited as the students move on to object oriented programming.

Object Oriented programming is a programming paradigm that is based on the concept of object that can contain data in the form of attributes or properties and code in the form of methods. Compared to procedural programming, object oriented programming is much more powerful and with power, comes complexity. Being a requirement for any computer science related course, object oriented programming has been an obstacle that all students have had to contend with.

With object oriented programming, students have had to learn the four pillars of this programming paradigm. These pillars have been identified as encapsulation, abstraction, inheritance and polymorphism. Further, students have had to learn more library importation and manipulation procedures, adding difficulty to the new concepts.

Inheritance has been identified as one of the most important concept in OOP. This functionality has allowed programmers to define a class following the description of another class. The advantage of this concept has been easing the application maintenance process. In addition, it has allowed the reuse of code, significantly reducing theimplementation time required to generate an application. With inheritance, new terminologies crop up including the base class and the derived class. The base class is the existing class from which code is inherited while the derived class is the new class that inherits code.

Another aspect that has enabled OOP is polymorphism. This concept occurs when there is a hierarchy of classes and they are related by inheritance. In other words, it means having many forms. Further elaborated, this concept has allowed the execution of a different function based on a call to a member function and the object that invokes the function.

The third pillar of OOP is data abstraction, otherwise known as data hiding. This concept has allowed the provision of only essential data to the outside world whilst hiding the background details. It relies on the separation and implementation, and this is provided by classes.

The last pillar involves the encapsulation of data. In OOP, programs are consistent of two fundamental concepts. These concepts have been identified as program statement (code) and program data. Encapsulation in this sense has allowed the binding of data and the functions that manipulate this very data. In addition, this concept keeps both the functions and the data free from outside interference and misuse. This lead to the important concept of data hiding.

These four pillars are not all thesestudents have to understand. They need to understand more concepts including objects, classes, methods and variables in their various forms. This has made it quite difficult considering the time allocated for teaching, as well as the conflict in the meaning of terms from the norm.

Over the duration of their course, students have to learn more than one programming language. In Gretsa University for instance, a diploma student has to learn C, C++, Visual Basic and SQL. During their degree level, they have to add more programming languages to their knowledge base including Java. The hindrance in this learning process has been in the understanding of the programming language syntax and structure which more often than not is unique for each programming language. Figures 1, 2, 3 and 4 illustrate this concept using a “Hello World!” program.

```
#include <iostream>
using namespace std;
int main(){
        cout<<"Hello World!"<<endl;
}
```

Figure 2: C++

```
#include <stdio.h>
intmain(){
        printf("Hello World!");
}
```

Figure 1: C

```
Module HelloWorld
  Sub Main()
        System.Console.WriteLine("Hello
World!")
  End Sub
```

Figure 3: Visual Basic

```
public class Main{
  public static void main(){
    System.out.println("Hello World!");
  }
}
```

Figure 4: Java

1.3 Statement of the Research Problem

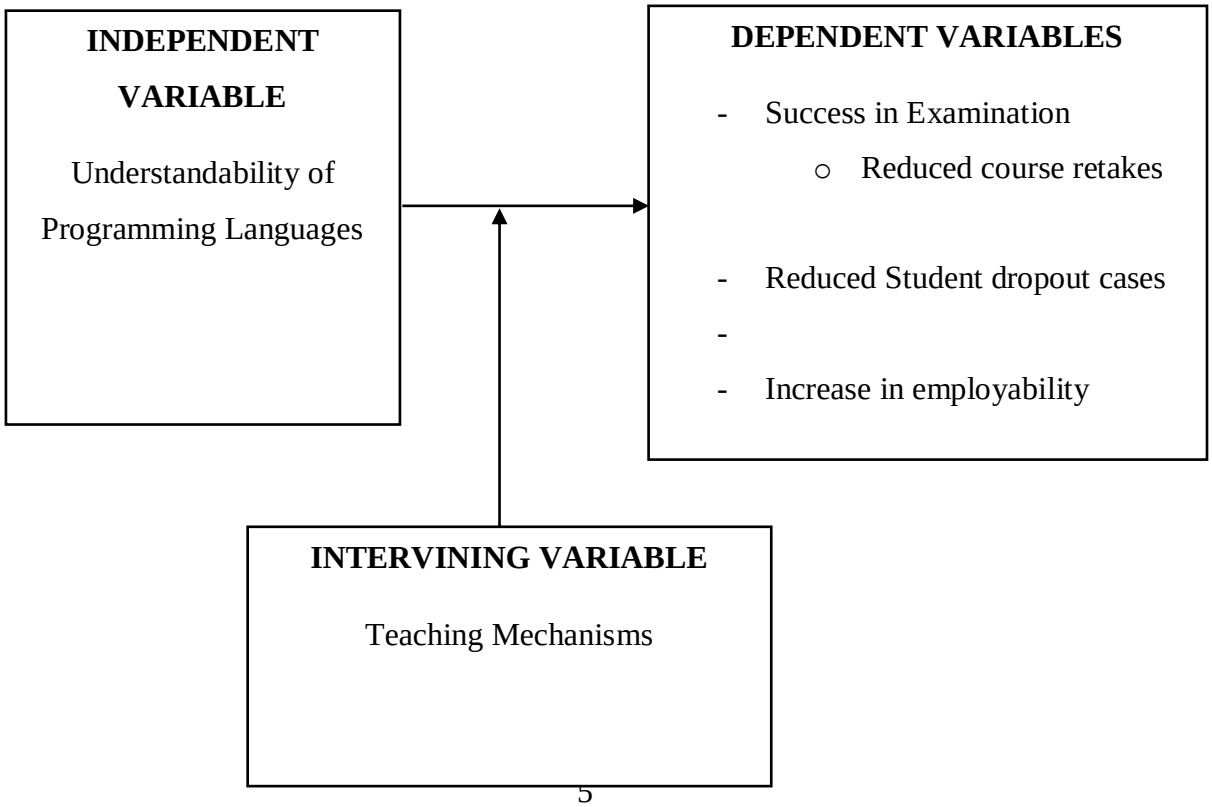
Programming has been proven to be a challenging to learn and understand. This proof has been founded on the high dropout out rates of students undertaking programming courses. As a result of traditional study methods employed for teaching programming, students get stuck at beginner level programming and lack morale to advance further (Matthew Butler, et. al. 2007).At Gretsa University, this problem is observed as some of the most failed units that result in supplementary examinations include OOP courses. Most students prefer to cram examinations which works well for theory units but not programming units.

1.4 Purpose of the Study

This study sought to assess the understandability of programming languages of students learning computer science and related courses, and the difficulty they experience during and after studies. Further, the study looked into mechanisms that could ease learning for students doing programming languages, especially OOP related languages.

1.5 Conceptual Framework

Figure 5: Conceptual Framework



1.6 Objectives of the Study

1.6.1 General Objective

To investigate the level of understandability of programming concepts and the difficulties associated with learning programming languages.

1.6.2 Specific Objectives

1. To investigate the level of understandability of programming concepts.
2. To investigate the difficulties associated with learning programming languages.
3. To investigate the teaching model used to teach programming languages ate Gretsa University.

1.7 Research Questions

1. How well do students understand object oriented programming concepts?
2. What difficulties do students face as they learn object oriented programming languages?
3. What teaching model is used to teach students object oriented programming languages?

1.8 Significance of the Study

This study's significance resides in its ability to illuminate on the aspect of how the experience of learning programming languages is constituted during the duration of study for students. If further opens territory of understanding the learning process including the difficulties encountered and recommends how to tackle them. This study would then allow for future reference in this study area and help lecturers as well as students come up with the best mode of teaching and study respectively. In turn, this would reduce the dropout and examination failure rates observed in the past.

1.9 Scope of the Study

This study shall be limited only to the study of students within the School of Computing and Informatics of Gretsa University, Thika. Gretsa University was selected as a significant scope due to its availability of students who studied computer science among other courses that taught programming languages. This meant that the student faced the challenges that the research topic was based upon. Further, the research was focused on the challenges faced by the students at the aforementioned location whilst learning computer programming languages, and especially object oriented programming languages.

1.10 Limitations of the Study

This study was anticipated to meet the challenge of information acquisition due to lack of willingness to provide information as may be required by the questionnaire, by the respondents.

CHAPTER TWO: LITERATURE REVIEW

2.1 Introduction

This chapter has covered various publications related to object oriented programming, teaching models and student behavior. Related quotations and discussions have been organized into review of literature, themes, theoretical frameworks and then summarized as gaps in literature.

2.2 Review of Literature

The process of learning programming languages is made up of three parties; the student as the learner, the lecturer as the teacher and the university as the facilitator (PaiviKinnunen, 2009). In a classroom facilitated by the university, the lecturer would often employ the instructional process to teach concepts. The instructional process was defined as; a wide concept consisting of all the important components that take place in classroom instruction as well as the steering factors defined in the curriculum, comprising of both teacher's and pupil's actions in totality, as applied in institutions of higher learning.

(PaiviKinnunen, 2009) further notes there were two theoretical frameworks that could be used to tackle the instruction process as a whole however, the most appropriate would be the general system theory (GST). In their description, elaborated that the GST had the ability to tackle the process nature of instruction, focusing on the process of teaching. In their illustration, they used a diagram representing input, an open system and output. In the analogy, the student would be the open system receiving input (instructions) and then acting on this input to give output (result). To reinforce the study, they were motivated by the outcome of high dropout rates for students, and represented the process as shown in figure 6.

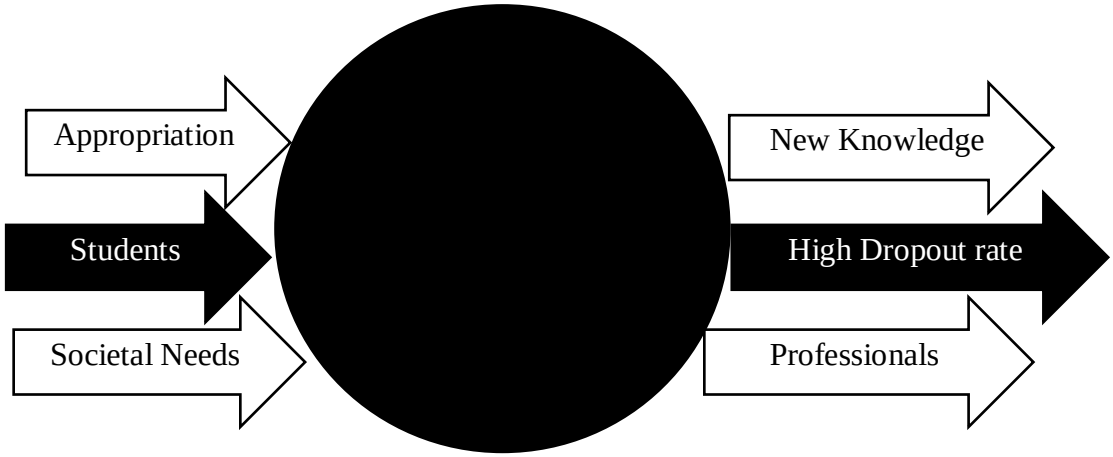


Figure 6: Instructional System

During the first year of study, the core aims of the units taught has been to provide students with the fundamentals of problem solving, program design and implementation of the programs (Matthew Butler et. Al., 2007). Some of the students that are involved in the study process have no prior knowledge of programming. Since these units are considered introductory, covering both structured and object oriented programming paradigms, the curriculum spans from low to high levels of conceptual complexity.

In this regard, challenges arise as the curriculum must balance between low level issues such as syntax needed for program implementation to high level issues such as object orientation. While the former allows for the implementation of the program and the latter allows for the efficiency in program design, both issues are indispensable. In as much as there are semantics understanding required and is clearly defined, verifiable and introductory, the content covered is not normally large. That coupled with decision making and loops ideally are easy to understand. On the other hand, object oriented pillars including encapsulation, abstraction, polymorphism and inheritance are considered high level issues operating from levels far beyond basic operations of the application (Matthew Butler et. Al., 2007).

(BusraOzmen et. Al., 2014) studied some of the issues that may cause difficulties in the programming learning process. After the study, they classified the themes into three; personal

problems, class activities problems and programming process problems. These problems have some factors they played in the overall failure of the students. Figure 7 illustrates their findings.

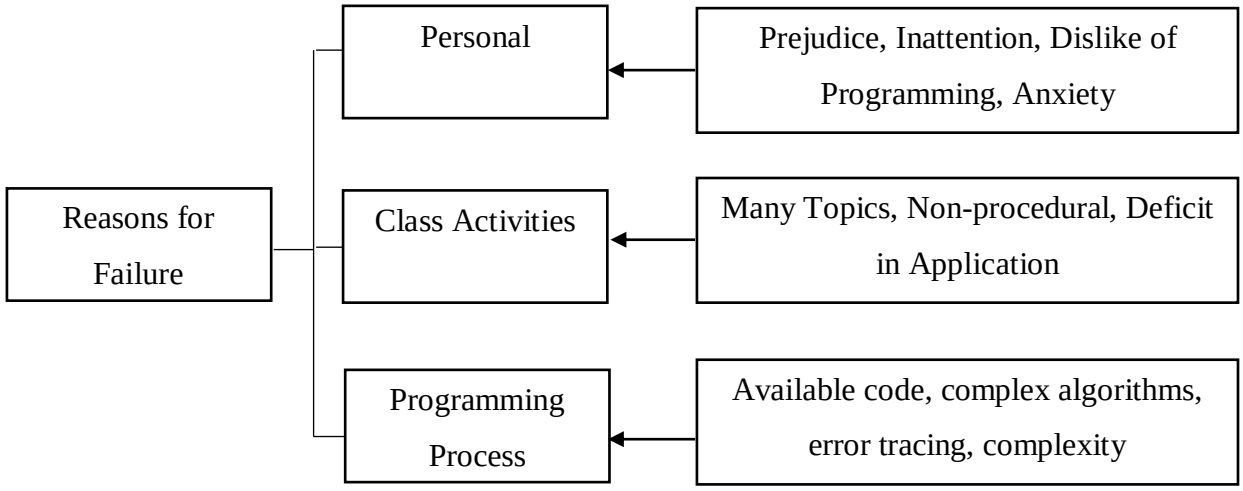


Figure 7: Reasons for failure in programming

2.3 Understandability of OOP Concepts among Students

For most students starting their programming courses, they reside in the scope of novices or beginners. These are students who have very little to no knowledge on matters programming. For a person to be considered an expert, they needed at least ten years of consistent programming. Common features for students beginning their programming courses included knowledge of only the surface level of programming. As a result, such students had the tendency to approach programming line-by-line as opposed to larger program structures (KirstiAla-Mutka, 2007).

In addition to programming line-by-line, students tended to spend very little time in planning and testing their code. As observed, students ended up making small localized fixes to fix problems encountered during execution as opposed to thoroughly reformulating the programs. This led to

frustration and failure to progress forward because such fixes would not result in permanent progress.

As observed, most students did not usually make much progress in their introductory programming courses. They either did not progress at all or they shifted to a different programming language to specialize in their future careers after they finished their courses. A more personalized problem resulted from students believing they had understood an issue whilst they were still in the process of learning. This resulted in ignorance and overall lack of understanding by the students.

2.4 Difficulties Associated with Learning OOP

Learning OOP has its own unique difficulties. In addition, it inherits all the difficulties that are experienced while learning structural programming. To learn a programming language, one has to understand several concepts during the learning process including syntax, semantics and program comprehension. With students, program comprehension becomes quite daunting especially when loop logic and debugging is concerned (Anna Eckerdal, 2009).

Over the years, object oriented programming has been seen as a natural way of conceptualizing real world problems. This often leads to underestimating the difficulties associated with the learning process with the teachers assuming its ease for students. In a study performed using Pascal/C and C++ programs, students were found to have major difficulty in identifying hidden actions. This made it more difficult for students to comprehend OO programs. As a result, most students opted for structural programs while solving and debugging errors.

Both the teachers and students considered constructors, virtual functions and function overriding as major difficulties in the same study.

2.5 Teaching Models Used in Teaching Programming Languages

The teaching model used during the learning process has been the core contributor in the overall success of the students who learn programming languages. Particular attention is necessary in the initial learning stage as it forms the foundation of the programmer's success (Gomes Anabela et. Al., 2007).

(Gomes Anabela et. Al., 2007) highlighted the first two important factors in the learning process as the teaching method and the study methods used by the lectures and students respectively. In their study, they diagnosed the problems that arose from the methods used in respect to teaching and study over the course of their research.

The traditional method of teaching that was widely in use had quite a number of shortcoming. For theoretical subjects that required only understanding of concepts, the methods were okay but when it came to programming, they put the students being taught at a disadvantage. The first issue that arose was the lack of personalization. The availability of personalized teaching would allow for better supervision, details explanations and immediate feedback. The second problem was the lack of support for all students' learning styles. This short-handed the students who learned slower or were more receptive to practical lessons.

Another problem that arose from use of traditional teaching methods was the teaching of dynamic concepts through static materials. The static materials included presentations, verbal explanations, diagrams and text. This led to failure of understanding program dynamics and consequently failure in understanding programming concepts in question. The last issue (Gomes Anabela et. Al., 2007) observed was the emphasis on teaching of syntactic details as opposed to promoting problem solving using said programming languages. This resulted in redundant abilities and difficulty in understanding of more complex concepts.

The second factor in the learning process involved the students' study methods. The first and major problem was the use of methodologies borrowed from other disciplines that included memorization of formulas, procedures, text or diagrams. These methods would not successfully work on programming since it required regular and active practice. The other shortcoming was the students' lack of motivation to study enough to acquire programming competencies. These two factors added to the overall failure when it came to programming understandability.

2.6 Theoretical Framework

The theoretical approach used by this research is the General System Theory. Dating back to the 1950's and established by an Austrian Biologist, Ludwig von Bertalanffy, this theory can be traced back earlier and has been modified over the years.

GST was developed as a holistic way of looking at the foal-directed behavior of complex system, concerning itself with the functions of a system at an abstract level. This then makes it possible to apply a system-theoretical approach for making sense of various phenomena across different disciplines. Though it was originally developed in the field of Biology, GST has been adopted by other field experts including those in the fields of mathematics and social sciences.

2.7 Summary of Identified Gaps in Literature

Literature had been done on the challenges that involved learning of programming concepts but such literature was majorly done outside of Kenya. During literature review, the researcher stumbled on only a few, old studies that had been done directly related to students in Kenya. Specifically, research lacked on programming difficulties identified within private institutions of higher learning.

CHAPTER THREE: RESEARCH METHODOLOGY

3.1 Introduction

This chapter covers the methods that have been used to collect the data relevant to this research. These methods include the description of the research design, the study area, target population, sampling technique, sample size, research instruments, data collection and analysis and the limitations that were encountered.

3.2 Research Design

This study used the exploratory research design. This design would enable the researcher to collect the views of the correspondents and understand their point of view. It would create a focal point on empathizing with the students and thereby understanding their concerns with regard to the study topic. Doing so would allow the researcher to translate their feedback into analyzable data more effectively and efficiently.

3.3 Study Area

This research was done in Gretsa University, Thika Sub County, Kiambu County, Kenya. This study area was selected due to the presence of students who were actively involved in the learning process of object oriented programming. Further, the institution was selected because it was a private institution which was identified as lacking in literature, and accessibility to the location was not problematic.

3.4 Target Population

The target population was comprised of a total of 2000 students. Of these students, only about 150 were active programming student. This meant that they had learnt programming or were

actively undertaking the course during this research. These students were recognized as a prime target population because they had knowledge in programming and had experienced the challenges that come along with programming.

3.5 Sampling Techniques

This research used two sampling techniques. The researcher used purposive sampling technique. This technique was used to identify the students who were undertaking computer science and computer science related courses that has object oriented programming as a requirement. This enabled the researcher to acquire the first group of students from the initial population of 2000 students. Once that was done, the researcher used snowball sampling technique. The technique enabled the researcher to acquire more students through referrals from the initial students who had been identified through purposive sampling.

3.6 Sample Size

A sample size has to be properly determined based on the expense of data collection and the need to have sufficient statistical power. It is important for any study with the goal of making inferences about a population for a sample. Sample size determination is the act of choosing the number of observations or replications to include in a statistical sample. The researcher acquired a significant number of students (112) from the total number of students who were undertaking computer science and related courses to include in the study. To acquire the study sample size, the researcher used the formula to find sample size for qualitative studies. The predetermined type 1 error was at 5% as a majority of studies considered the expected proportion in population significant below 0.05. Whilst previous studies reviewed in literature reviewshowed that only about 8% of the respondents viewed research to be extremely difficult. The absolute error (precision) was set at 5% or 0.05. The calculation was done as show below.

$$\text{Sample Size} = \frac{Z_{1-a/2}^2 P(1-P)}{d^2}$$

$$\text{Sample Size} = \frac{1.96^2 \times 0.08(1-0.08)}{0.05^2} = 112$$

3.7 Research Instruments

Research instruments are measurement tools designed to obtain data on a topic of interest. The researcher used questionnaires as the research instruments. This would allow the respondents to answer questions, giving enough data required for the study.

3.8 Validity of Measurements

The researcher could not directly influence the response of the students under survey with regard to the answers they gave to the questions asked. This meant the answers were valid opinions from real subjects under study.

3.9 Data Analysis

Once data was collected, the researcher had to process that data info information form the questionnaires. The researcher grouped questions with similar goals and represented them in form of graphs and tables for ease in explanation. This then enabled the availability of patterns that could be used in the other phases of the research.

3.10 Logistical and Ethical Considerations

This research encountered issues with the privacy of data collected, data collections using questionnaires and assurances of confidentiality to the respondents who were used during the research exercise.

CHAPTER FOUR: FINDINGS AND DISCUSSIONS

4.1 Introduction

This chapter has covered all information related to the data collected using the questionnaire used for this research. Information has been represented using tables, charts and graphs to simplify and provide an overview of all data collected. This chapter consists of an overview of the findings, the discussion of the findings, critique of other studies and inferences of the study.

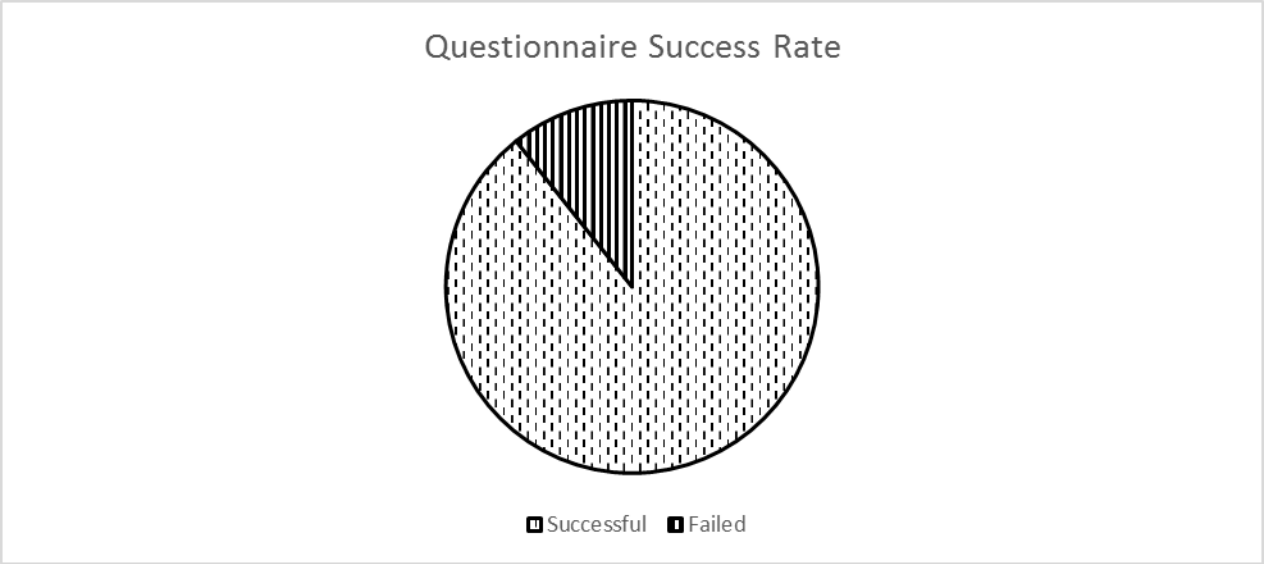
4.2 Overview of Findings

The study had 112 questionnaires handed out to the students of Gretsa University. Of these questionnaires, 109 were returned to the researcher representing a 97.3% return rate. Of the returned questionnaires, only 100 were usable with good data, representing 91.7% of the returned questionnaires. Overall, the questionnaires used for the study research were 100 out of the 112 handed out representing 89.3% success rate in questionnaire collection. This information is represented below.

Table 1: Questionnaire Return Rate

Research Action	Number	Percentage
Questionnaires handed out	112	100%
Questionnaires returned	109	97.3%
Usable questionnaires	100	91.7%

Figure 8: Questionnaire Success Rate



The research also found that the lecturers use the traditional teaching model to disseminate information to the learners. Mostly, the studies were theoretical and the use of hard copy printer reading material was the major method in which lecturers used to teach. In addition to that, the researcher also observed that the difficulties the students faced while learning were majorly centered around the concepts of OOP including Abstraction, Encapsulation, Inheritance and Polymorphism. This supported the fact that OOP concepts are challenging to learn. Adding to the difficulties experienced, the respondents also noted the lack of practical studies and low attendance of lectures which resulted to low grades. The respondents were found to understand OOP concepts moderately and properly understanding the other concepts of programming including syntax, semantics and logic.

4.3 Discussion of the Findings

4.3.1 Respondents' Profile

The total number of respondents for the study was 100 with 73% of the respondents being male, 25% being female and 2% not responding to the questions. Of the total, 30% were students

pursuing a bachelor's degree, 44% were pursuing a Diploma and 26% were in their certificate level. The study also acquired 30% of respondents as students doing Computer Science as a course and 70% being students pursuing Information Technology. Of the total respondents, 43% of the respondents were in their first year of study, 29% were in their second year of study, 27% were in their third year and 11% were in their fourth year. The first and second years saw the largest number of students since Certificate students only had roughly one year of study in their course.

When it came to their performance, the data collected showed that most of the students were average performers. This was observed with only 11% having attained an A in their grades, 25% having scored a B, 45% scoring a C, 9% scoring a D and 10% having failed in their current study level. Further, the study sought to establish the number of drop outs as observed by the respondents. Of the total respondents, 84% had observed a dropout rate of between 1 and 5 of their colleagues while 16% had observed a dropout rate of between 6 and 10 students. This data is represented in the consecutive tables and figures.

Table 2: Gender Distribution of Respondents

Gender	**Representation**
Male	73%
Female	25%
Rather not Say	2%

Table 3: Study Level of Respondents

Study Level	**Representation**
Degree	30%

Diploma	44%
Certificate	26%

Figure 9: Course Representation of Respondents

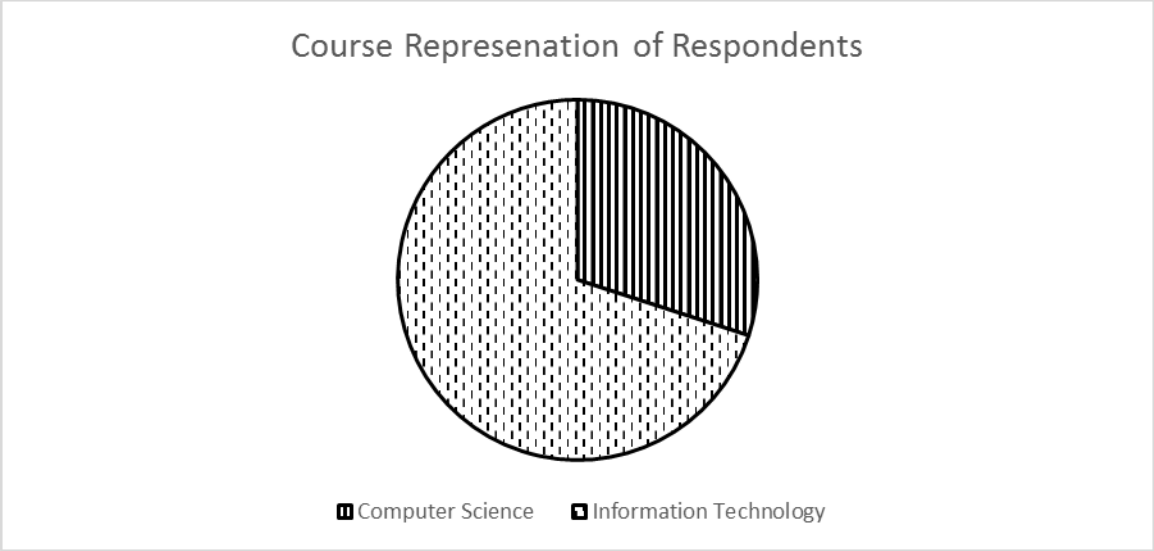


Table 4: Year of Study of Respondents

Year of Study	**Representation**
Year 1	43%
Year 2	29%
Year 3	17%
Year 4	11%

Figure 10: Performance Representation of Respondents

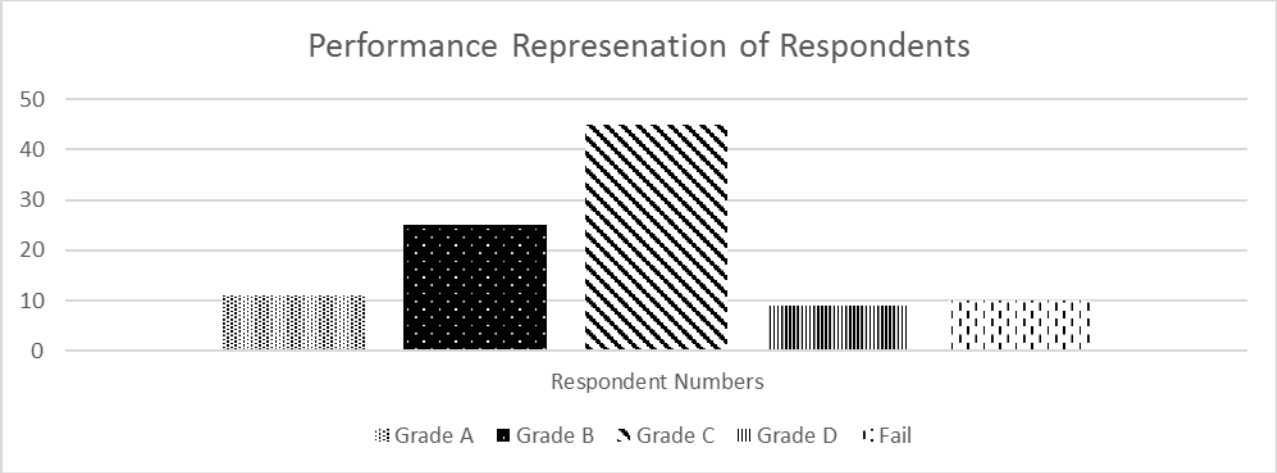
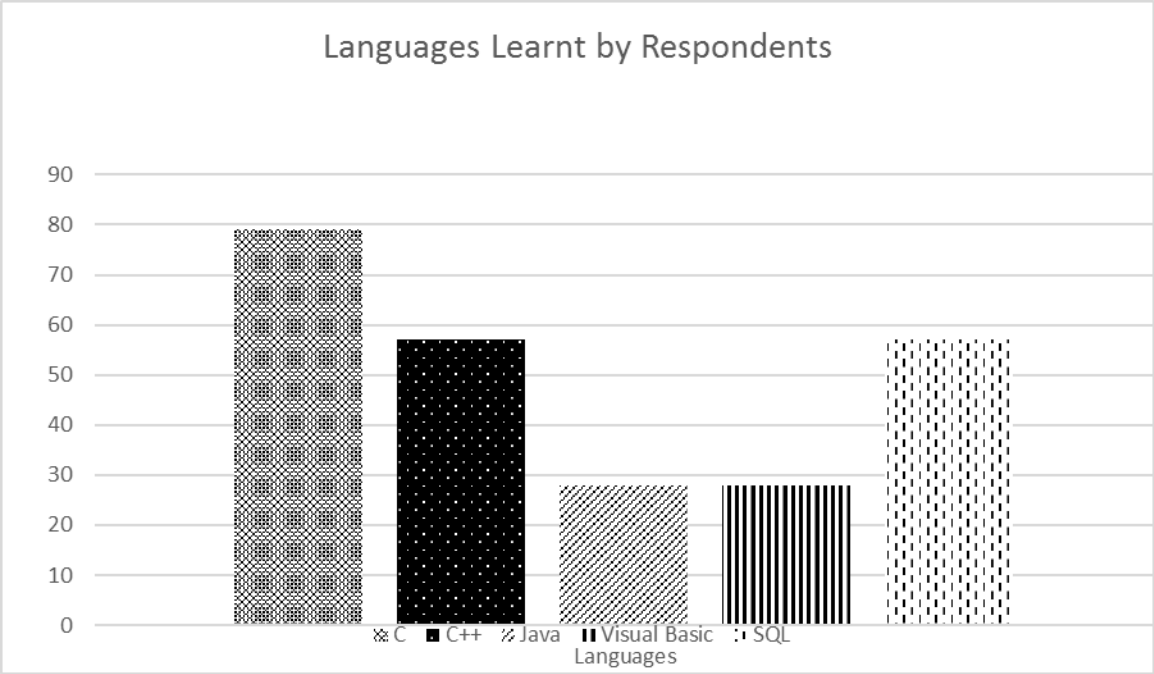


Table 5: Student Dropout Rates

Dropout Rage	Representation in %
0	0%
1 – 5	84%
6 – 10	16%
Over 10	0%

Figure 11: Programming Languages Studied by Respondents



4.3.2 Programming Concepts Difficulty

The main aim of this study was to understand the difficulties that were experienced by students as they learnt the various OOP concepts. These concepts included the four pillars of OOP which were; Inheritance, Abstraction, Encapsulation and Polymorphism. Further, the study included other aspects included in OOP including the syntax, program logic, variables, functions, arrays, loops and decision making. All these aspects were fundamental parts of all programming languages including Object Oriented Programming, Event Driven Programming and Structured programming.

The study collected data from the respondents as follows. On program logic, 84% of all respondents found it highly difficult to understand, 14% found it moderately difficult, and only 2% found it being easy. Syntax of OOP was found to be simple by 57% of the respondent, moderately hard by 37% and highly difficult by 4%. With regards to variables, 60% found it to be easy, 39% moderate and only 1% hard in terms of difficulty. Decision making was found to

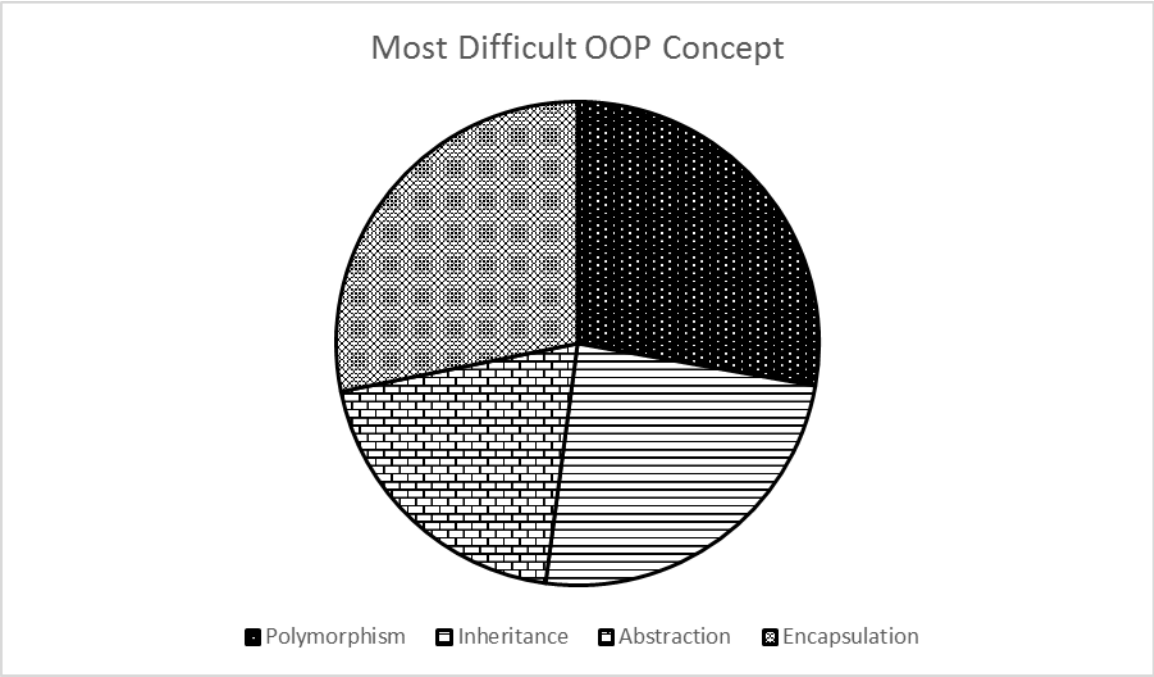
challenge only 28% of the respondents with 67% and 28% finding it moderately hard and simple respectively. Loops were found to be hard for 56% of the respondents, moderately difficult to 43% and simple for 1% of the respondents. Arrays were found to be moderately difficult for the majority of the respondents at 53%, hard to 43% and simple to 3%.

Methods were flagged as hard by only 27% of the respondents, with 69% saying they were moderately hard and 4% flagging them as simple. The concepts of OOP were found to be generally hard to understand with 72%, 63%, 51% and 73% finding the polymorphism, inheritance, abstraction and encapsulation as hard to understand respectively. This information is represented in the table and figure below.

Table 6: Difficulty in Understanding OOP concepts

Concept	**Low Difficulty**	**Medium Difficulty**	**Highly Difficulty**
Algorithms (Program Logic)	2	14	84
Syntax	57	39	4
Variable	60	39	1
Decision Making	5	67	28
Loops	1	43	56
Arrays	3	52	45
Methods	4	69	27
Polymorphism	1	27	72
Inheritance	9	28	63
Abstraction	13	36	51
Encapsulation	8	19	73

Figure 12: Exclusive OOP Concepts Difficulty



4.3.3 Lecture Data

Lectures have been a fundamental part of every course. For any students to pass, or to have been considered to have studied for a course, they need to have attended these lectures up to a certain percentage. In addition, lectures are required for the dissemination of information required for the student to learn. This section looks into data related to lectures.

The research found that a majority of the students who were involved in the study attended between 60 – 80% of all lectures required for their course. 3% attended above 80% of their lectures, 67% attended 60 – 80% of their lectures, 19% attended between 40 – 60% of their lectures, 9% attended 20 – 40% of their lectures and 2% attended between 0 – 20% of their lectures.

On the reasons that students had on their skipping of lectures, sickness ranked top at 53% with 18% sighting lack of interest, 11% sighting time issues, and 18% not specifying the exact reason for not attending the lectures. A majority of the respondents also felt that lectures did make it hard to understand concepts in OOP. 87% of the respondents agreed to completing assignments while 13% did not complete given assignments. On study materials used, most indicated that theoretical hard copy books were the largest mode of tutoring with practical programming taking only 32%. This information is summarized in the tables and figures that follow.

Table 7: Lecture Attendance Frequency

Frequency	**Percentage Representation**
80 – 100 %	3%
60 – 80 %	67%
40 – 60%	19%
20 – 40%	9%
0 – 20%	2%

Figure 13: Reasons for Skipping Lectures

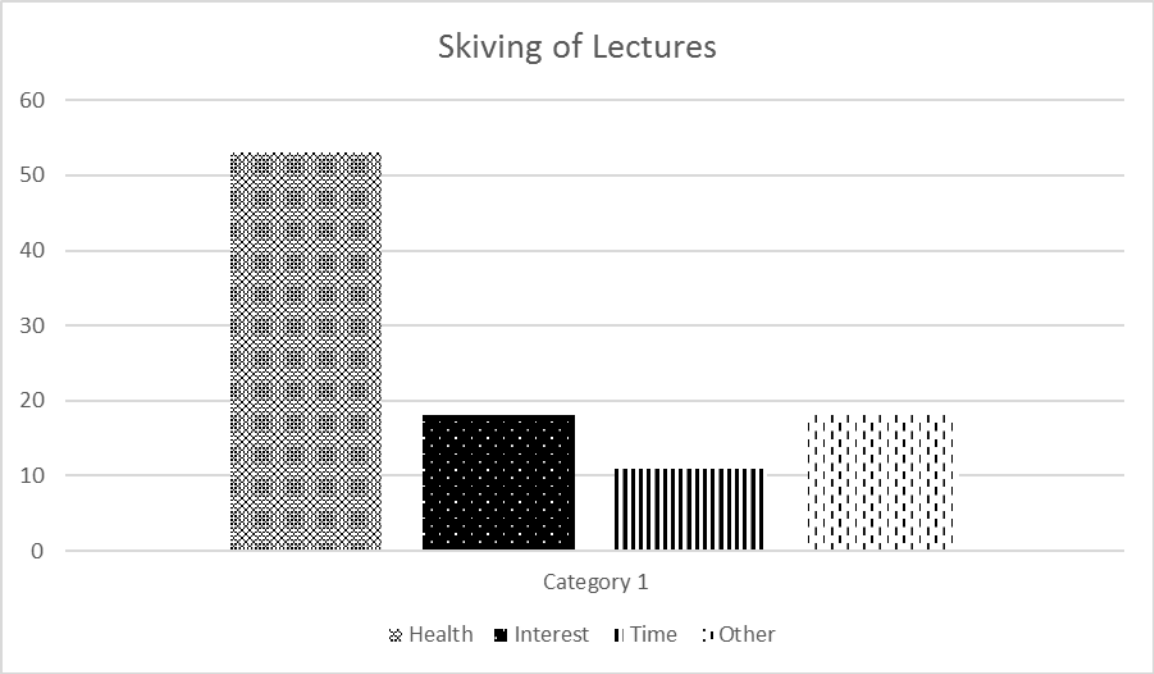


Table 8: Lecture Experiences in OOP

Experience	True	False
Lectures Improved understanding of concepts	41%	59%
Lectures made it difficult to understand concepts	37%	63%
Teaching model used disseminated information well enough	29%	71%
Lectures were too theoretical	81%	19%
Lectures lacked engaging study materials	74%	26%

Figure 14: Completion of Assignments by Students

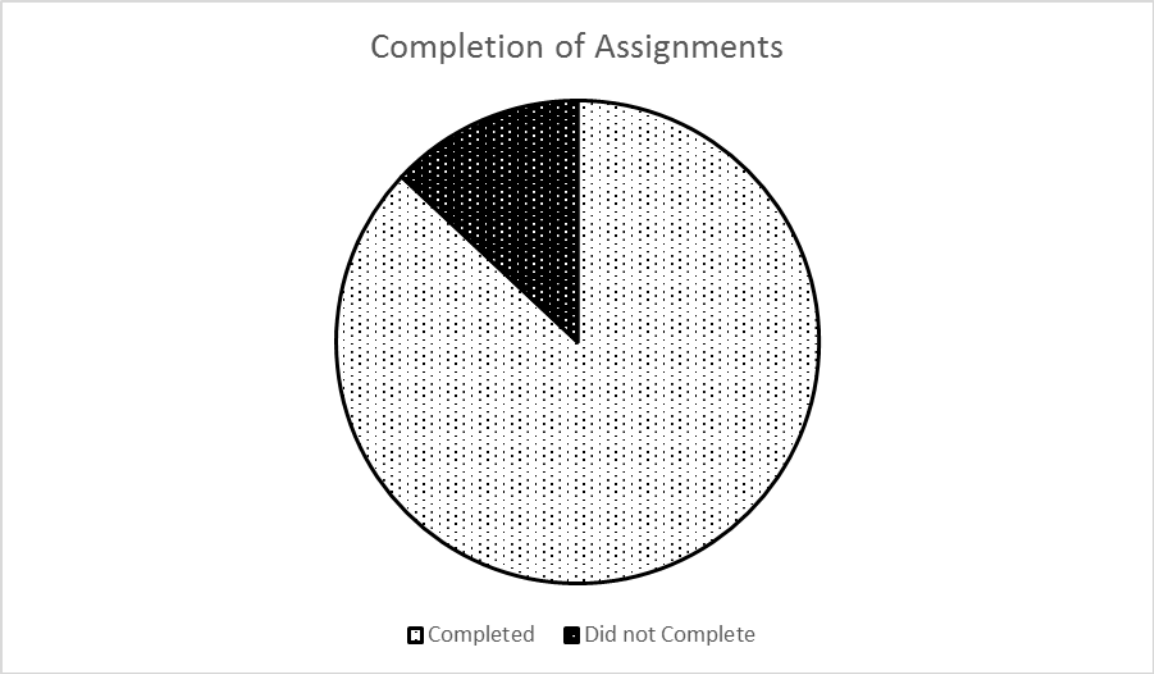


Table 9: Study Material Used save for Hard Copy Books

Material	**Representation**
Presentations	12%
Practical Programming	32%
Visual Aids	7%
Videos	1%

4.3.4 Personal Study Data

In addition to lectures, students had to make sure they added to the information they acquired through personal study. Of the respondents involved, 33% agreed to attending programming study groups while 67% did not. 76% said they did their own practice in programming while the rest declared they did not. Of the respondents, 9% said they never did programming practice weekly or daily, 32% said they did it daily and 59% declared they engaged in the practice at least once a week. The study also sought to find out why the current group had not dropped out of

study. 67% attributed this to passion, 20% were driven by the desire to acquire a job which 4% and 9% said they were coerced into doing the course and lack of a better course respectively.

Figure 15: Engagement in Programming Study Groups

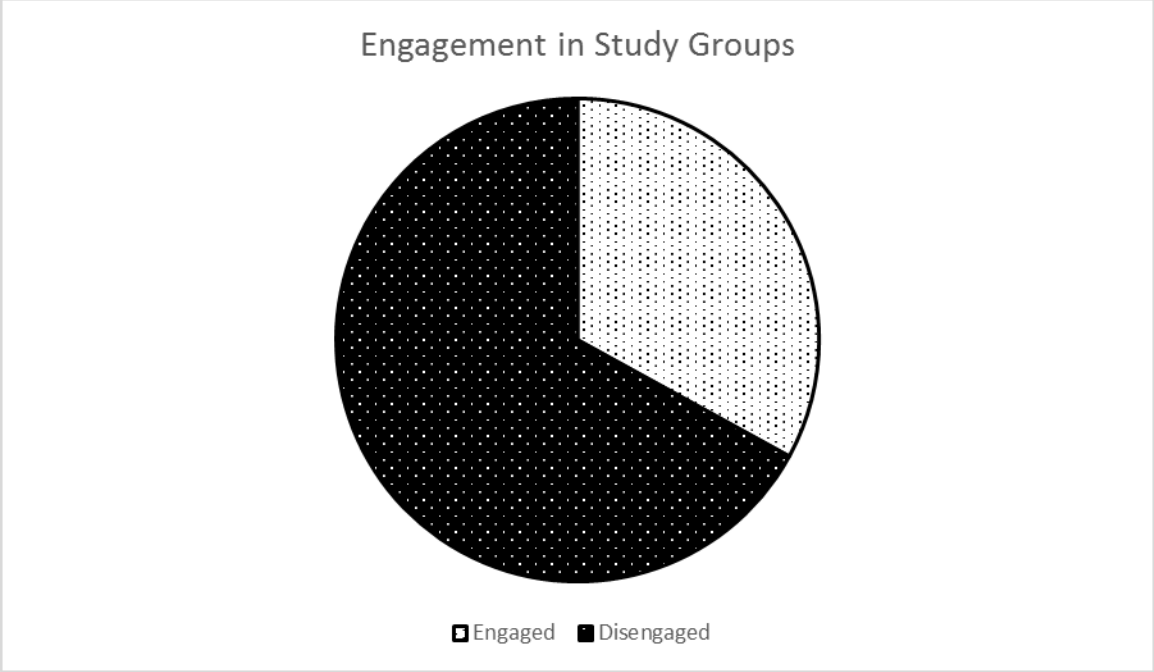


Figure 16: Programming Practice

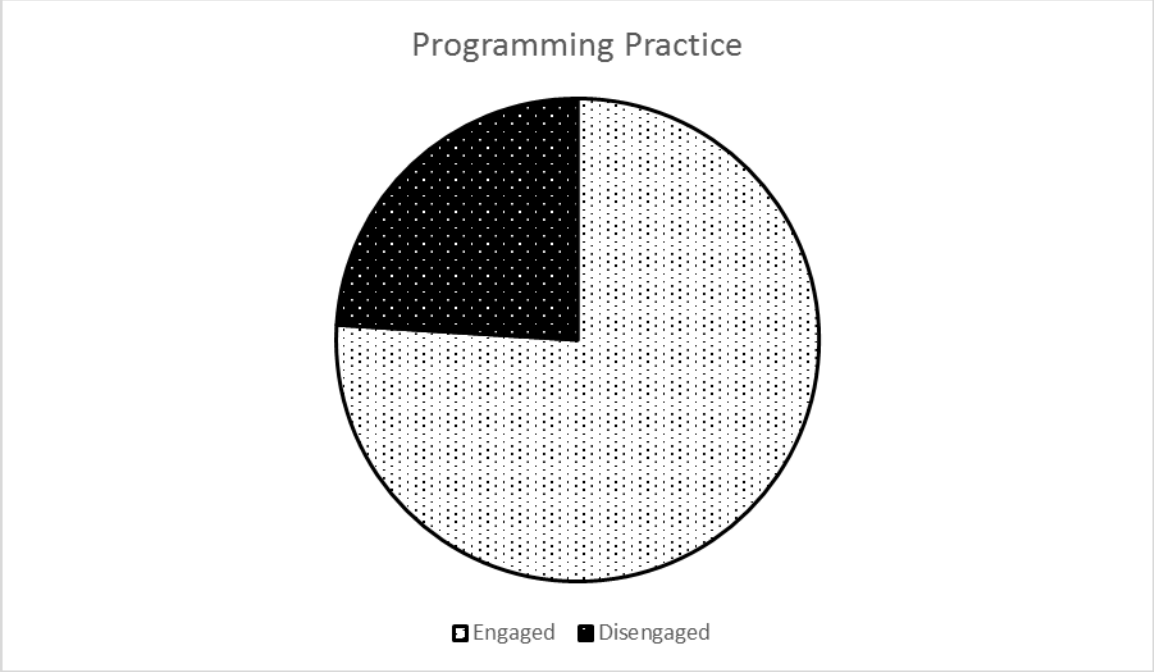


Table 10: Frequency of Programming Practice

Frequency	Representation
Rarely	9%
Weekly	59%
Daily	32%

Table 11: Reasons for not Dropping out

Reason	Representation
Passion	67%
Coercion	4%
Job Opportunities	20%
Lack of better course	9%

4.4 Critique of Other Studies

Studies reviewed in the literature of this study supported the findings the researcher gathered. (Matthew Butler et. Al., 2007) noted in their study that OOP concepts and Arrays represented a huge problem to the students with an average rating of 5.12 and 4.19 out of 7 respectively. These findings showed that the highest rated difficulties in the curriculum were conceptual in nature. Compared to the lowest rank in difficulty, the difficulty in concepts almost doubled it.

(PaiviKinnunen, 2009) noted that the attendance level for lecturers by the students directly determined the ability of the student to understand the concepts of OOP and consequently pass the examination. A higher attendance rate meant that there was s likelihood of understanding the concepts of passing the examinations.

(Gomes, et. Al., 2007) noted that the teaching methodologies used for programming languages were traditional and quite ineffective due to the nature of programming languages. This then lead to the students not understanding concepts and failure in development in programming. This this study, the researcher found the methodologies used were traditional and lacked engaging content and visual aids as well as enough practical exercises.

4.5 Inferences of the Study

The challenges that face students as they learn programming are many and variant on a number of factors including; teaching methodologies, student interest, attending of lectures, difficulty of concepts and prior knowledge on subject matters. These very factors led to the ability of the student to stay in the course without dropping out.

Programming languages that were considered difficult such as OOP were learnt from the second year of study through the fourth year. This formed a foundation for programming students due to first learning structured programming and programming logic as well as other related concepts. With time and understanding, this made it easier for the students to understand concepts in more advanced languages.

In OOP, the difficult concepts include Inheritance, Encapsulation, Abstraction and Polymorphism. These concepts coupled with fundamental concepts of all programming languages including logic, semantics, syntax and decision making and loops made programming difficult and hard to understand. They made it even more difficult because memorization of concepts was not feasible. The trend in memorization had been acquired for theoretical courses and it helped the students pass examinations and consequently succeed in their course work.

CHAPTER FIVE: SUMMARY, CONCLUSION AND RECOMMENDATIONS

5.1 Introduction

This chapter consists of the summary, conclusions and recommendations with regard to this research. These are covered in section 5.2 through 5.5.

5.2 Summary

The researcher pursued the research topic with the aim of understanding the challenges the students faced while learning programming languages, especially object oriented programming languages that included C++ and Java. This was done through the assessment of understandability of such programming languages.

The researcher used several mechanisms to gauge understandability and form an overall picture in the success of students with regards to OOP languages. These mechanisms include the understanding of concepts, the passing of examination through a look at their grades, the teaching methodologies used and the likelihood of dropping out of the course.

In the study, the researcher found that there was a need to modify the teaching model that was used in the teaching of object oriented programming languages. The traditional teaching methodologies were not found to be efficient. These methodologies focused on the memorization of theoretical concepts as opposed to understanding them. As a consequence, the students were not able to comprehensively apply the theoretical knowledge acquired for the purposes of developing their knowledge in OOP. The study found the use of visual aids, more practical lessons and other engaging teaching methodologies to be more beneficial to the students as opposed to traditional teaching methodologies.

The fault was not however on the teaching methodologies alone. The study found that there was a lack of enthusiasm in students acquiring more knowledge on their own through personal study and study groups related to programming. Personal study and study groups had the advantage of fostering what was learnt in class and expounding knowledge beyond curriculum coverage.

5.3 Conclusion

This study concluded that the understandability of programming was lacking for a majority of the respondents who were students. To advance their understanding of concepts, the curriculum had to implement a two-pronged approach in an effort to improve understandability of programming concepts. The first would be the improvement in the teaching methodologies to accommodate requirements for programming languages. The second would be the incitement of students to adopt self-imposed practices instead of waiting for curriculum imposed practices.

Learning programming has been observed to be inherently difficult, and the reasons for this difficulty is complex. Programming requires a set of skills, not a single skill to understand. Those skills form are required by the programmer at any point in time when they work on programming exercise. The basic idea would be developing a problem solving environment during the learning process. This would allow students to deal with programming issues as a set of logical problems requiring unique answers instead of problems requiring memorized solutions.

5.4 Recommendations for Practice

This study recommends a two-pronged change in the curriculum related to learning of programming languages. The first is in the teaching methodologies. In the teaching methodologies, the researcher would recommend the use of dynamic material to teach dynamic concepts. Further, the teaching should be focused more on the logic rather than the norm of

syntax teaching. Lastly, the teachers should try to implement the use of personalized teaching methods that take care of the needs of groups of students with the same understanding capacities.

The second change in curriculum would involve the students. The curriculum should implement ways in which students should be motivated into furthering their research save for tasks that are only done in lectures.

5.5 Recommendations for Further Research

Thestudy found that not a lot of researchers have focused in the challenges the students face as they learn programming in private universities in Kenya. Illuminating this concept could enable improvement of programming skills and overall productivity in the computers science industry through the advancement of curriculum. The study therefore recommends further research in challenges that face students as they learn programming, the level of understanding they have as they do their studies and teaching methodologies used to teach programming.

REFERNCES

Aska P. and Davenport D (2009). *An Investigation of Factors Related to Self-Efficacy for Java Programming among Engineering Students.* The Turkish Online Journal on Educational Technology. (TOJET), 8(1).

Caspersen, M. E. (2007). *Educating Novices in the Skills of Programming.* (PhD), University of Aarhus, Denmark.

Bennedsen, J., and Schulte, C. (2008). *What Does Object First Mean? – An International Study of Teacher Perception of Objects First.* 7th Baltic Sea Conference on Computing Education Research. Joensuu.

BusraOzmen and ArifAltun (2014). *Undergraduate Students' Experiences in Programming: Difficulties and Obstacles.* Turkish Online Journal on Qualitative Inquiry. July 2015, 5(3).

David Weintrop and Uri Wilensky (2015). *The challenges of Studying Blocks-based Programming Environments.* Learning Sciences and Computer Science. Northwestern University.

Eckerdal, A. (2009). *Novice Programming Students' Learning of Concepts and Practice.* Uppsala University. IS 1651-6214.

Gomes, Anabela and Mends A. J. (2007).*Learning to Program – Difficulties and Solutions.* Engineering Institute of Coimbra – polytechnic Institute of Coimbra.

Hawi, N. (2010*). Casual attributions of Success and Failure Made by Undergraduate Students in an Introductory-Level Computer Programming Course*. Computers and Education (54) 2010, 1127-1136.

Jiau, H. C., Chen J. C. and Su K. F. (2009). *Enhancing Self-Motivation in Learning Programming Using Game Based Simulation and Metrics*. IEEE Transcations on Education. 52(4), 555-562.

KirstiAla-Mutka (2007).*Problems in Learning and Teaching Programming.* Institute of Software Systems, Tampere University of Technology.

Lau, W. W. F and Yuen, A. H. K. (2011). *Modelling Programming Performance: Beyond the Influence of Learner Characteristics.* Computers and Education 55 (2010), 2018-228.

Law K., Lee V. and Yu Y. T. (2010). *Learning Motivation in E-Learning Facilitated Computer Programming Courses*. Computers and Education. 55(2010), 218-228.

Matthew Butler and Michael Morgan (2007). *Learning Challenges Faced by Novice Programming Students Studying High Level and Low Level Programming Concepts.* Faculty of Information Technology. Monash University.

PaiviKinnunen (2009). *Challenges of Teaching and Studying Programming at a University of Technology - Viewpoints of Students, Teachers and the University.* TKK Research Reports in Computer Science and Engineering A. TKK-CSE-A4/09 Espoo 2009

Sheard. J., Simon Hamilton, M and Lonnberg, J. (2009). *Analysis of Research into the Teaching and Learning and Programming.* The Fifth International Workshop on Computing Education Research Workshop. Berkely, CA, USA.

Thune, M. and Eckerdal, A (2009). *Variation Theory Applied to Students' Conceptions and Computer Programming.* European Journal of Engineering Education.

APPENDICES

Questionnaire

I am Ali Ibrahim Ali and I am conduction a research in the difficulties associated with computer programming, specifically in OOP. The data collected shall be purely for academic purposes and shall be kept confidential, private. This questionnaire shall take 15 – 20 minutes of your time.

Student Information

1. What is your gender? Male __ Female __ Prefer not to Say __
2. What is your level of study? Degree __ Diploma __ Certificate __
3. What course are you pursuing? _____
4. In what year of study are you? Year 1 __ Year 2 __ Year 3 __ Year 4 __
5. What is your average score in programming courses? A__ B__ C__ D__ F__
6. What number of students has dropped out of your class? 0__ (1-5)__ (6-10)__ Over 10__
7. What languages have you studies? C __ C++ __ Java __ Visual Basic __ SQL __
8. Which was the most difficult to learn? C __ C++ __ Java __ Visual Basic __ SQL __
9. Of the below outlined OOP concepts, note with either Low, Medium or High; the level of difficulty you experienced when learning.

Concept	**Difficulty**
Algorithms (Program Logic)	
Syntax	
Variables	
Decision Making	
Loops	
Arrays	
Methods	
Polymorphism	
Inheritance	
Abstraction	
Encapsulation	

10. What is your approximate lecture attendance with regards to programming units? (tick where appropriate)

Attendance Frequency	
80 – 100%	
60 – 80%	
40 – 60%	
20 – 40%	
0 – 20%	

11. What was the major determinant in skipping a lecture? _____

_____

_____

_____

_____

12. For the lectures you attended, which of the following statements describes your experience best?
    a. Lectures improved my understanding of concepts. True___ False___
    b. Lectures made it difficult to understand concepts. True___ False
    c. The teaching model used disseminated information well. True___ False___
    d. Lectures were too theoretical. True___ False__
    e. Lectures lacked engaging study material. True___ False__
13. Do you attend study groups with regards to programming? Yes__ No__
14. Do you finish assigned work by lecturers with regards to programming? Yes__ No__
15. Other than written, hard copy materials; what other materials do you use with regards to programming study in lectures? _____

_____

16. Do you do your own programming practice? Yes__ No__
17. If yes, how often? _____

18. What reasons have prevented you from dropping out of this programming course? _____

_____