Cape Peninsula
University of Technology

Perceptions of computer programming students on interactive environments for teaching Object-Oriented concepts using Java

by

PATRICK MWANSA

Thesis submitted in partial fulfilment of the requirements for the degree

Master of Technology: Business Information Systems

in the Faculty of Business and Management Sciences

at the

CAPE PENINSULA UNIVERSITY OF TECHNOLOGY

Supervisor: Dr. Michael Twum-Darko

September 2017

DECLARATION

I, Patrick Mwansa, declare that the contents of this thesis represent my own unaided work, and that the thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

Signed **Date**

ABSTRACT

The skill of programming necessitates knowing programming tools, problem solving and effective techniques of program design and implementation. Most students are incapable of fully understanding and utilising the feature set of Integrated Development Environments (IDEs). The feature set of certain IDEs comes with a lot of functionalities and students have to spend a lot of their time studying the features of the IDE without paying much attention to the syntax and semantics of the programming language.

The main objective of this study was to examine the perceptions of students on interactive environments for teaching Object-Oriented concepts using the Java programming language in two integrated development environments. This was done by adopting the ISO 9126 model to select generic external system quality characteristics and sub-characteristics that might influence student evaluation of an IDE. The proposed model was applied on NetBeans and JCreator LE 5.0 as IDEs for teaching Java programming using OOP concepts.

The study adopted a mixed method research approach using interviews and questionnaires. A single-case study was used for data collection and analysis. The approaches collected data from two groups of students using either NetBeans or JCreator and who were learning OOP concepts. The study further looked at the students' class tests and exam results in an effort to have an objective overview of how students performed. These groups of students were at two different campuses of the selected University. Each group had already been exposed to the Java syntax.

The result from this study was general guidelines to establish an interactive OOP development environment for teaching and learning of Java programming that enhances OOP comprehension.

This research study involved human subjects. It was, therefore, a requirement to seek ethics approval. Additionally, the objects involved were students of a selected University and as such a consent letter was sought from the University.

Keywords: Object-Oriented Programming, Java, NetBeans, JCreator 5.0, IDE

ACKNOWLEDGEMENTS

I would like to show gratitude to GOD, my creator, for giving me strength, health and life to soldier on through my studies.

I would sincerely like to thank my supervisor Dr. Michael Twum-Darko for his encouragement and guidance while working on this thesis. You were not only a supervisor, but an excellent role model for me. This work would never have been completed without you.

Additionally, I would like to thank, from the bottom of my heart, my wife (Melisa Mulenga), parents, brothers and sisters, for their support during the journey that marked this work. Their phone calls and emails have been a valuable source of encouragement and strength– especially when things were not going so well.

I also acknowledge the financial assistance from Walter Sisulu University. Opinions expressed in this thesis and the conclusions arrived at are those of the author, and are not necessarily to be attributed to the Walter Sisulu University and Cape Peninsula University of Technology.

DEDICATION

I dedicate this thesis to my parents Laban Musalula Mwansa (late) and Josephine Kombe Mwansa.

GLOSSARY

Abbreviations and Acronyms

OOP	Object-Oriented Programming
CS2	Computer Science Level 2
IDE	Integrated development environment
RCP	Rich Client Platform
SWT	Standard Widget Tool Kit
ISO	International Organisation for standards
GUI	Graphical User Interface
UI	User Interface
OSGI	Open Specification Group Initiative
LTK	Language Tool Kit

Definition of terms

JFace: The user interface toolkit with classes handling many common UI programming tasks.

ITICSE: A conference that has been held annually since 1996, in late June or early July. It has been held in many different countries.

Syntax: The structure of statements in a computer language.

Debug: The process of identifying and removing errors from computer hardware or software.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE : INTRODUCTION

1.1 Introduction

Students often face difficulties in using Integrated Development Environments (IDEs). Tasks relating to program compilation, debugging and execution are not easily accomplished. In addition, students cannot easily locate and interpret error messages that exist in the program code. Therefore, the ease of use is not realised and coupled with the non-existence of object-orientation support. Teaching Object-Oriented programming (OOP) concepts remains a challenge at many universities. Ala-Mutka (2012) reasons programming as being an art that includes knowledge of programming tools and languages, problem-solving skills, and also effective approaches for program design and implementation. A collective approach emphasises on teaching the basics of a programming language and then builds upon to more complex strategies of the whole programming process.

Teaching programming using OOP concepts requires a suitable programming environment. Many IDEs are available with different functionalities and feature sets. An IDE is a graphical user interface (GUI)-based workbench designed to aid a developer in building software applications with all the required tools combined in one environment (Techopedia, 2014).

Generally, many academics argue on the features of various OOP languages, but little attention is given to the programming environments. Today, several programming environments exist for OOP languages like Java. However, picking the right one that enhances OOP concepts comprehension for students remains a problem.

This study focused on the use of IDEs specifically developed for teaching and learning OOP in Java. Two IDEs, namely JCreator LE 5.0 and NetBeans, were used as an intervention to the current curriculum to observe students' reactions at the selected University with regard to ease of use, complexity and many other features that improves OOP understanding. However, many other IDEs exit that can be used to evaluate OOP comprehension, including other aspects that can assist or hinder students' learning of OOP. This research considered only students' perception of usability on IDEs and how it affects OOP comprehension but did not consider other factors that might influence usability of IDEs.

Consideration was given to the adoption of the ISO 9126 model for selecting generic external system quality characteristics and sub-characteristics appropriate for user evaluation that might influence the assessment of these programming tools. The motivation for using these two IDEs for this research was based on the availability as they are open source software and require no licence implications. Furthermore, the selected University has in previous years been using these IDEs for teaching and learning of Java programming. The emphasis was on students who had already been exposed to the syntax of Java programming language and were currently focusing on OOP concepts. This approach was aimed at establishing an interactive programming environment to be used in the teaching and learning of OOP concepts in Java.

The selected University is situated in the densely populated and generally under-developed far-eastern surroundings of South Africa's Eastern Cape Province. Most students originate from rural schools with deprived educational resources and are often understaffed with teachers (Magwashu, 2013). In addition, majority of students come from economically deprived homes, therefore most of these students have had little or no exposure to a computer in their first-year enrolment at university.

The study aimed at contributing to the curriculum development on how to teach object-oriented programming concepts using appropriate tools. This ascertained measures that provide solutions to the identified problems in the OOP teaching methods.

1.2 Background

More often than not, learning OOP concepts consists of a difficult mental challenge for students. It is due to this fact that this research was carried out to uncover the hindrances behind the phenomenon(Tan *et al*., 2009a). Furthermore, research studies show that most difficulties faced are those independent of the program paradigm, whereas other difficulties are associated with the particular attributes of each paradigm (Wahid *et al*., 2008).

Bennedsen and Caspersen (2007:32-36) claim that false views on failure and pass rates can have severe repercussions on the quality of introductory programming. This is because a lecturer with high failure rates tends to accept that "this is just the way programming courses are since all programming courses have high failure rates" and therefore not take any action to improve the performance. These false views also grow in the students' perceptions thus

affecting their performance, whereas there are actually underlying challenges that need to be investigated.

The IDEs that are used for coding OOP concepts normally produce syntactical errors that are both time and effort consuming as they are not easily displayed. Detecting their occurrence is almost unachievable coupled with execution problems. However, many misconceptions, if not most of them, have to do with things that are not readily visible, but hidden within the execution-time (Sirkiä & Sorva, 2012). Murphy *et al*. (2010) adds that debugging is problematic for novice programmers and further suggests a pairing of students approach. Nevertheless, if an IDE is not user friendly, the problem still remains unsolved.

In more instances than one, students often call for the attention of the lecturer to problems that relate to the IDEs, for example failure to debug or locate output screens of their compilations. Students are lost in these environments as they are developed for more professional users; they present an overwhelming set of components and functionalities. The effect is the same as having no IDE at all (Kölling *et al*., 2003).

The feature set of certain IDEs come with numerous functionalities and students have to spend a lot of their time studying the features of the IDE without paying much attention to the syntax and semantics of the Java language. This impacts on the loss of time to improve teaching and learning through the use of better tools (Kölling *et al*., 2003). Kölling *et al.* (2003) further add that such environments focus on building the Graphical User Interface (GUIs) which conveys a misleading picture of programming and object-orientation.

Nienaltowski, Pedroni and Meyer (2008) argue that providing more detailed compiler messages after program compilation does not seem to help novice programmers' comprehension, or help identify the error faster or better. However, integration of various information exchange into an IDE tool where richer ways for collecting, presenting and interacting with code are available would be helpful (Hartmann *et al.*, 2010).

Generally, the introductory courses in computer science follow the sequence that introduces students to programming basics with extra one or two courses teaching data abstraction / data structures. The first course is referred to as Computer Science 1(CS1) and the second as Computer Science 2(CS2) using the ACM's 1978 computing curricula (Hertz, 2010). However,

this study will be done on the CS2 course consisting of students doing OOP concepts in Java programming.

This research investigated the use of IDEs in teaching and learning of OOP. A comparative study was done on two IDEs using generic external system quality characteristics and sub-characteristics of the ISO 9126 model. The IDEs used include: JCreator LE 5.0 and NetBeans.

1.3 Problem Statement

1.3.1 Introduction

Some students are incapable of fully understanding and utilising the feature set of Integrated Development Environments (IDEs), thus affecting the comprehension of OOP concepts. The feedback that results from using these IDEs in response to error messages is also not sufficient (Nienaltowski *et al.*, 2008). One of the biggest problem for students learning to program is correcting syntactical errors (Denny *et al*., 2014). Furthermore, Settle *et al*. (2014) argue that motivation is perceived as significant in the usage of programming tools as students are not highly motivated to relate their existing goals to the tool (IDE) being used.

The activities in teaching computer programming that are proposed for students do not relate to their life activities and experience (Kaucic & Asic, 2011). Mostly a variety of tools for teaching students is not utilised. Xinogalos (2009) suggests the usage of multiple tools for teaching cognitively challenging subjects. Although there are currently formally accepted IDEs as teaching tools in use, which has facilities for creating, editing, compiling and testing Java programs, students' comprehension of OOP concepts still leaves a lot to be desired. The special characterised difficulties like programming techniques and the language that is used need to be resolved (Xinogalos, 2012). This is also to counteract the high failure rate that has not changed significantly overtime so as to meet the industry demands for good programmers (Watson & Li, 2014).Therefore, this research through a perception study produced guidelines that helped to identify an interactive OOP development environment for teaching and learning of Java programming.

1.3.2 Research objective

The main objective of this study was to examine the perceptions of students on interactive environments for teaching Object-Oriented concepts using the Java programming language in two integrated development environments. Given this objective, an attempt was made to

develop a relationship between IDEs quality of use and OOP concepts. Furthermore, this led to a framework that intends to address why this problem has existed. To address this objective the following questions were devised to tease out the factors likely to influence the election of an IDE to improve teaching and learning of OOP concepts using Java.

1.3.3 Research questions

The main research question to address the objective of this research is:

Does the quality of use of an IDE enhance the comprehension of object-oriented programming concepts through teaching and learning of Java programming language?

In order to answer this question, a number of secondary research questions have to be answered. They include the following:

(a) Does the use of an IDE affect students' understanding of OOP concepts?

(b) Does the use of a particular IDE increase students' confidence in learning Object–Oriented concepts?

(c) What are the most common mistakes and misconceptions students make during program development in a particular IDE?

1.4 Research Methodology

1.4.1 Introduction

In this study positivist research philosophy was used to determine the impact of quality of use of IDEs on students' comprehension of OOP. However, to enrich the data and analysis thereof, mixed method research approach was adopted as a data collection instrument. While the research philosophy is phenomenological, the study used interviews on a focus group of programming students during the research process. A large part of the interviews focused on the intended students' use of the selected IDEs using the Java programming language. The outcome of the interviews was used to develop themes which underpinned the design of the questionnaire. The survey questionnaire collected data that was used to establish the relationship of the data set. As secondary data, the study further looked at the students' class tests and exam results in an effort to have an objective overview of how students performed.

1.4.2 Case study

Robson (1993:146) defines a case study as “a strategy for doing research which involves an empirical investigation of a particular contemporary phenomenon within its real life context using multiple sources of evidence”. There are different types of case studies. Yin (2003) categorises case studies as explanatory, exploratory, or descriptive, and furthermore they can be differentiated between single, holistic case studies and multiple-case studies (Baxter & Jack, 2008). This study, however, followed a single case study approach. This was relevant as data was collected from two groups of students based on two different campuses as the unit of analysis. Each group was taught by a different lecturer using the same OOP curriculum in Java programming language but different IDEs.

1.4.3 Unit of analysis

The analysis was conducted on two groups of students studying Java programming at Computer Science level 2 (CS2) on two different campuses at the selected University. At this level, students are already exposed to OOP concepts and the two IDEs namely: JCreator LE 5.0 and NetBeans, only one IDE was used on either campus. A total population of 55 students with consideration of demographics were used. One group had a total of 34 students on one campus and 21 students on the other campus. Most of the selected students before university enrolment originated from rural schools with deprived educational resources (including use of computer technology) and are often understaffed with teachers and therefore are technologically disadvantaged.

1.5 Delineation of the research

This research was focused on students’ perception of usability on IDEs and establishing general guidelines to establish an interactive OOP development environment for teaching and learning of Java programming that enhances OOP comprehension through quality of use. The guidelines were based on the respondents’ participation and feedback.

The following limitations of the study should be borne in mind:

(a) This study focused on how students use programming tools to improve the understanding of OOP concepts;
(b) The study used only the Java programming language; and
(c) JCreator LE 5.0 and NetBeans were used as IDEs.

1.6 Research ethics considerations

This research study involved human subjects. Therefore, it was a requirement to seek ethics approval from the Cape Peninsula University of Technology (CPUT) ethics committee. Additionally, the objects involved are students of a selected University and as such, a consent letter was sought from the university used as a case study. This process was carefully explained to the students about what was expected of them so as to solicit their full participation and contribution.

According to Bazerman and Gino (2012:3), ethics is the behavioural method of the study of methodical and forecasted behaviour in which individuals make ethical conclusions and judgments of others that are in conflict with the institution and the assistance of the broader society. The data provided by the students of the selected University was sanitised (omission of student names and identification numbers) for confidentiality of respondents' information. An agreement was signed between the selected University and the Cape Peninsula University of Technology that the confidential information can only be for the intended purpose.

Confidentiality: The researcher assured the contributors confidentiality on the information provided by them.

Right to Privacy: The study valued the privacy of contributors and guaranteed that the description of the contributor's performance remains confidential.

Informed Consent: The researcher notified the contributors of the type of research being conducted and acquired approval from them.

Protection from harm: It was required that due to the type and subject matter of the questionnaire, no contributor was hurt. The researcher advised descriptively beforehand the subject matter and type of questionnaire to the contributor.

Honesty with professional colleagues: The researcher did not manufacture data to assist the results. The research conclusions were projected in a comprehensive and sincere manner.

Dignity: The researcher did not humiliate and mock contributors.

1.7 Justification and contribution to research

Conducting this research brought major contribution to curriculum development for OOP concepts. The contribution provided informed facts on the following:

(a) The problems students face while using IDEs;
(b) If the IDE "quality of use" has an effect on learning OOP concepts; and
(c) Misconceptions and mistakes students make when using the selected IDEs.

1.8 Overview of the chapters

This thesis is presented in the following chapters. Chapter 2 consists of the literature review on computer programming students' difficulties, Object-Oriented paradigm, student cognition and integrated development environments. Chapter 3 describes the research methodology, consisting of the rational of the method and the actual method. Chapter 4 presents the results analysis and major findings of this research. Conclusion and recommendations for future research study are presented in chapter 5.

CHAPTER TWO : LITERATURE REVIEW

2.1 Introduction

In the previous chapter, the fundamental reasons and the objectives of this research were delineated. The problem statement and the main research questions were also discussed. To recapitulate, the study examines the teaching and learning of OOP concepts in Java programming language using two software programming environments: JCreator LE 5.0 and NetBeans. The focus is to evaluate students' perception of usability on IDEs through quality of use using the ISO 9126 model. This in turn determines which environment best influences student interaction and enhances the comprehension of OOP concepts. The chapter further narrates on programming difficulties, Object-Oriented paradigm, student cognition and integrated development environments with due consideration given to the two selected IDEs.

The aim of this chapter is to provide an overview of the concepts in academic literature concerning the teaching and learning of object-oriented programming including the tools used. This is presented by topics to help in understanding and interpreting how the problems identified in the teaching and learning of OOP at a selected University is to be addressed by this thesis.

2.2 Difficulties of learning computer programming

According to Lister *et al*. (2004), students often do not know how to program at the conclusion of their introductory programming courses as reported in the study by the ITiCSE 2001 working group ("the McCracken Group"). This is because of their inability to problem-solve. A good explanation for this phenomenon is that students lack the ability to describe a problem which can be decomposed into sub-problems, implement them and then put them back together as a finished solution. Whalley *et al*. (2006) agree by stating that students lack the knowledge of basic programming constructs; however, those who are familiar with the constructs lack the ability to problem solve. On the contrary, Kinnunen and Malmi (2006) claim that students do not dedicate much time to programming and they mostly lack motivation. Furthermore, time constraints and class size at tertiary institutions result in students not receiving feedback or individual explanations and attention (Rogerson & Scott, 2010).

Experience has shown that students are faced with challenges when using programming languages to write computer programs that solve problems. This is a serious worry because of the high failure levels which in turn demotivates students. Therefore, traditional teaching approaches and learning methods are not appropriate for many students. In addition, Gomes and Mendes (2007a) postulate that since teaching is not personalised, the classroom environment needs to provide permanent student supervision. This can be achieved by allowing a tutor to monitor students evolution, clarify doubts and propose problems and activities. Tan *et al*. (2009b), however, describe students as being poor in mastering the skill because they start learning programming in single context before learning structure and style which leads to a negative programming habit.

Considering the observation of Tan *et al*. (2009b), one can conclude that a systematic approach which follows defined logical steps and trend can easily reinforce comprehension which this research intends to determine.

In the next section, the Object-Oriented paradigm is reviewed in order to understand the main aspect that leads to students having difficulties with learning computer programming. The Object-Oriented paradigm is one of the key computer programming features that students find difficult to easily comprehend.

2.3 Object-Oriented paradigm

OOP concepts are of paramount importance in the programming arena. Almost all universities globally offering computer science or Information Technology have programmes running with modules that include OOP concepts. However, students' understanding of the concepts has been the main obstacle. Besides working in a particular programming language, many aspects must be considered whether focusing on the syntax or the underlying paradigm (Börstler *et al.*, 2003). In contrast to these views, Lahtinen *et al*. (2005) argue that students tend to have problems in various aspects of program construction, thus it is important for students to do programming by themselves while they are learning. This should include carefully designed materials that lecturers can use to guide students' knowledge and skill construction. However, learning to program is related to the students' perception of what establishes program correctness (Stamouli & Huggard, 2006).

On the other hand, Eckerdal *et al*. (2005) claim that students need to learn fundamental abstract concepts and reach a certain level of understanding of these concepts for them to appreciate the Object-Oriented paradigm. Eckerdal (2009) further adds that students' learning of concepts is reached at different levels of granularity. However, Sorva (2013) suggests that unproductive behaviour and dysfunctional programs are as a result of incorrect and incomplete understanding of programming concepts.

When it comes to the topic of OOP concepts, this study postulates that independent learning will develop the skill of programming; however, most students lack self-motivation. Whereas some are convinced that students can learn and improve while they do it themselves, this study maintains that the environment in which students program in has adequate influence.

The next section gives an insight into students' cognitive skills. Cognitive skills are required as this helps students' ability to preserve information. The next section is relevant in order to give a picture of students' ability for OOP concepts retention.

2.4 Student cognition

Or-Bach and Lavy (2004) suggest that programming OOP concepts is a complex cognitive activity associated with a programming paradigm and hence it is not an isolated technical skill. Furthermore, it constructs a cognitive task analysis classification which deals with abstraction and inheritance, and further relates abstraction to Object-Oriented programming while at the same time determining the higher order cognitive skill which students find hard to conceptualise. In contrast, Vihavainen *et al*. (2011) argue that learning programming is efficient when beginners learn from people who already know the skill, thus cognitive apprenticeship focuses on learning by doing that encourages maximising coaching and guidance to students. On the other hand, Caspersen and Bennedsen (2007) state that optimisation in learning is a question of balancing and not maximising nor minimising the cognitive load.

According to Gaspar and Langevin (2012), educational research studies often have interventions distinguished by whether they affect pedagogy of content or pedagogy of instruction. The former is discussed as focusing on what topics are being taught, in what order, and how they link to one another, for example objects-first versus fundamentals-first. The latter is more concerned with how these topics are taught, for instance active learning versus traditional learning. Alternatively, Segedy *et al*. (2013) mention a systematic approach that is used to interpret and evaluate the

learner behaviour in open-ended learning environments using a model-driven assessment. The model uses cognitive and metacognitive processes important for completing an open-ended learning task. It shows that students employ several learning behaviours relating to solution construction and evaluation.

After looking at the learning behaviour, it is important to also discuss the programming tools. The next section delves into programming tools.

2.5 Programming tools

Many novice programmers interpret programming tools as error free and capable of doing everything. This is a misconception that tends to be detrimental as learners perceive any fallible and malfunctioning of such tools as their personal failure (Lee & Ko, 2011). This is because the first line of code beginners write often leads to unexpected behaviours, such as syntax errors, runtime errors, or program output that the learner did not intend. In addition, Storey (2005) suggests using adaptive interfaces which may perhaps be tailored to suit different kind of users and tasks. This is because software tools normally have many features which may be overwhelming not only to novice users, but also to expert users. On the other hand, Kasurinen *et al*. (2008) claim that students often lose interest in programming because complex models and structures have to be learned before anything visually impressive can be created.

Generally, lecturing object-oriented programming has mainly focused on showing the students the syntax and semantics of the language. Literature shows that many tools have been proposed to help address programming learning difficulties. Many of those tools use animation and simulation techniques, trying to take advantage of the human visual system potential. Inherent dynamic concepts are better understood using animation as compared to static formats (Gomes & Mendes, 2007b). Although most students are visual learners, lecturers are still inclined to present information verbally. Visual aids in the classroom improve learning by up to 400 percent and demonstrated by fact that 65 percent of the population are visual learners (Kydiam, 2012).

Carlisle (2009) argues that a large body of evidence has supported the idea that students understand programming concepts better when given a visual representation. This brought about the development of a visual programming environment for introducing object-oriented programming called RAPTOR.

Analysing Carlisle's view, one therefore poses the question: what then enhances students' understanding of OOP concepts?

In answering this question, van Haaster and Hagan (2004) drew the following conclusion:
"The underlying objective of educational software visualisation tools is to support student understanding of the mechanisms of software development. Visualisations can help students in numerous ways; for example, visual debuggers can help students to reconcile the cause and effect relationship between the source code that they write and the resulting output" (Van Haaster & Hagan, 2004: 455–470).

Although dynamic and interactive external representations have the potential to improve learning, they can also impede learning as they put pressure on the learner to consistently interpret various visualisation dynamics (Bodemer *et al.*, 2004).

Most research has shown the need for visualisations in the teaching and learning of object-oriented programming, though the distinction between the environment and the execution of a program must be clarified (Ragonis & Ben-Ari, 2005). But the question remains, does it improve the conceptualisation of object-oriented concepts?

In the next section, an in-depth description of IDEs is provided. This is to understand what IDEs are and how they function, including the internal system characteristic and how it responds to human interactions. This is important to understand before applying user evaluation.

2.6 Integrated Development Environments

An integrated development environment (IDE) is a packaged application software program that has a programming environment embracing a code editor, compiler, debugger and a graphical user interface (GUI) builder (Rouse, 2007). Olivero *et al*. (2011) claim that IDEs are tools that provide means to construct programs. The authors further add that IDEs create a platform for program comprehension of which certain IDEs generate an impediment to program understanding as they treat software elements as text, thus creating counter productiveness in program comprehension. IDEs are generally meant to provide an environment that makes program development easier in one integrated software. In contrast, DeLine and Rowan (2010) argue that the "bento box" design that partitions tools in distinct areas makes programmers disoriented in locating information that exist in different areas and putting it together. This also

causes interruption that requires extensive time and effort for a programmer to recover from (Parnin & Rugaber, 2012).

Figures 2-1, 2-2, 2-3 and 2-4 are relevant to this study to give us an understanding of various components that exists in IDEs. This examines the literature on the functioning and how IDE interfaces are generated. The abbreviations in the diagrams are provided in the glossary section of this thesis.

"Eclipse Software Development Kit (SDK) which is both the leading Java™ integrated development environment (IDE) and the single best tool available for building products based on the Eclipse Platform" (IBM, 2006) is shown in Figure 2-1 as a cloud depicting the various components of Eclipse IDE.

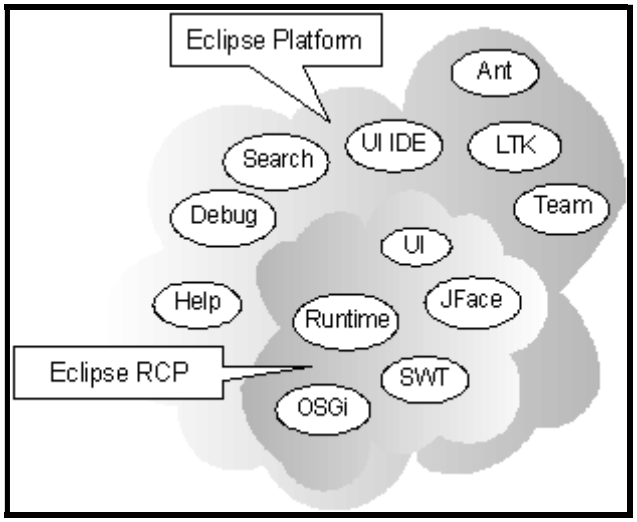

Figure 2-1: Eclipse Rich Client Platform (RCP) cloud with various integrated components (IBM, 2006)

Figure 2-2 shows the Eclipse platform with seamlessly-integrated tools that can be plugged in and be part of the integrated environment.

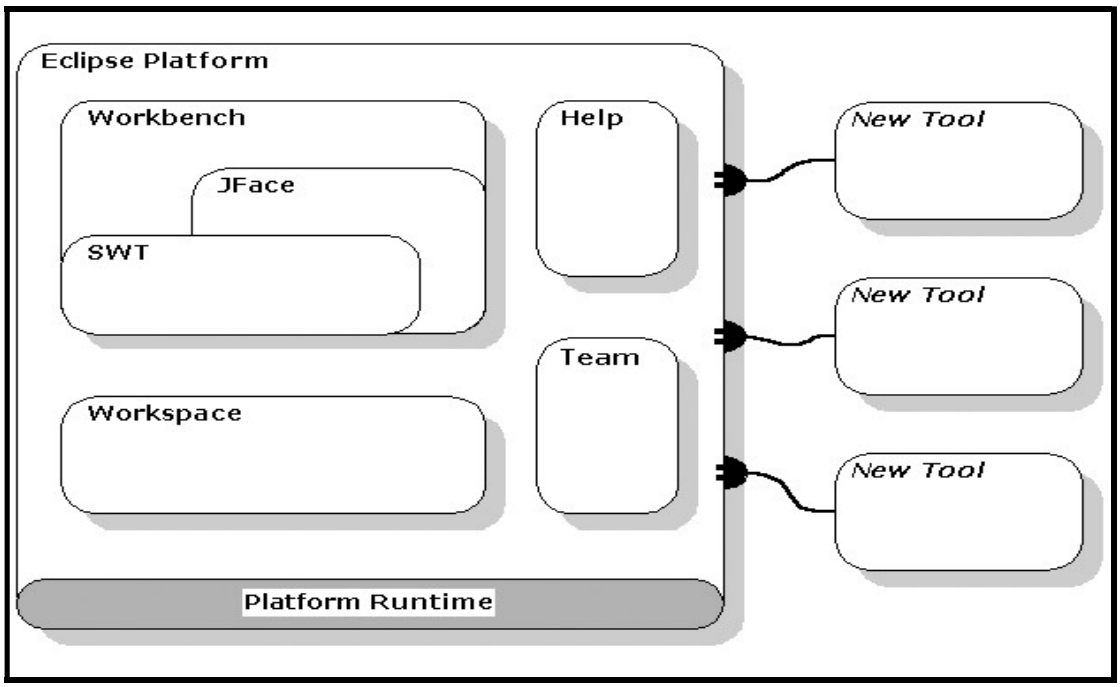

Figure 2-2: Eclipse Platform architecture "bento box" design that partitions tools (IBM, 2006)

Despite all the complications that come with the seamlessly-integrated tools, the partitioned design, however, offers various advantages that assist more especially experienced programmers to do their work. Salman (2010) articulates some advantages as follows:

- **Less time and effort**: The tools and features help you to organise resources, prevent mistakes and provides short cuts accordingly.
- **Enforce project or company standards**: IDEs allow programmers within a certain company to adhere to certain standards in the way they do things. Standards are enforced by templates and predefined libraries that are shared by programmers working on the same team.
- **Project management**: The IDEs encompass features that automate required tasks like documentation which is very helpful for entry programmers because of the visual presentation of resources that clearly outline the project and make easier management of the various tasks in the project.

2.6.1 JCreator LE. 5.0

In this case study, JCreator is one the IDEs to be assessed for quality of use. JCreator is a powerful IDE developed by Xinox Software Company. Unlike most IDEs, JCreator has two types of tools that can be configured. The first type is the Java Development Kit (JDK) tools. The user can use JDK tools to compile, debug, and run the project (Xinox Software, 2010). JCreator, like

any other IDE, also comes with a partitioned design with different tools in separate portions. In this IDE package, the following feature set includes:

- Advanced editor with code-folding;
- Popup for code completion;
- Popup for code snippets;
- Popup for code identifiers;
- Source code navigation;
- JSP, Ant and CVS support; and
- Feature rich Debugger.

The diagram in Figure 2-3 shows the JCreator interface that is used in this study.

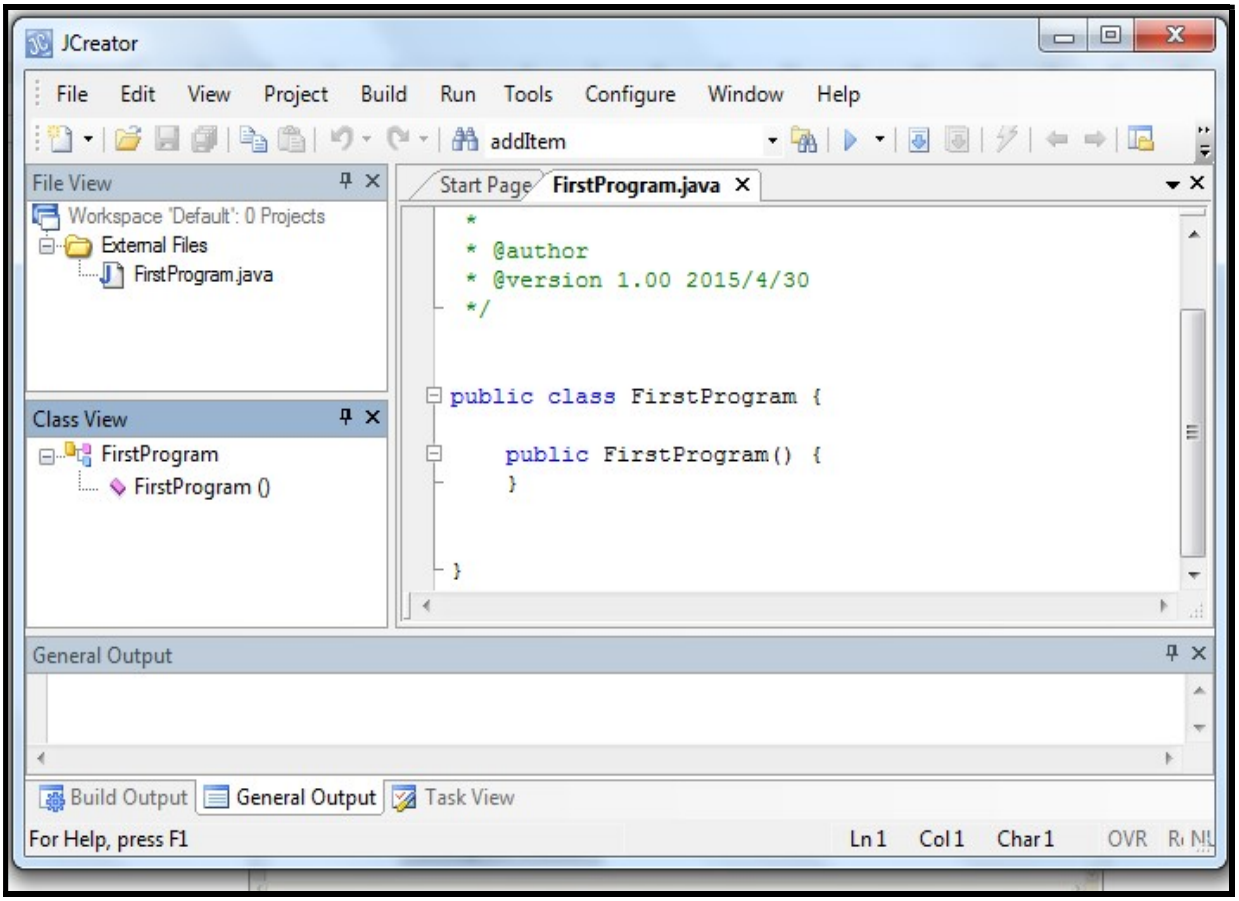

Figure 2-3: JCreator LE 5.0 "Bento box" Partitioned interface (Xinox Software, 2010)

2.6.2 NetBeans IDE

This study also looks at NetBeans IDE which is an integrated development environment available for Windows, Mac, Linux, and Solaris. The NetBeans project consists of an open-source IDE and an application platform that enable developers to rapidly create web, enterprise, desktop and mobile applications using the Java platform (ORACLE, 2013). Apart from its rich user interface, NetBeans allows:

- Best Support for Latest Java Technologies;
- Fast and Smart Code Editing;
- Easy and Efficient Project Management;
- Rapid User Interface Development; and
- Write Bug Free Code.

The diagram in Figure 2-4 shows the NetBeans interface that is used in this study.

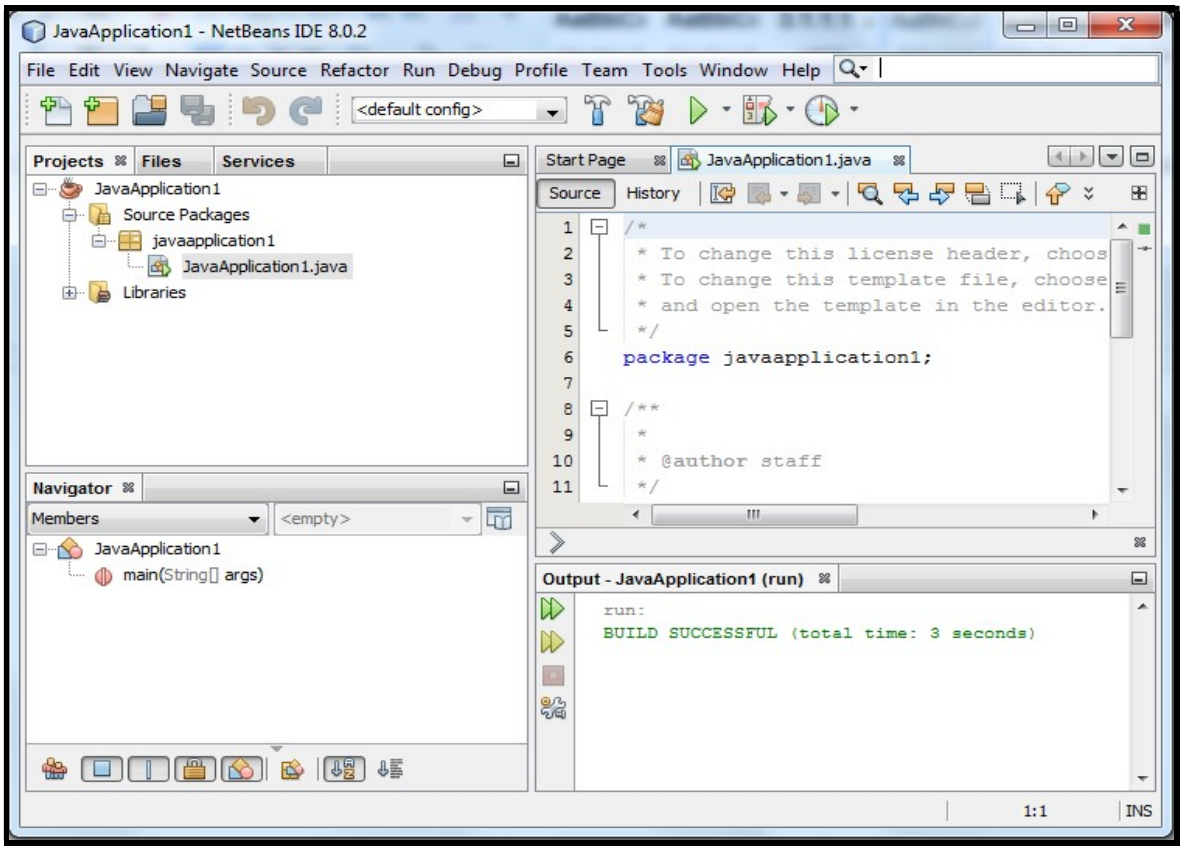

Figure 2-4: NetBeans "Bento box" partitioned interface (ORACLE, 2013)

Section 2.7 gives a description of the ISO 9126 software quality model. This model is significant for this study on account of its ability to be adopted and used for a specific software quality evaluation because of its generic nature.

2.7 ISO 9126 framework

Software quality models are necessary for evaluation and setting goals for software products quality (Zeiss *et al*., 2007). The international ISO/IEC standard 9126 defines a general quality model for software products. Padayachee *et al.* (2010a) claim that "Quality of use" is the user's view of the quality of software operating in an environment, and is measured by the results of using the software in the environment rather than properties of the software itself.

Although the ISO 9126 has been criticised as difficult to be made operational and non-practical (Kanellopoulos *et al*., 2010), it nevertheless fits as a model that can be customised because of its generic nature and therefore meets the intended evaluation for software quality for this study on IDEs.

The ISO 9126 model is defined by six software quality characteristics: functionality, reliability, effectiveness, usability, maintainability, portability and 22 sub-characteristics. However, to test student usage of IDEs and appreciation of OOP concepts, only four characteristics and seven sub-characteristics were tested on the two IDEs. The diagram in Figure 2-5 shows the ISO 9126 model and Figure 2-6 shows the adopted ISO 9126 model for this study.

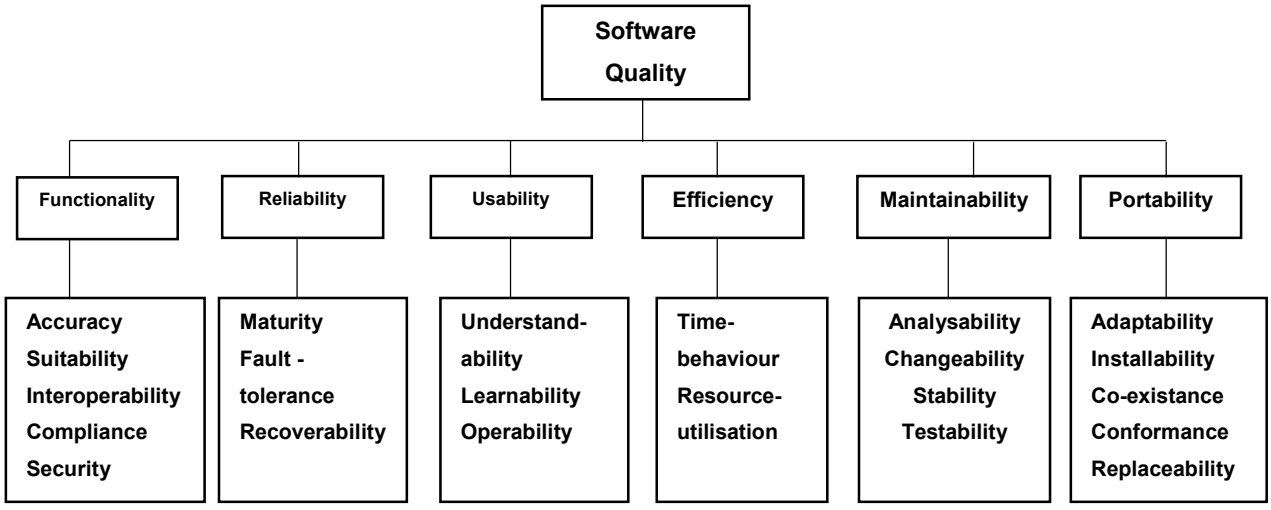

Figure 2-5: Software quality characteristics
(Bevan,1999 as cited by Padayachee *et al.*, 2010b)

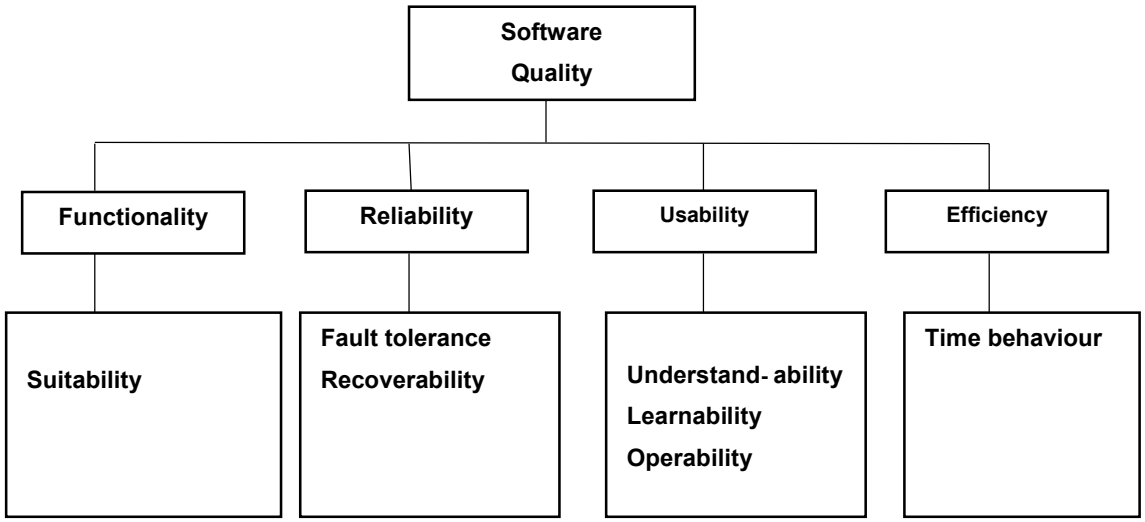

Figure 2-6: Adopted ISO 9126 model

The identified sub-characteristics were assessed on two IDEs and a comparative analysis was done to achieve the best interactive IDE. The other characteristics and sub-characteristics have been omitted in the adopted model as they do not fit user external quality evaluation for this study. Sub-characteristics like suitability aims at identifying if the software features are fit for use to encourage comprehension, learnability and operability. Time is an important factor in software use therefore assessing its response time plays a major role in software quality of use.

External and Internal software qualities are differentiated by the ISO 9126 and their related metrics. Internal qualities concern the intermediate deliverables such as source code. In contrast external qualities concern the behaviour of the computer system of which software is part of it (Seffah *et al.*, 2006). In this sense, metric of measurement such as reliability, efficiency, usability and functionality play a vital role in evaluating software use. On the other hand Alrawashdeh *et al.* (2013) lists a number of scholars that have used the ISO 9126 model metrics, the e-book system (Fahmy et al.,2012), website eLearning systems (Padayachee et al.,2010), computer based systems (Valenti et al.,2002) and government systems (Quirchmayr et al.,2007), claiming the generic nature of the ISO 9126 allows easy measurement of users perceptions using selected metrics that suits the objective of measurement. Hence using IDEs will require usability, functionality, efficiency and reliability as applicable metrics.

2.8 Problem conceptualisation

Given the literature surveyed thus far, the figure below illustrates how the research perceived the problem.

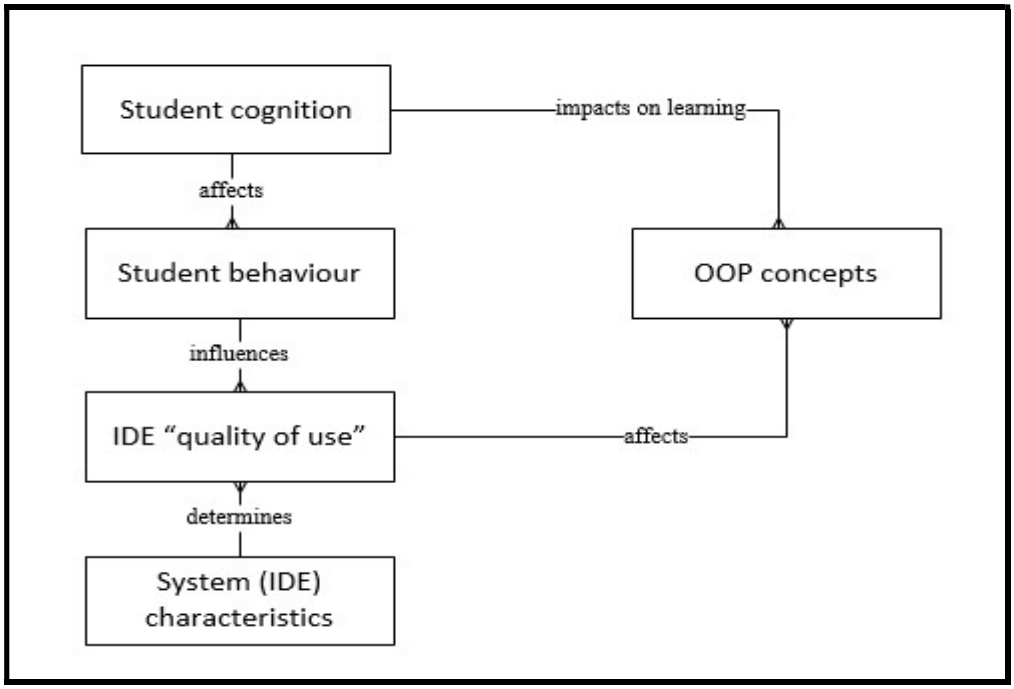

Figure 2-7: Teaching and learning OOP concepts

Figure 2-7 is as a result of the analysis of the ISO 9126 framework and the various literatures reviewed on OOP concepts and student cognition while learning to program. Therefore, the context of the study involved analysing students' cognition and how it affects their behaviour. Student behaviour also has a connection with the system "quality of use" and its response to student interactions or usage of the IDE. In addition, the OOP misconceptions faced by students and how OOP is interpreted in the IDEs were analysed. System (IDE) characteristics features play a major role in the evaluation of student (user) satisfaction. Figure 2-7 has, however, not been covered in various previous studies analysed in this literature. It nevertheless gives the basis of this study to establish guidelines of how quality of use influence OOP comprehension.

2.9 Summary

This chapter set out to review current literature on the nature of teaching and learning of object-oriented programming. This purpose was achieved by first reviewing the difficulties of learning computer programming, which was further supported by a review of Object-Oriented paradigm and student cognition with regard to their behaviour and conceptualisation of OOP concepts.

Secondly, the various aspects associated with programming tools and specific IDEs literature was reviewed. Lastly, a discussion on ISO 9126 as framework led to the development of a problem conceptualisation diagram called Teaching and Learning of OOP concepts on which this research is focused. This led to the identification of the research variables marked in Figure 2-8 with V, W, X, Y. “W” will therefore evaluate all the research variables including IDE quality of use, student behaviour and OOP concepts as shown in the diagram below.

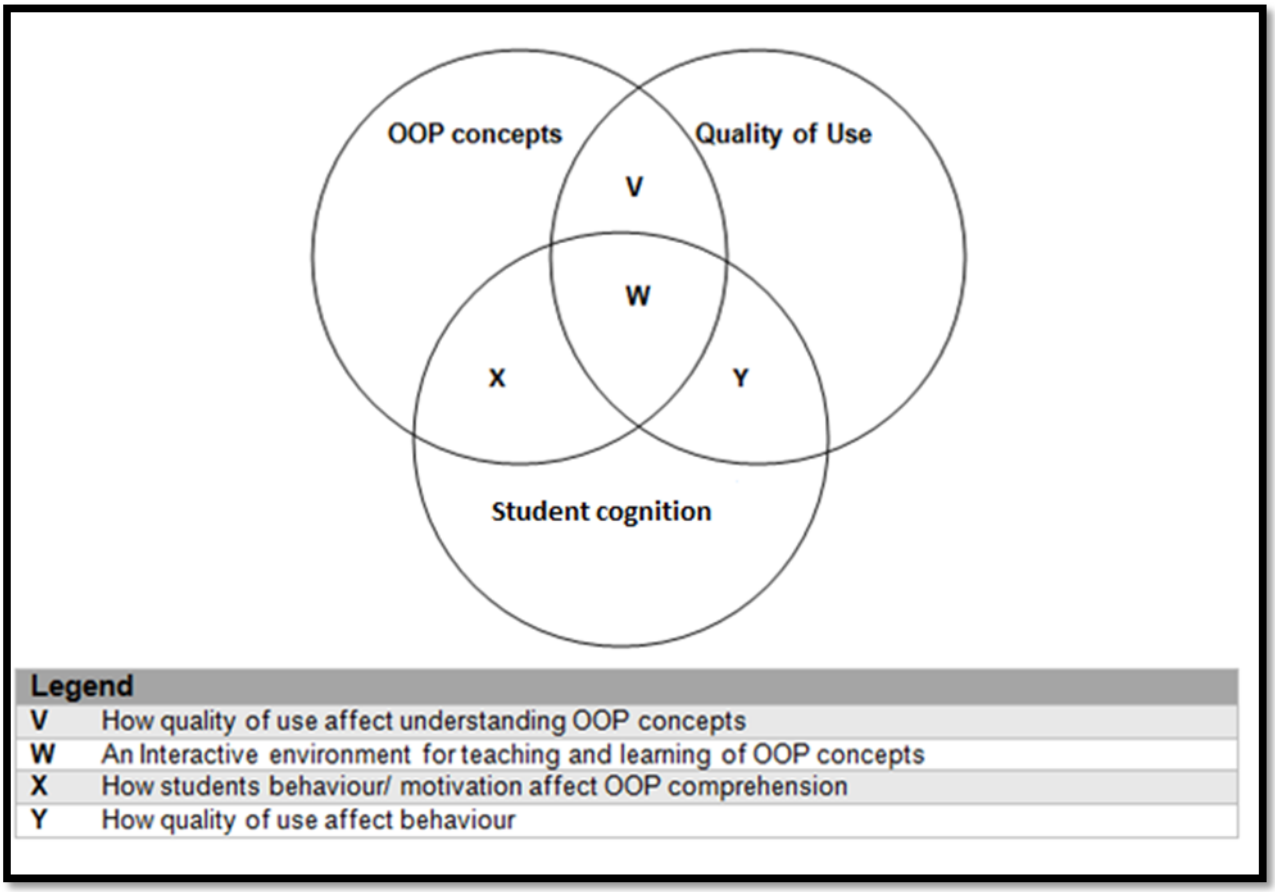

Figure 2-8: Research variables and outcome

In the next chapter (Chapter 3) a discussion of the research approach for this study is made; in addition, the selected University as a case study is further explained in depth.

CHAPTER THREE : RESEARCH DESIGN

3.1 Introduction

The previous chapter reviewed the relevant literature on teaching and learning of object-oriented programming and IDEs. It also presented the ISO 9126 model as an appropriate framework for evaluating perceptions of computer programming students on IDEs for teaching OOP concepts. This chapter describes the research design and methodology used in this study. The methodology was designed to answer the research questions and address the objectives of the study, as outlined in Chapter 1. The chapter apprehends and defines the context of the research philosophy, research design, research approach, data collection methods and data analysis techniques to ensure the reliability and validity of the research. Finally, data triangulation as used in the study is discussed. The research methodology is outlined in Figure 3-1 of the research onion.

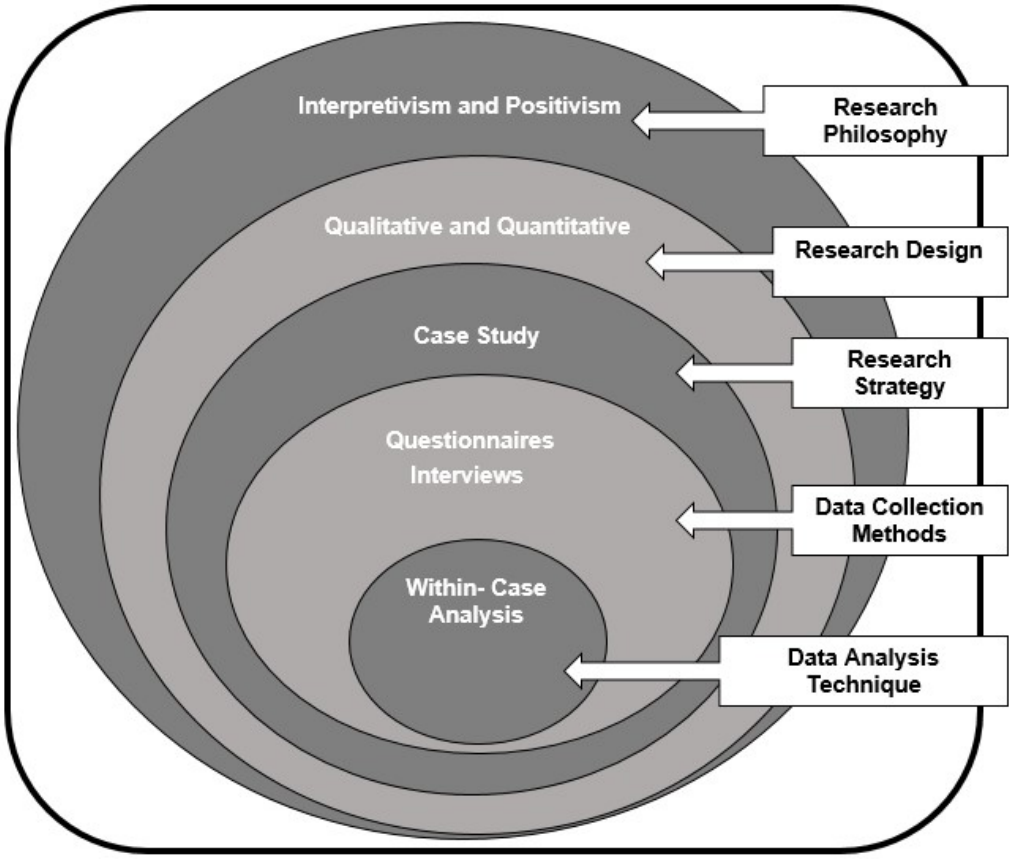

Figure 3-1: Research Onion (Saunders *et al*., 2009)

The parts of the research methodology are shown in different layers of the research onion in Figure 3-1 and will be elaborated further in the applicable sections of this chapter. The research onion illustrates the phases of the research process that was conducted in this study. Detaching each layer of the onion will reveal each part of the research process that was undertaken. The research onion has been engulfed with the research philosophy as the outer layer; this carries high relevance to this research. Flowers (2009a) articulates the importance of allowing various research paradigms and issues of ontology and epistemology as they refer to perceptions, beliefs, assumptions and nature of reality and truth to research of this kind.

3.2 Research philosophy

Information Systems as a field has utilised a wide range of research strategies with different fundamental philosophical paradigms to make people understand the use of Information Systems (Oates, 2005). Oates (2005) further defines a paradigm as a set of shared assumptions or ways of thinking about some aspect of the world, with different philosophical paradigms having different views about the nature of the world (ontology) and the ways we acquire knowledge (epistemology). On the other hand, Flowers (2009b) stresses the importance of considering different research paradigms as they describe perceptions, beliefs, assumptions and the nature of reality and truth (knowledge of that reality), this can influence the way in which research is carried out from design through to conclusion. In addition, Granell (2014) indicates research philosophy as guiding the researcher in the following aspects:

- Helps the researcher to refine and specify the research methods to be used in a research project, that is, to clarify the overall research strategy to be used. This would include the type of evidence gathered and its origin, the way in which such evidence is interpreted, and how it helps to answer the research questions posed.
- Enables and assists the researcher to evaluate different methodologies and methods and avoid inappropriate use of research methods and unnecessary work by identifying the limitations of particular approaches at an early stage.
- Helps the researcher to be creative and innovative in either selection or adaptation of methods that were previously outside his or her experience.

According to Creswell (2008), philosophical ideas still influence the practice of research and need to be identified. Creswell (2008) further suggests that researchers need to advocate the

philosophical ideas explicitly by the interconnections of worldviews, strategies of inquiry and research methods as shown in Figure 3-2.

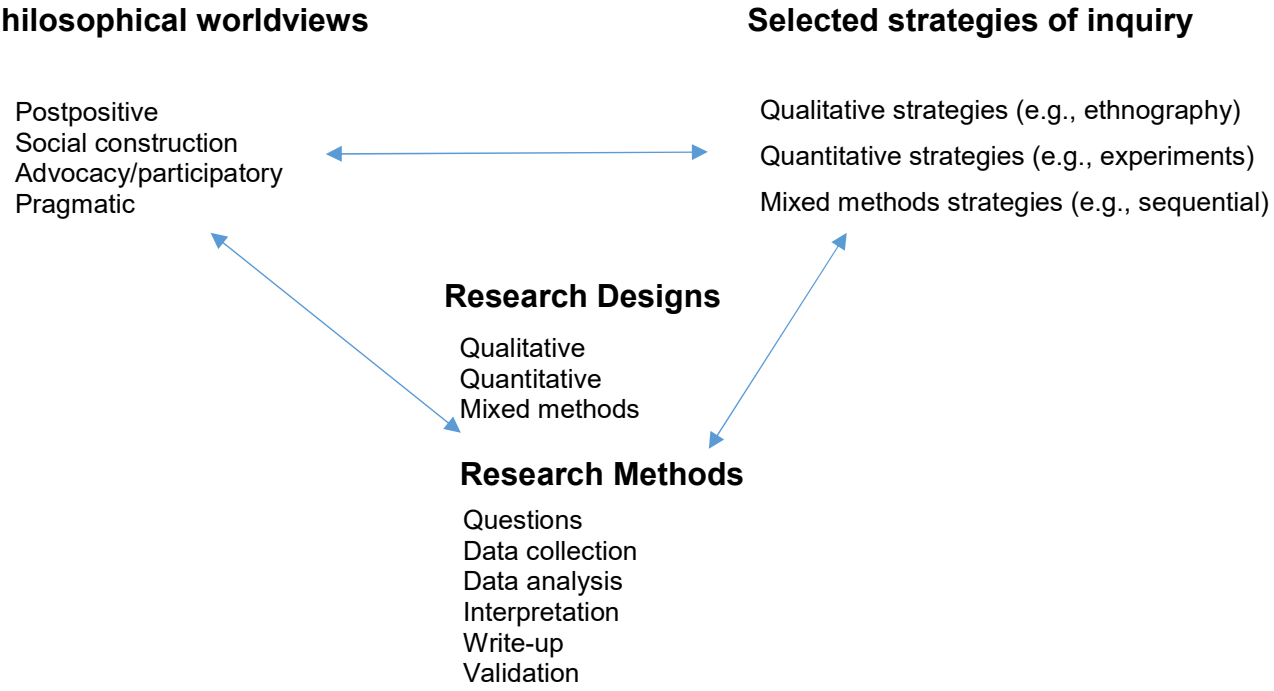

Figure 3-2:Framework for Design (Creswell, 2008)

The research onion model by Saunders *et al*. (2009) has identified ten philosophies. However, Oates (2005) and Creswell (2009) mention three philosophies as the most important for research in Information Systems namely: Positivism, Interpretivism and Critical research. The diagram in Figure 3-3 illustrates three philosophical assumptions.

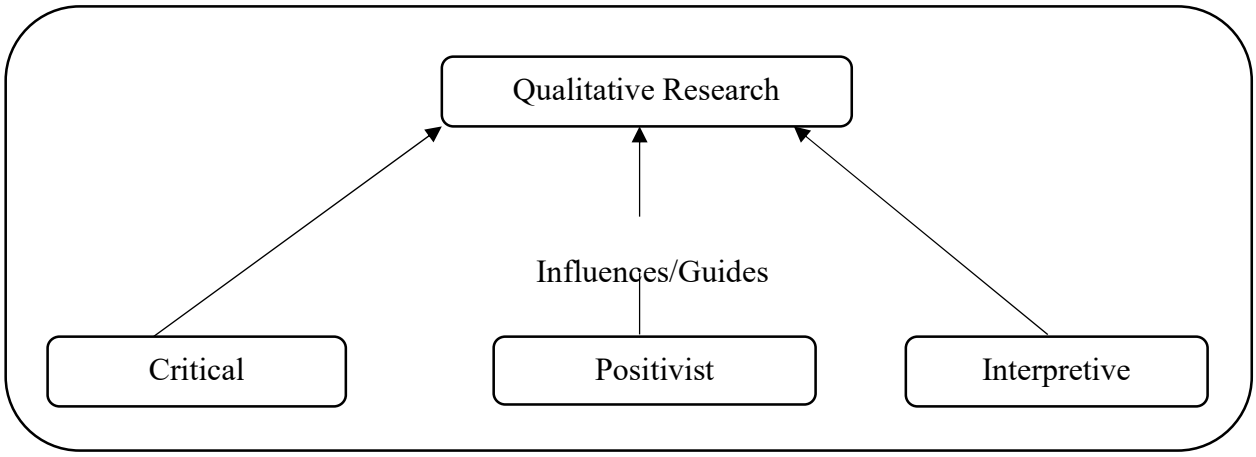

Figure 3-3: Philosophical assumptions (Myers, 1997)

3.2.1 Positivist

Gray (2013) defines **Positivism** as a philosophy which argues that reality exists external to the researcher and must be investigated through the rigorous process of scientific inquiry and is closely linked to **Objectivism.** In principle, Positivism argues the following:

- Reality consists of what is available to the senses – that is, what can be seen, smelt, touched, etc.
- Inquiry should be based upon scientific observation (as opposed to philosophical speculation), and therefore on empirical inquiry.
- The natural and human sciences share common logical and methodological principles, dealing with facts and not with values.

This will, however, provide generalisation of results to a larger degree and due to its quantitative nature, future predictions can be made. According to Johnson and Onwuegbuzie (2004:14-26), Positivism is "useful for obtaining data that allow quantitative predictions to be made". In contrast, Cohen *et al*. (2007) add that it fails to take account of our unique ability to interpret our experiences and represent them to others.

3.2.2 Interpretive

Interpretivism is closely linked to constructivism. Interpretivism asserts that natural reality (and the laws of science) and social reality are different and therefore require different kinds of methods (Gray, 2013). On the other hand, Oates (2005) describes interpretive research as being concerned with understanding the social context of an information system: the social processes by which it is developed and construed by people and through which it influences and is influenced by the social settings. Constructivism often addresses the interaction among individuals and focuses on the specific context in which people live and work in order to understand the historical and cultural settings of the participants (Creswell, 2009). In theoretical terms, Interpretivism sees the world as too complex to be reduced to a set of observable 'laws'. Generalisation is less important than understanding the real workings behind 'reality'.

3.2.3 Critical

According to Porter (2003), the essence of critical theory lies in its interest in the ways people think and act and how social circumstances influence those thoughts and actions.

In this study Positivism was used to identify a measurable quantity of students that either agree or disagree on the satisfaction of using IDEs. This could not have been achieved by an interpretive approach alone as it only focuses on the perceptions of how students interpret and feel about the quality of use. However, it was used to support positivism approach by adding on how students feel when using the selected IDEs. Nonetheless, Positivism and Interpretivism are the most common paradigms used (Altinay & Paraskevas, 2008; Dominick, 2006; Oates, 2005). The integration of these two paradigms provided a broader context to the students' perceptions and a better understanding of the different angles in which the research problem was handled. The study focused on a detailed understanding of a specific environment and groups of the selected University and may look to offer generalisable knowledge to other similar settings.

The bases for this study are the underlying theoretical paradigms which influence the reasoning and approach taken in this study. The research paradigm is also an indication of which school of thought (principles) the study is aligned to. Quite a number of philosophical paradigms exist; but for the purposes of this study the philosophical framework was narrowed down to the choice of Positivism supported by Interpretivism by using a qualitative approach through interviews on focus groups..

3.2.4 Qualitative approach

Flick (2007) defines and identifies qualitative approach as an intention to approach the world 'out there' (not in the specialised research settings or laboratories) and to understand, describe and sometimes explain social phenomena in a number of ways by:

- Analysing experiences of individuals or groups.
- Analysing interactions and communication in the making which can be done based on observing or recording practices of interacting and communicating and analysing the material.
- Analysing documents (text, images, film or music).

3.2.5 Quantitative approach

Quantitative approach is the process of testing objective theories by examining the relationship between variables where these variables can be measured, typically on instruments, so that numbered data can be analysed using statistical procedures (Creswell, 2008). Oates (2005) on

the other hand, defines quantitative data as evidence based on numbers which is generated by experiments and surveys.

3.2.6 Mixed methods approach

In this type of research, the investigator collected and analysed data, integrated the findings, and drew inferences using both quantitative and qualitative approaches in a single study or program of inquiry (Tashakkori & Crewwell, 2007 as cited in Teddlie, 2009).

In this study, mixed methods research was used to achieve the following as alluded to by Creswell (2012):

- When both quantitative and qualitative data, together, provide a better understanding of your research problem than either type by itself.
- When one type of research (qualitative or quantitative) is not enough to address the research problem or answer the research questions.
- To incorporate a qualitative component into an otherwise quantitative study.
- To build from one phase of a study to another.

Establishing guidelines for an interactive environment that enhances OOP concepts comprehension with quantitative data only could not have provided sufficient data to incorporate the students and experts views. Therefore, qualitative data was needed to provide more information regarding students' motivation towards learning OOP and capture all the research variables that were identified through problem conceptualisation.

3.3 Research design

Thyer (1993) defines traditional research design as a blueprint or detailed plan for how a research study is to be completed by operationalising variables so that they can be measured, selecting a sample of interest to study, collecting data to be used as a basis for testing hypothesis, and analysing the results. In addition, Kerlinger (1986) delineates a research design as a plan, structure and strategy of investigation conceived in order to obtain answers to research questions and problems. Therefore, through a research design one can attain the following functions:

- Conceptualise an operational plan to undertake the various procedures and tasks required to complete your study (Kumar, 2005).

- Ensure that these procedures are adequate to obtain valid, objective and accurate answers to the research questions (Kerlinger, 1986).

3.3.1 Research process

The research process outlines the summary of the various stages undertaken to achieve the objectives. The diagram in Figure 3-4 depicts the process followed for this study.

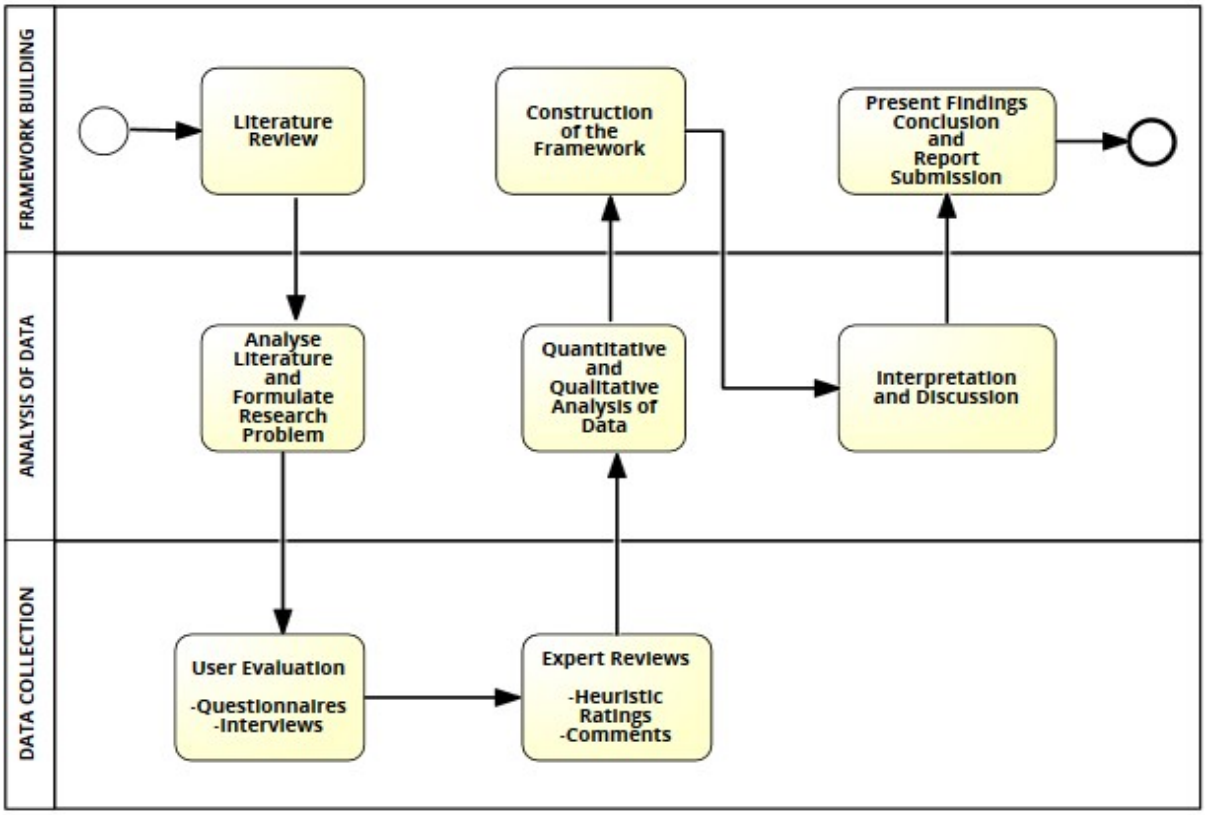

Figure 3-4: The Research Process

The research design in this study is developed, based on the following problem statement, objectives and research questions:

3.3.2 Problem statement restated

Some students are incapable of fully understanding and utilising the feature set of Integrated Development Environments (IDEs), thus affecting the comprehension of OOP concepts. The feedback that results from using these IDEs in response to error messages is also not sufficient (Nienaltowski *et al*., 2008). One of the biggest problems for students learning to program is syntactical errors (Denny *et al*., 2014). On the other hand, Settle *et al.* (2014) argue motivation

as having perceived significance in the usage of programming tools as students are not highly motivated to relate their existing goals to the tool (IDE) being used. If this is not addressed, universities will continue to produce programmers who are not fully capable of perfectly developing and deploying a software system that solves problems.

3.3.3 Research objective restated

The main objective of this study was to examine the perceptions of students on interactive environments for teaching Object-Oriented concepts using the Java programming language in two integrated development environments. Given this objective, an attempt was made to develop a relationship between IDEs and OOP concepts. Furthermore, this led to a framework that will address why this problem exists. To address this objective, the following questions were devised to tease out the factors likely to influence the selection of an IDE to improve teaching and learning of OOP concepts using Java.

3.3.4 Research questions restated

The main research question that addressed the objective of this research is:

Does the quality of use of an IDE enhance the comprehension of object-oriented programming concepts through teaching and learning of Java programming language?

In order to answer this question, a number of sub research questions have to be answered. They included the following:

- Does the use of an IDE affect students' understanding of OOP concepts?
- Does the use of a particular IDE increase students' confidence in learning Object–Oriented concepts?
- What are the most common mistakes and misconceptions students make during program development in a particular IDE?

The research design that was used during this research study is illustrated in Figure 3-5.

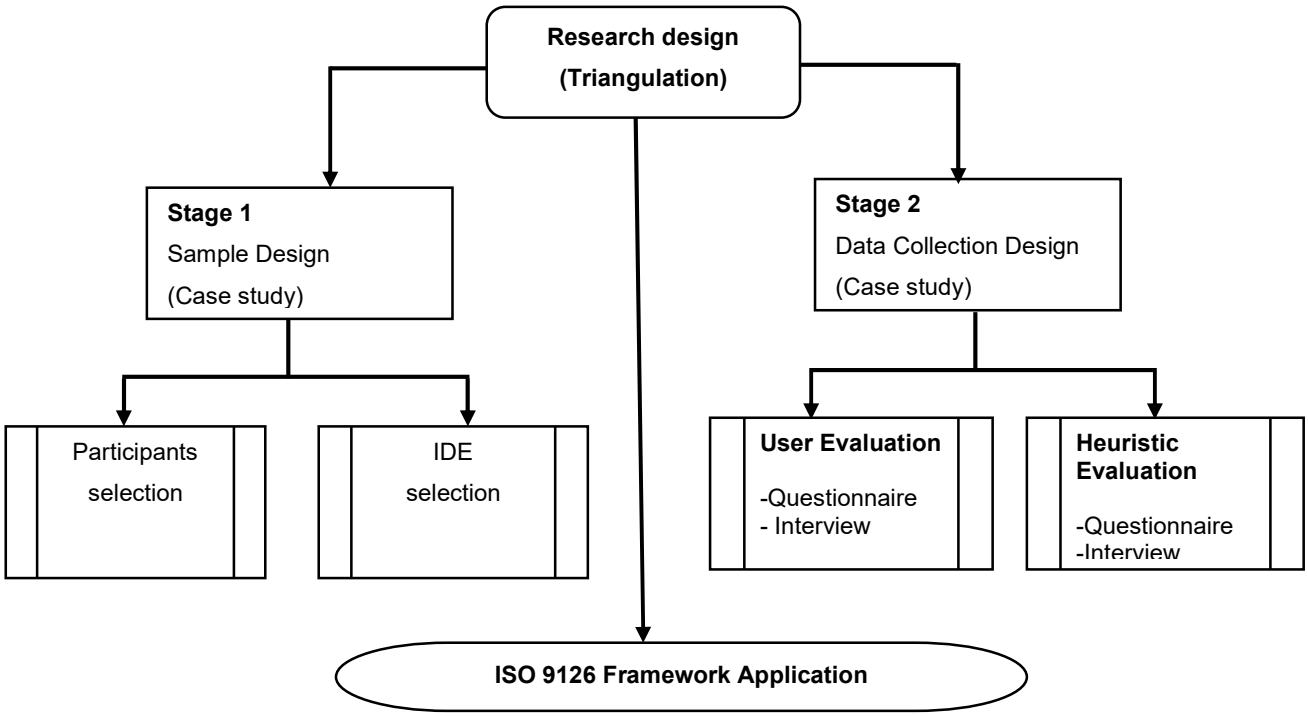

Figure 3-5: The Research Design overview

The research study started with an investigation into teaching and learning of object-oriented concepts that helped define its background. It was aimed at identifying the existing challenges that students encounter with respect to utilising the IDEs, and particularly OOP comprehension. This helped to identify and define the research questions (cf. Chapter 1, Section 1.3.3) which were answered after the completion of this study. This was presented as part of the research proposal for this study.

The aim of the second phase was to perform the literature review to provide a clear understanding of the problems students face when learning computer programming. It also provided definitions and discussion on the object-oriented paradigm with due consideration of students' cognition while learning to program focusing on the selected University. This phase also aimed at identifying various programming tools and their role in assisting OOP comprehension. In addition, an ISO 9126 framework was discussed which led to the adoption of external sub-characteristics to be used in the testing of students' perceptions of usability of selected IDEs.

After conducting the literature review, the next step concentrated on:

- Developing a problem conceptualisation framework as perceived by this study to identify the research variables involved and also to guide this research towards developing general guidelines to assist on selecting which IDE enhance OOP comprehension;
- Identifying and selecting the two applications (IDEs) that meet the criteria set by the researcher, and
- Identifying the participants to evaluate the selected applications. Similar to the application, the participants had to meet the criteria set by the researcher. The participants were separated into two groups: the users and the experts (students and Java lecturers).

The third phase concentrated on the development of the case study in the form of testing students' perception of usability in using the selected IDEs and how it relates to OOP comprehension, whereby students evaluations and expert reviews were obtained. After completing the students' perception of usability evaluations, phase four of this research aimed at analysing and interpreting the data collected from both students' perception of usability evaluations and expert reviews, and the study further looked at the students' class tests and exam results in an effort to have an objective overview of how students performed.

These interpretations were used in the final phase (phase five) of this research study, which aimed at recommending general guidelines to establish an interactive OOP development environment for teaching and learning of Java programming that enhances OOP comprehension through quality of use.

3.4 Research strategies

Saunders *et al*. (2009:600) define research strategy as "the general plan of how the researcher will go about answering the research questions". On the other hand, Bryman (2012:22) identified research strategy as "a general orientation to the conduct of research". There are various different research strategies with distinctive characteristics available from which a researcher may select. In Saunders *et al*.'s (2009) research onion survey, case study, experiment, grounded theory, action research and ethnography are listed. From these various strategies, this research sought to adopt the case study research strategy as the appropriate strategy for research. The following sections briefly describe the case study strategy and justify its preference as opposed to other strategies.

3.4.1 Case study

According to Robson (1993:146), a case study is “A strategy for doing research which involves an empirical investigation of a particular contemporary phenomenon within its real-life context using multiple sources of evidence”. Case studies can be done in different types. Yin (2003) categorises case studies as explanatory, exploratory, or descriptive and furthermore they can be differentiated between single, holistic case studies and multiple-case studies (Baxter & Jack, 2008).

The three case studies as categorised by Yin (2003) are:

- **Exploratory study**: This is used to define the questions or hypotheses to be used in the subsequent study.
- **Descriptive study**: This leads to a detailed analysis of a particular phenomenon and its context.
- **Explanatory study**: This explains why events happened as they did, or why particular outcomes occurred.

Whether exploratory, descriptive or explanatory, a case study can be a single case or multiple cases (Oates, 2006:144; Yin, 2003).

- **Single case**: This examines one case only.
- **Multiple cases**: These examine more than one case. The researcher must look at any similarities between the cases.

The following indicate a case study research approach process as specified by Simons (2009) and Yin (2003):

- To determine and define the research questions;
- To select the cases;
- To perform the sampling;
- Selection of data collection and analysis techniques;
- Prepare and collect data; and
- Evaluate and analyse the data collected.

Given the above, the research design of this case study followed these steps. The first step (identifying research questions) was tackled in Section 3.3.4. The remaining steps are discussed next.

3.4.2 Case selection

In this research, a single case is investigated with the intention of performing a descriptive study. Ryan and Filene (2012:1) define a single case as "*an evaluation method that can be used to rigorously test the success of an intervention or treatment on a particular case (i.e., a person, school, and community) and to also provide evidence about the general effectiveness of an intervention using a relatively small sample size".* This step involves the choice of IDE tools to evaluate, sample and select the subject participants.

3.4.3 Research population

Salkind (2012) defines population as the total of all the individuals who have certain characteristics that are of interest to a researcher. For the purpose of this research, the researcher saw the participants as consisting of three groups: the NetBeans IDE application users, JCreator IDE application users, and the usability experts for both IDE applications.

The target population in this case was typical students who are learning to program in Java at a selected University. However, for the purpose of this study it was appropriate to target learners doing Java programming at second year Computer Science (CS2) as the accessible population. The following criteria were used to select them:

Table 3-1: Criteria for user selection in the accessible population

- They had to have been already exposed to the Java syntax at CS1 level;
- They had to have usability knowledge of either NetBeans or JCreator IDE;
- Expert reviewers must have been teaching Java programming; and
- Expert reviewers must have used several IDEs during their experience as Java programming lecturers.

Table 3-2 shows the target population of selected students.

Table 3-2: Age category and Gender of students (target population)

Age Category	JCreator		NetBeans	
	Male	Female	Male	Female
18 – 21	5	7	1	0
22 - 25	9	10	12	3
26 - 30	2	1	3	2
Total	16	18	16	5

The selected students performed a crucial part of this study to determine whether IDE quality of use affects OOP concepts comprehension. The responses were used to determine the guidelines for an interactive environment for teaching and learning OOP concepts. On the other hand, experts were seen as people with experience in lecturing Java programming language with exposure to different IDEs. The experts were chosen to evaluate and uncover their perception of usability on IDEs, to comment on the design issues of the selected IDE applications, and also give feedback on students' behaviour with regard to motivation while learning to program. The expert reviews assisted in determining the relationship between learning OOP concepts, students' perception of usability in using IDEs and the students' behaviour while learning to program.

3.4.4 Unit of analysis

The analysis was conducted on two groups of students doing Java programming at CS2 level on two different campuses at the selected University. Students at this level were already exposed to OOP concepts and the two IDEs: JCreator LE 5.0 and NetBeans, these two IDEs were used in the previous years with each campus only using either and not both at the same time. 55 students indicated in Table 3-2 with consideration of demographics were used as total population. One group had a total of 34 students on one campus and 21 students on the other campus. Three Java programming lecturers as expert reviewers were used for heuristic evaluations. Most of the selected students before university enrolment originate from rural schools with limited or no educational resources and are often understaffed with teachers and therefore are technologically disadvantaged.

3.4.5 Sampling technique (Purposive)

Field (2005:120) defines sampling as “a smaller (but hopefully representative) collection of units from a population used to determine truths about that population”. The main objective of sampling was to make sure that the target population is represented in the process (Mouton, 1996). The research adopted the purposive sampling technique to enable a deliberate hand-picked potential user-participants that are likely to produce valuable data to meet the purpose of the research (Oates, 2005).The sample was chosen on who the researcher thought would be appropriate for the study. This was used primarily because of the limited number of people that have expertise in the area being researched.

3.5 Data collection methods

There are different methods that a researcher can utilise to gather data from the selected population. The following are some of them:

- **Observation:** This is an organised process of observing, recording, describing, analysing and deriving meaning out of an individual’s or from a group of people’s behaviour (Saunders, Lewis &Thornhill,2009).
- **Interview:** This is a method in which an interviewer poses questions to a respondent, and the respondent provides answers to the questions. They can be structured or unstructured. Structured interviews comprise a set of questions asked in a standardised order and the interviewer does not deviate from the interview schedule; these are based on close-ended questions. While unstructured interviews questions are sometimes referred to as ‘Discovery Interviews’ and are more like a ‘Guided Conservation’ than a strict structured interview (McLeod, 2014).
- **Questionnaire:** This is a method in which the respondents are asked to answer the same questions in a certain order. They are appropriate to collect data in a large population and can take the form of an interview-administered questionnaire, an online questionnaire, a postal questionnaire, or a self-administered questionnaire (Saunders, Lewis & Thornhill, 2009).

Given the above, questionnaire and unstructured interview schedule was used in this study. The interview schedule contains a set of prepared questions designed to be asked exactly as worded (McLeod, 2014). The standardised format of interview schedule assisted in asking each

interviewee some questions in the same order. Students were interviewed in groups and they recorded their response in the interview schedule. This was done so as to accommodate even the shy students.

A set of eleven questions were used on the questionnaire to test usability of the software, including efficiency and reliability that looks at response time of IDEs and ease of use. To attest the confidence in understanding OOP concepts students were given a further set of question on the questionnaire and interview schedule to gather their confidence in understanding OOP concepts. These questions were made up of Object-Oriented concepts, e.g. class, encapsulations and many other Object-Oriented concepts.

3.6 Data generation

A means whereby empirical field data or evidence can be produced is called data generation (Oates, 2005:36). Data can take two forms (Oates, 2006: 36): Quantitative or Qualitative. The former is numerical data; the latter is all other types of data other than numerical, such as words, images and sounds, etc. (Oates, 2006: 36). In this study, both qualitative data and quantitative were collected using the data-collection instruments in Table 3.3 and Table 3.4.

Table 3-3: **Quantitative Data-Generation Instruments**

Quantitative data generation instruments
• ***Satisfaction questionnaires; Likert-scale ratings;***   • ***Expert review ratings; and***   • ***Students' actual Test and Exam mark results***

Table 3-4: Qualitative Data-Generation Instruments

Qualitative data generation instruments
• ***Expert's comments;***   ***and***   • ***Expert answers to questionnaire open ended questions***

This study used more than one data-generation method for the following reasons, according to Oates (2005: 37):

(a) To look at a phenomenon of interest in different ways, resulting in the production of more data; these extra data could improve the quality of the research.
(b) To enable a comparison of the findings from one method with the data from another method.

The above is mostly referred to as method triangulation.
In this study, user (student) evaluations and expert reviews were chosen. It aimed at gathering three types of data: students' perception of usability in using IDEs, OOP concepts misconceptions while using IDEs, and user behaviour while programming.

3.7 Data analysis techniques

The collected data through questionnaire, interviews schedule, class tests and exam results was analysed using an Excel spreadsheet through the use of pivot tables and graphs. Excel was found to be easy to use and was sufficient for the task. The analysis of uncovering patterns and trends in data sets were done using the spreadsheet and subsequently interpreted, i.e., explaining those patterns and trends. The analysis and interpretation were done to the research questions with the objective of highlighting useful information, suggesting conclusions, and supporting decision-making (Walsham, 2006).

Most research questions in qualitative approach studies lead to different classes of data analysis namely within-cases, cross-cases and holistic-case analysis. This research, however, employed the within-case analysis. Creswell (2007) defines within-case analysis as analysing, interpreting and legitimising data that help to explain a phenomenon in a bounded context and make up a single case, department, organisation or community.

3.8 Ethical Considerations

At the time when the study proposal was submitted for consideration, The Cape Peninsula University of Technology Research Ethics Committee had requested for a completed research ethics form on how the research was to be carried out. Risks to the participants and any other person had to be clearly stated. The research proposal satisfied the stringent requirements set by the University.

In the study, procedures stated in the ethics form were followed. All participants were given a clear picture of what the study was about and what they were required to do. They were also told that they could withdraw at any time and that a joint decision would be made on what to do

with the data collected upon withdrawal from the study. Furthermore, confidentiality was ensured through the use of pseudonyms and the identity of the participants was not revealed without their permission.

3.9 Summary

This chapter narrated the research study questions with the associated objective. It raised the need to investigate factors that affect OOP comprehension through quality of use by performing user evaluations. User-satisfaction questionnaires, unstructured interview schedules and expert reviews were used to gather data in order to develop an understanding towards creating guidelines of an interactive environment for teaching and learning OOP concepts using Java programming language. Method triangulation was used to analyse the data using an Excel spreadsheet.

The next chapter focuses on the findings of the collected data and further give an analysis and interpretations of the patterns and trends in the data.

CHAPTER FOUR : ANALYSIS AND DISCUSSION

4.1 Introduction

The purpose of this chapter is to discuss the result of the semi-structured questionnaire and interview schedule responded to by a total of 55 participants. Prior to the commencement of the research study, the importance, basis and intention of the study were provided to the respondents. Moreover, the respondents were also given the assurance that all the data they gave was used solely for the purpose of the research and the identities of the respondents were kept confidential.

The findings presented in this section addressed the research question: Does the quality of use of an IDE enhance the comprehension of object-oriented programming concepts through teaching and learning of Java programming language. It focused on the most common mistakes students make during program development in a particular IDE; it attempted to answer whether the use of an IDE affects the understanding of OOP concepts and whether mistakes suggested usability problems in IDEs with any other misconceptions. The focus was mostly centred on students' perceptions on how they are currently fairing with the IDEs while learning OOP concepts using Java. The findings were from the questionnaires and focus groups done with students based at two different sites of the selected University's Department of Information Technology.

The chapter presents the general demographics of the students, such as their gender, age, timeframe programming and previously used programming languages. Focus was on the students' perceptions on the flexibility of using IDEs and whether it assisted them to understand OOP concepts. After presentation of the overall findings, an analysis and interpretation of the results is narrated.

4.2 Findings

This section presents findings for the two campuses of the selected University with one campus using JCreator and the other NetBeans as IDEs for learning OOP concepts using Java. The findings to the following research questions

as presented in Table 4.1 follows:

Table 4-1: Research questions

Does the quality of use of an IDE enhance the comprehension of object-oriented programming concepts through teaching and learning of Java programming language? - Does the use of an IDE affect students' understanding of OOP concepts? - Does the use of a particular IDE increase students' confidence in learning Object–Oriented concepts? - What are the most common mistakes and misconceptions students make during program development in a particular IDE?

The study findings are based on students' perceptions on how they are currently fairing with the JCreator and/or NetBeans IDE on their computers. The findings from the questionnaires and focus groups done with students are independently presented beginning with the campus which uses JCreator and then the campus which uses NetBeans. The section will first present the demographics of the friendliness of the current software on the student computers. The study goes on to look into detail the students' perceptions on the flexibility of using JCreator and whether it assists them to understand Object-Oriented (OOP) concepts. After presenting the findings, an analysis of the results will be presented.

4.3 FINDINGS FOR CAMPUS USING JCreator

4.3.1 Demographics

4.3.1.1 Gender

A total number of 34 students fully answered the questionnaires for this study on this campus with 18 (53%) females and 16 (47%) males as shown in Figure 4-1.

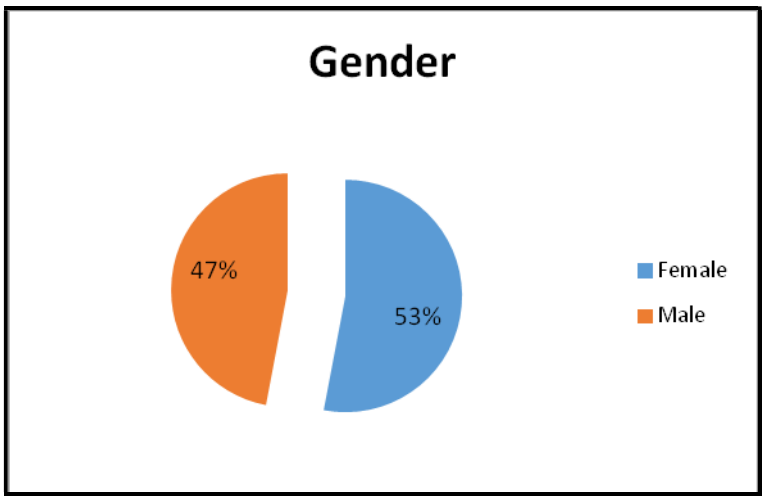

Figure 4-1: Gender

4.3.1.2 Age

The majority of the students were from the age category of 22 – 25 years with 30% females and 26% males.

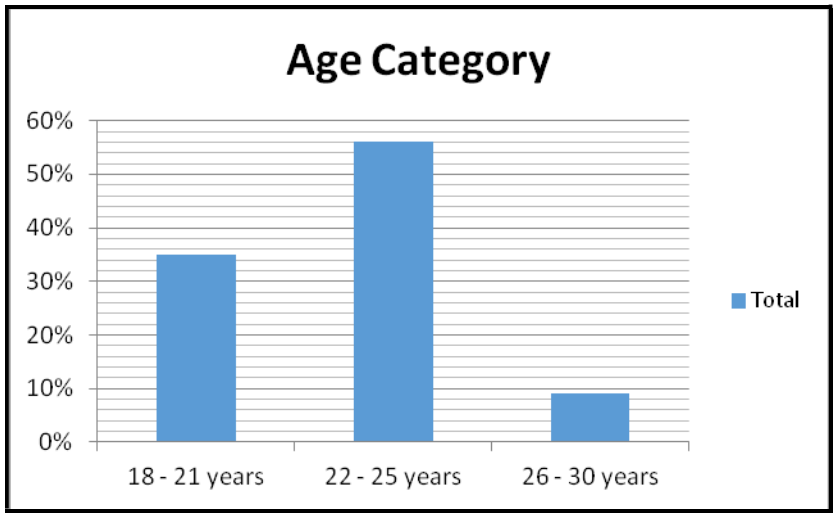

Figure 4-2: Age category

4.3.1.3 Timeframe of programming

The findings show that 91% of the students have more than a year programming and only 9% have less than a year programming. Figure 4-3 shows that of the 91% of the students with more than one year programming, 50% are females and 41% are males.

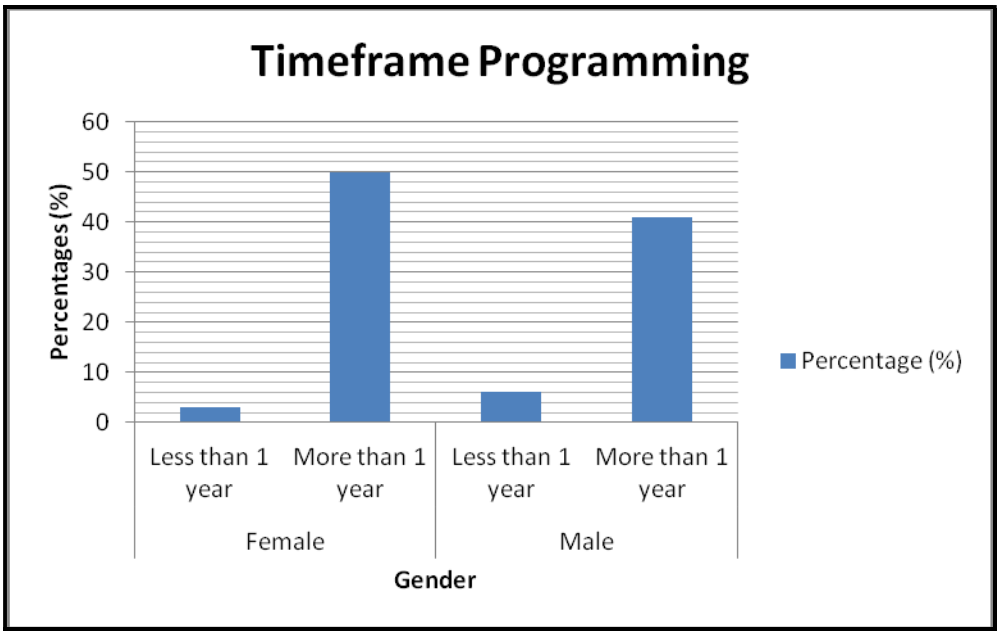

Figure 4-3: Timeframe of programming

Findings indicate that regardless of the number of years spent on programming, all 34 students have used Vb.Net and Java programming languages. Students who have more than a year of programming have in addition used other programming languages such as ASP.net; HTML; C#.Net and php.

Table 4-2: Programming languages used

Timeframe programming	Programming languages used before	Total %
Less than 1 year	Vb.Net and Java	9
More than 1 year	Java	3
	Java, Vb.Net, ASP.net and HTML	3
	php, html , javascript and Java	3
	Vb.Net	15
	Vb.Net and Java	56
	Vb.Net, C#.Net	3
	Vb.Net, html, php and Java	9
Grand Total		**100**

4.3.2 Understanding object-oriented concepts using JCreator

The findings reflected that majority of the students found it easy to understand object-oriented concepts using JCreator with the highest recorded on Class concept (76, 5%), followed by Object (67, 6%) , 65% find Method Overriding easy, and 59% on Inheritance. Encapsulation and Polymorphism recorded the lowest each with only 35% of the students stating that it is easy to understand, with 32% stating that they feel it is moderate to understand.

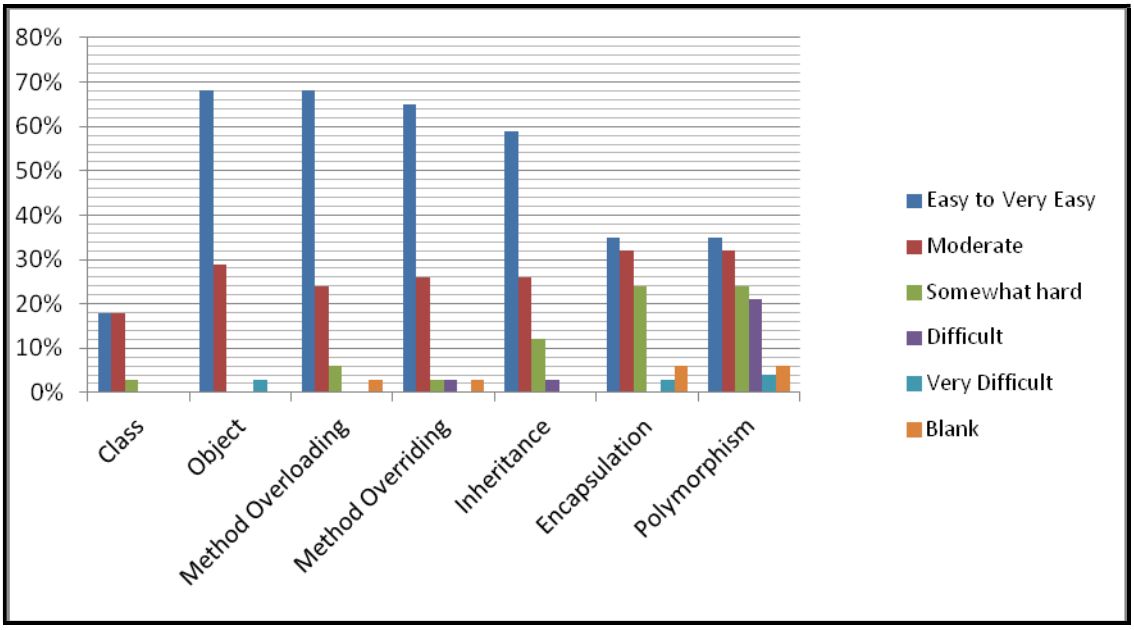

Figure 4-4: Understanding object-oriented concepts using JCreator

The findings from the focus group discussion also confirmed that 92% of the students find it easy to learn OOP. The remaining 8% stated that although learning OOP is challenging, it becomes interesting as one learns. They further stated that learning OOP requires clear presentations and examples to make it easy to understand and grasp. Other students stated that in order to understand OOP, it is essential to have more practice and pay more attention to details. They further reiterated that there are many concepts that one needs to understand before one can fully master OOP.

4.3.3 Easiness of JCreator Software

The study was also concerned about students' perceptions if they find JCreator easy to use. The findings indicate that 56% of the students find it easy to use the JCreator software; 47% are able to learn how to use everything offered with JCreator. 50% agreed that navigating through menus

and toolbars is easy to do. Moreover, 47% agree that it is easy to find options they need in the menus and toolbars.

The findings show that students find five sections of the JCreator software as neutral to use namely that the software is engaging (65%), contents of the menus and toolbars matching their needs (41%). 41% were neutral on whether getting started with the JCreator version is easy. 44% stated that they are neutral when it comes to discovering new features and 50% are neutral on the satisfaction they get from using JCreator.

The most prominent feature that most students disagree to be easy to use was discovering new features (32%) followed by finding the contents of the menus and toolbars matching student's needs (21%). Only 21% disagree with the following notions that getting started with the software version is easy, finding options and the software responding timeframe is easy.

Table 4-3: Student perceptions on JCreator LE 5.0 IDE

	Question	Disagree (%)	Neutral (%)	Agree to Strongly Agree (%)	Blank (%)
1	Software easy to use	**6**	**38**	**56**	**0**
2	I am in control of contents of menus & toolbars	**12**	**44**	**44**	**0**
3	I will be able to learn all offered in software	**9**	**41**	**47**	**3**
4	Navigating through menus & toolbars is easy	**9**	**41**	**50**	**0**
5	Software is engaging	**9**	**65**	**24**	**3**
6	Contents of menus & toolbars match my needs	**24**	**41**	**29**	**6**
7	Getting started with the software version is easy	**21**	**41**	**38**	**0**
8	Finding options in menus & toolbars is easy	**21**	**32**	**47**	**0**
9	Software responds in time	**21**	**38**	**41**	**0**
10	Discovering new features is easy	**32**	**44**	**24**	**0**
11	Software is satisfying to use	**18**	**50**	**32**	**0**

4.3.4 Confidence gained using JCreator on the following object-oriented concepts

As depicted in Figure 4-5, it was largely affirmed by 53% of the students that they gain more confidence on Class object oriented concept by using JCreator. At par with 38% of the students, they acknowledged that they gain confidence in Method Overloading, Method Overriding and Inheritance, and just 18% stated Encapsulation.

Object had 14 (41%) students stating that they gain confidence quite a bit, followed by Inheritance, Encapsulation and Polymorphism all with 12 (35%) students citing that they gain quite a bit of confidence.

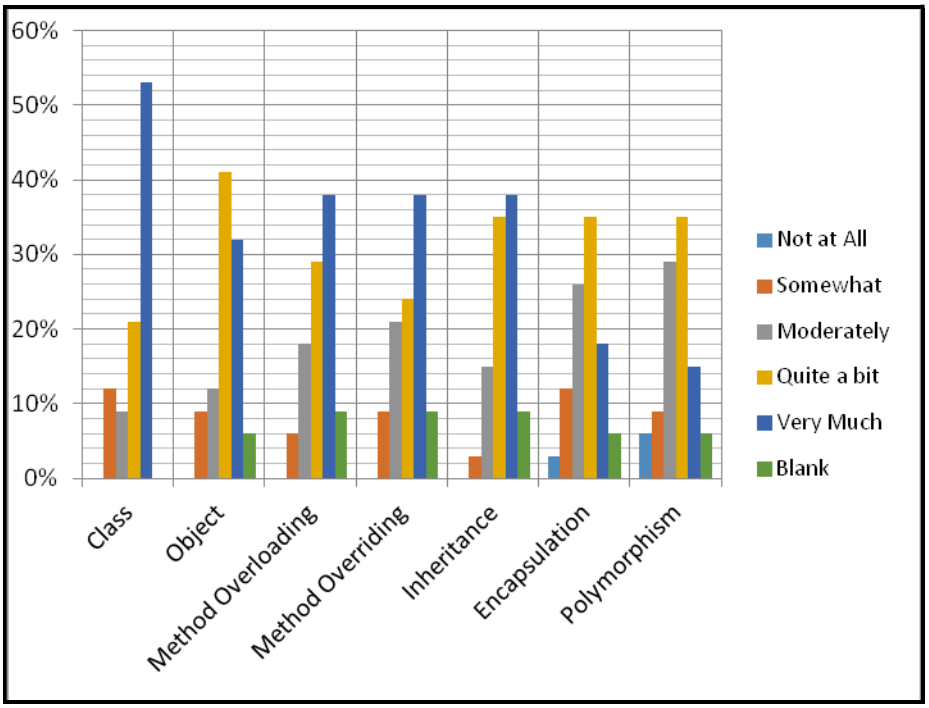

Figure 4-5: Confidence gained from using JCreator

When it comes to being able to relate between a Class and an Object in JCreator, 90% of the students affirmed that they can easily relate the two. The remaining 10% did not state whether they can relate between a Class and an Object in JCreator.

4.3.5 Recovering from errors and common mistakes using JCreator

44% of the students acknowledged that they find it easy to very easy to recover from system (IDE) error messages in JCreator as shown in Figure 4-6. The majority find it somewhat hard to very difficult to recover from system (IDE) errors.

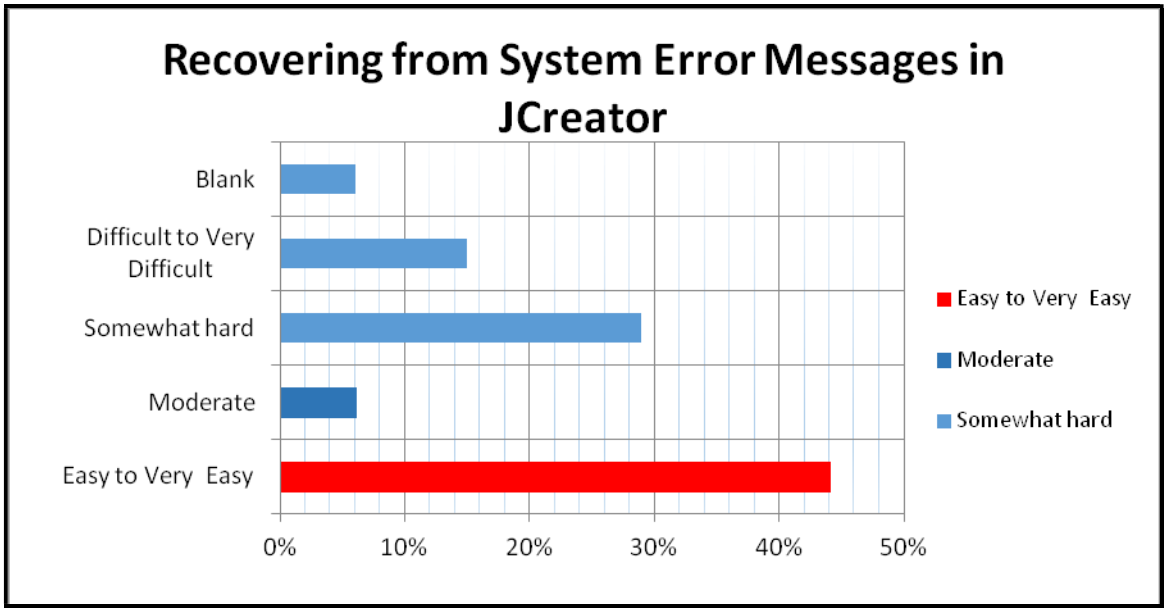

Figure 4-6: Recovering from error Messages

4.3.6 Recovering when output (animation) is not the movement expected

A few students find it easy to very easy to recover when output (animation) is not the expected movement (32%), and 29% cited that they find it moderate/fair to recover from unexpected movements. The other 29% emphasised that it is somewhat hard to difficult for them to recover once the output is not the expected movement as shown in Figure 4-7.

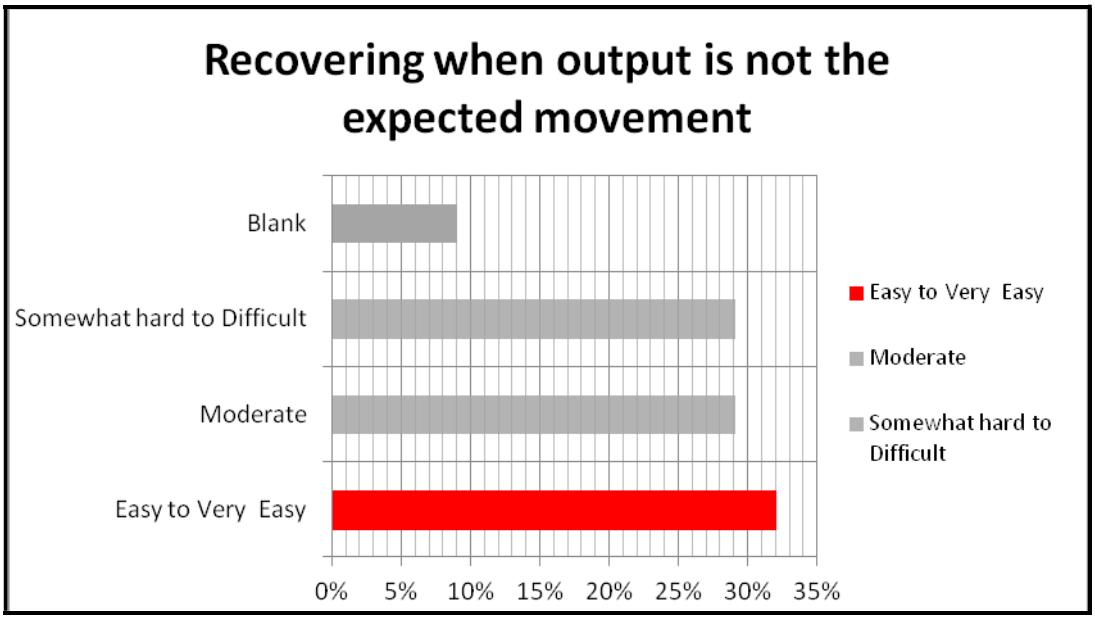

Figure 4-7: Recovering when output is not the movement expected

Findings from the focus group discussions had students admitting that they usually make spelling mistakes. For instance, they misspell Class names, variables and many others since the IDE does not correct spellings. They also confuse the lower and upper case. When compiling, students make the mistake of running the program and they find it difficult to stop it. One of the critical errors stated was that the student fails to create the correct object. They only try to create it after they realise that it is needed. The other error is making all Class variables private. It was also stated that students find it difficult to debug. On the other hand, some students admitted that they simply forget certain things and they feel that they need more motivation and to remain focused so as to avoid these mistakes.

4.3.7 Using JCreator to complete task

Figure 4-8 show that 50% of the students find it easy to complete tasks using JCreator software. 38% reported that it is moderate and the remaining 11% stated that it is difficult; 3% failed to give a response.

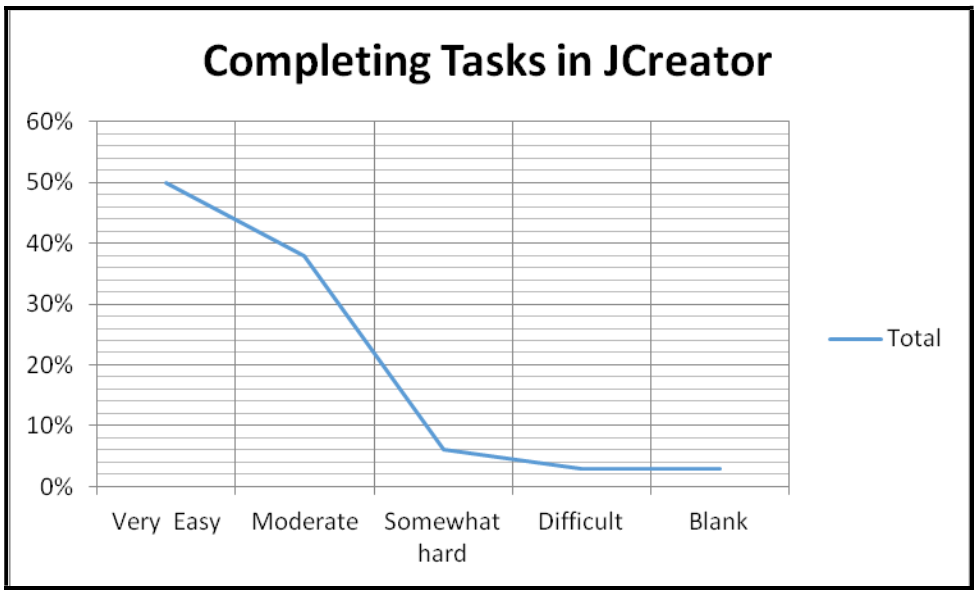

Figure 4-8: Completing Tasks in JCreator

4.3.8 Difficult Tasks to accomplish in JCreator

Figure 4-9 shows that fixing and handling errors in JCreator has been cited by 29% of the students as the most difficult tasks to accomplish. The students (15%) also mentioned that they find it difficult to save multiple files and classes. 9% reported that solving exceptions is difficult and also Polymorphism was cited as some of the difficult tasks.

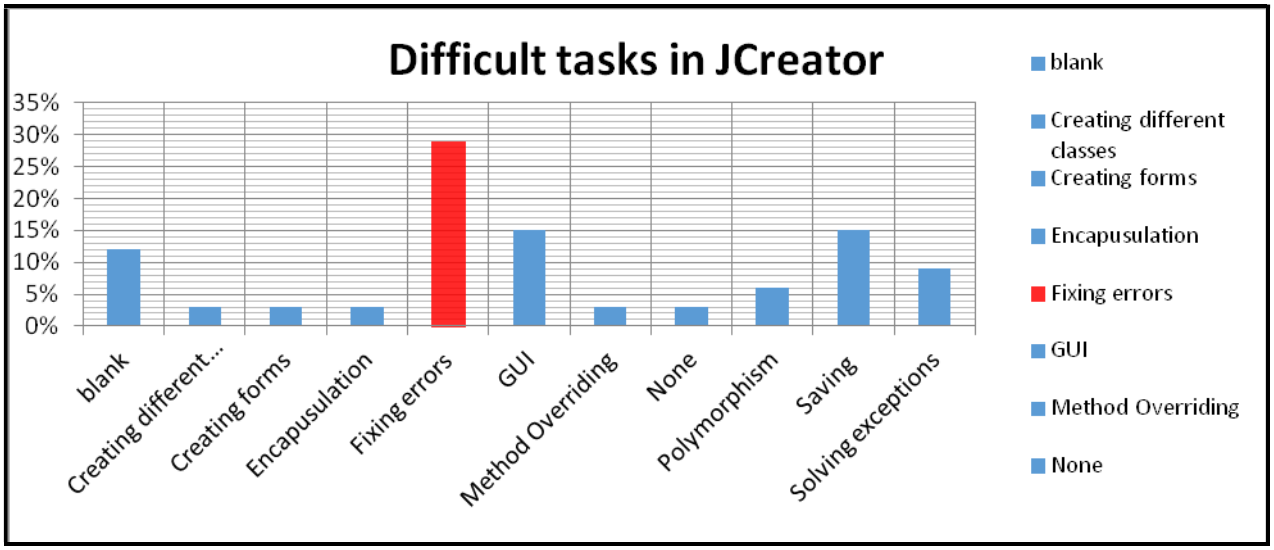

Figure 4-9: Difficult tasks in JCreator

4.3.9 Misconceptions

A few students stated that they have not seen the application of JCreator IDE being applied practically in the real world of programming; some opined that they consider the software as outdated and not used in the industry. Another student felt that JCreator is only good to orient students to the programming field, but not for practical use.

32% of the students find Coding with JCreator to be exciting and gives them the feeling of wanting to learn more. JCreator is easy and friendly to use for most students. They further assert that the IDE is not too complex to understand and they would highly recommend the software. They view JCreator as user friendly. When compared to NetBeans, JCreator is said to have more positives than negatives. In addition, they mentioned that JCreator provides a good platform for being introduced to programming. Amongst those who prefer JCreator, they also acknowledge that JCreator improves error handling because errors can be easily detected. Others admit that JCreator has helped them to comprehend Java and has greatly assisted them to improve their programming skills.

Some students (18%) who do not consider JCreator as a good IDE stated that the interfaces are so difficult to understand, even keywords like ‘super’. They further asserted that JCreator can be hard at times, especially when it comes to detecting and fixing errors and most of the time it leaves errors that are not easy to modify. On the other side, even though they view JCreator as

not very user friendly, they do acknowledge that JCreator can be easy to use if one understands it well.

One student cited that although using JCreator is very easy, sometimes it can be difficult as there are tasks that need much attention and concentration such as testing classes.

Generally, the findings show that most students find it easy to learn using IDE. They acknowledge that although it can be challenging, it is quite interesting. IDE requires much attention and one needs to be willing to explore the tools, controls and features. Students stated that it would be much easier to learn using IDE if it could locate and fix errors.

4.4 Data Analysis and Interpretation: Campus using JCreator

4.4.1 Difficulties of learning Programming

4.4.1.1 Demographics

Gender

The findings as shown in Figure 4-10 show that 66.7% of females find JCreator software easy to use as compared to the males with only 44%. 50% of the males find JCreator neutral to use. Neutral may imply that they do not understand fully how the software works or they understand how the software works to a lesser extent and they do not have confidence to fully rate how easy it is to use the software. This implies that the students have an incomplete understanding of programming concepts as stipulated by Sorva (2013). The results are also consistent with the study done by Chege *et al*., (2013), who also confirms that with regard to outcomes of schooling such as academic performance, school attendance or even completion, females outperformed males in over half of the school, as an example in rural Kirinyaga, Kenya.

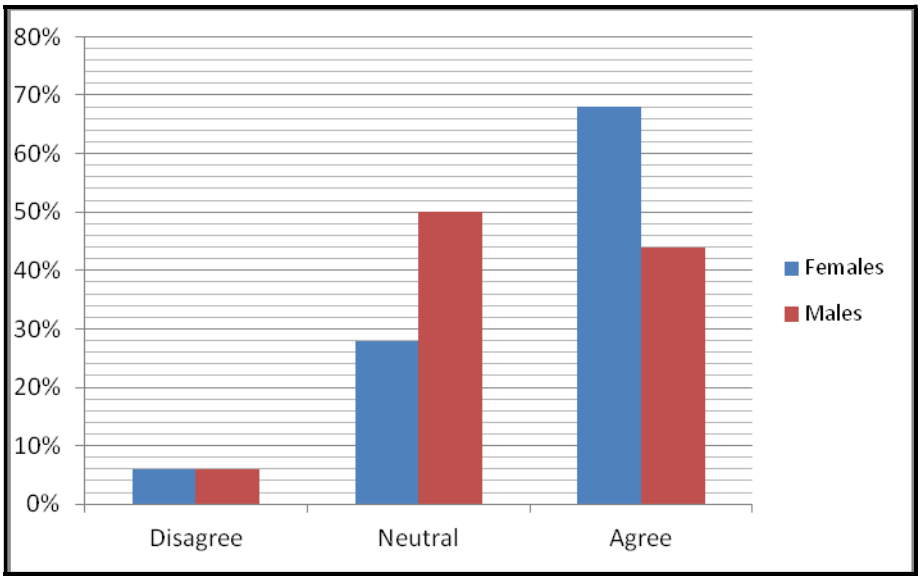

Figure 4-10: Gender analysis

Age

Amongst the 67% of females who find JCreator easy to use, 58% fall in the 18 – 21 age group category, followed by the 33% who fall on the 22 -25 age category. Amongst the 44% of males who find JCreator easy to use, the majority fall between the 22 – 25 age group (43%), followed by the 29% falling between 18 – 21 and 26 – 30 age categories respectively. The results indicate that females between 18 – 21 easily understand and grasp concepts and as they grow older the level of understanding drops. Whereas with males, they perform better when they are between the 22 – 25 age group category as reflected in Figure 4-11. Thus, it may be advisable to enrol females at a younger age and males when they are much older and mature between the ages of 22 – 25.

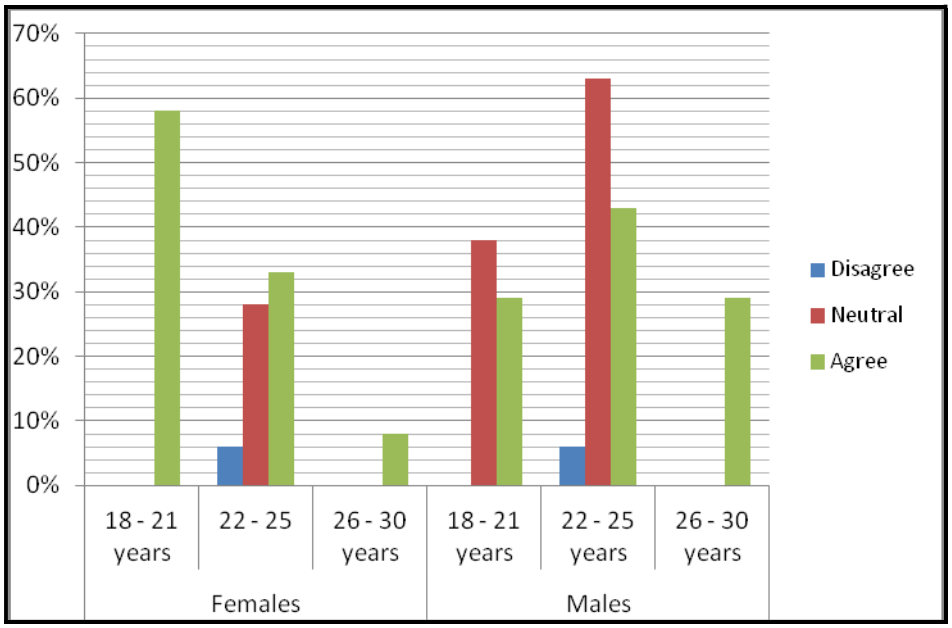

Figure 4-11: Software easy to use per age and gender

4.4.1.2 Programming timeframe and previous programming experience

Results as shown in Table 4.4 indicate that 61% of females with more than a year of programming experience find it easy to use JCreator. This can be attributed to the experience acquired during the timeframe of programming and the fact that they had less new concepts to familiarise with. Unlike with the males, 50% who have been programming still stated that they find the software neutral to use. This can be due to the issue of age as the results indicate that males who are much older easily grasp and understand concepts well. The results also reflect that when students are not able to gain from their prior knowledge, it is either because they struggle to change their mindsets to new paradigm, or after obtaining the OOP mindset are unable to apply their previous knowledge in the new environment (Goosen & Pieterse, 2005). So regardless of the fact that these males have had a year of programming experience, the issue of age can be pointed out as a contributing factor. Studies conducted by Byrne and Lyons (2001) and Allert (2004) indicated that students who had previous programming experience tended to perform better than other students.

Table 4-4: Software easy to use in relation to timeframe programming

Gender	Software easy to use	Timeframe programming	%
Female	Disagree	More than 1 year	5.56%
	Neutral	More than 1 year	28%
	Agree	Less than 1 year	6%
		More than 1 year	61%
Female Total			**100.00%**
Male	Disagree	More than 1 year	6%
	Neutral	More than 1 year	50%
	Agree	Less than 1 year	13%
		More than 1 year	31%
Male Total			**100%**

4.4.2 Object-Oriented concepts:

4.4.2.1 Understanding OOP concepts using JCreator

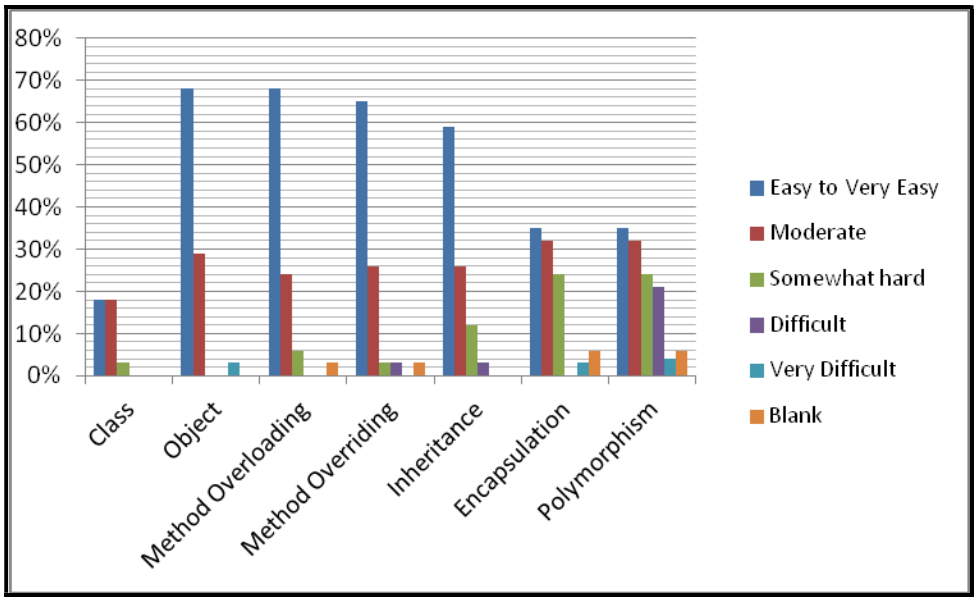

Figure 4-12: Understanding OOP concepts with JCreator

Sicilia (2006) argued that programming instructors should carefully help their students in comprehending the OOP concepts and in translating the conceptual models into Java programs. The study found that most students understand most of the OOP concepts, especially Class,

Object, Method Overloading, Method Overriding and Inheritance. This reflects that most of the students have a strong base as Object refers to the main component of the OO paradigm that is used for carrying out specific tasks. A good understanding of Object tends to give the student a good potential to carry out most tasks.
The study realises that only 35% of the students respectively understand Encapsulation and Polymorphism.

According to Ghanim and Al-khafaji (2014), Polymorphism refers to the provision of multiple forms and methods. When used in the OOP context, this term implies that different objects may respond individually to the same message. Therefore, Polymorphism may be used to indicate different implementations. Polymorphism also supports greater abstraction wherein a single message can evoke different behaviour. This implies that students still find it difficult to fully understand this OOP concept. If its application is more or less done in real world scenarios, it may actually enhance students understanding.

4.4.2.2 Confidence gained

JCreator has boosted 60% of the students' confidence to understand the Class concept. However, when it comes to other Object-Oriented concepts, such as Object, Method Overloading, Method Overriding and Inheritance, merely 39% of the students affirmed to have gained confidence to understand these concepts. The fact that JCreator does not fully build confidence for students to understand most of the OOP concepts could be the reason why most students find it difficult to fix errors and to recover from unexpected movements. Most scholars stipulate that understanding the programming language is an essential part of computer science and the key to success. The results indicate that the use of IDEs does not guarantee that students will be able to fully understand OOP concepts.

4.4.2.3 Common mistakes and difficult tasks to accomplish in JCreator

Only 44% of the students acknowledged that they find it easy to very easy to recover from system (IDE) error messages in JCreator. The results are quite consistent as only 29% of the students reported that they find it difficult to fix and handle errors in JCreator. In addition, 15% confirmed that they struggle to save multiple files and classes and solving exceptions as well as Polymorphism. Furthermore, merely 32% find it easy to recover from unexpected movements.

This can be due to the fact that students do not fully understand the JCreator software itself and they do not take much time to broadly understand it. Kinnunen and Malmi (2006) assert that

because students do not dedicate a lot of time to programming, they end up lacking motivation and dedication to fully learn and understand the program. This further confirms the claim by *Kasurinen et al.* (2008) that students often lose interest in programming because complex models and structures have to be learned before anything visually impressive can be created. This leads to some students to believing that JCreator programme is more theoretical and difficult to be practically implemented in the real world.

4.4.2.4 Satisfaction in using JCreator LE

Figure 4-13 shows that 64% of the students find the JCreator LE software satisfying to use with 39% females and 25% males. When students lack a full understanding of a software, they lose interest thus resulting in low satisfaction levels. Ironically, 50% of the students reported that they find it easy to complete tasks in JCreator, yet they also affirmed that they do not find much satisfaction in using the software and they do not have much confidence in understanding most of the Object-Oriented concepts when using JCreator.

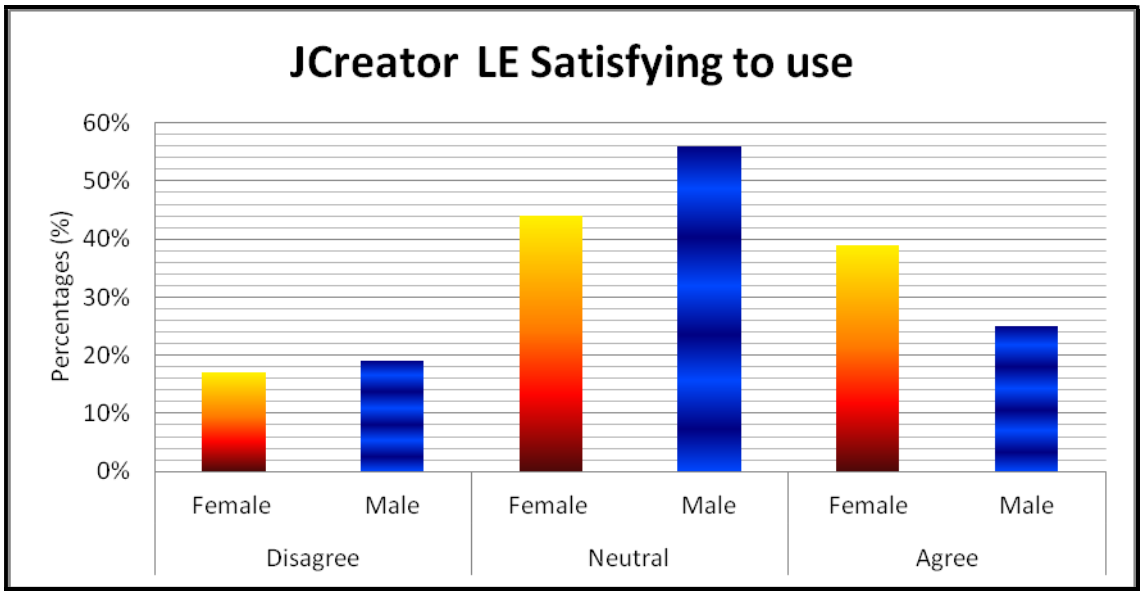

Figure 4-13: JCreator LE satisfying to use

Overall, as depicted in Figure 4-14 below, JCreator has great potential to capacitate students to comprehend programming and moreover understand OOP concepts as almost half of the students (47%) acknowledge that if properly mentored they will be able to use all that the software has to offer. This implies that students need to be mentored by experts who are highly skilled in programming as affirmed by Vihavainen *et al*.(2011). It is also important to bear in mind

that most of these students come from rural disadvantaged backgrounds where they had limited opportunities to clearly understand programming and the value it offers to the real-world environment.

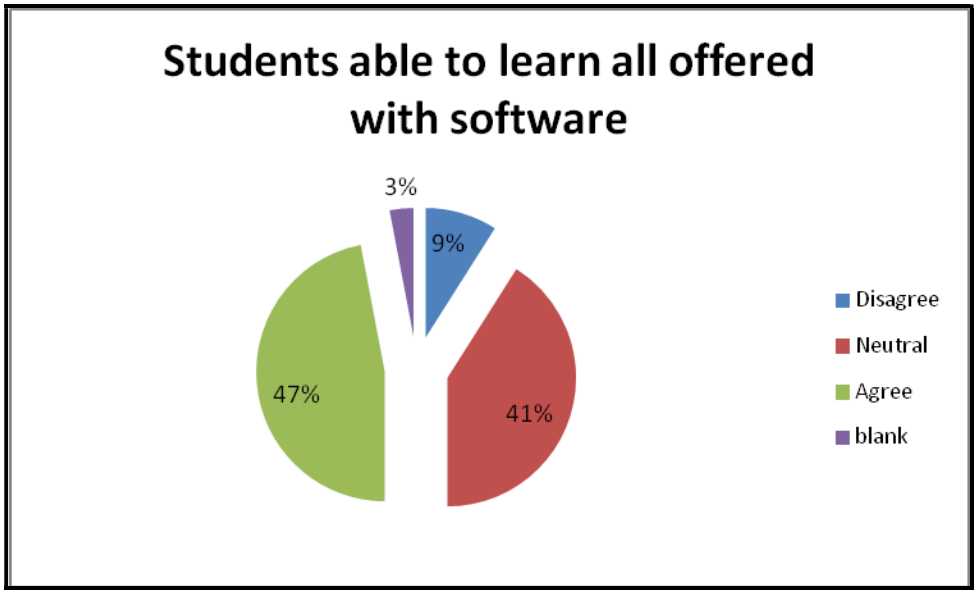

Figure 4-14: Students will be able to learn all offered with software

More efforts need to be effected to further motivate students to unlock their potential and sharpen their skills of comprehension and memorization abilities which are crucial to programming. For instance, the findings indicate that the software menus and toolbars need to be adjusted to meet various needs since only 29% of the students find the menus and toolbars matching their needs as shown in Figure 4-15.

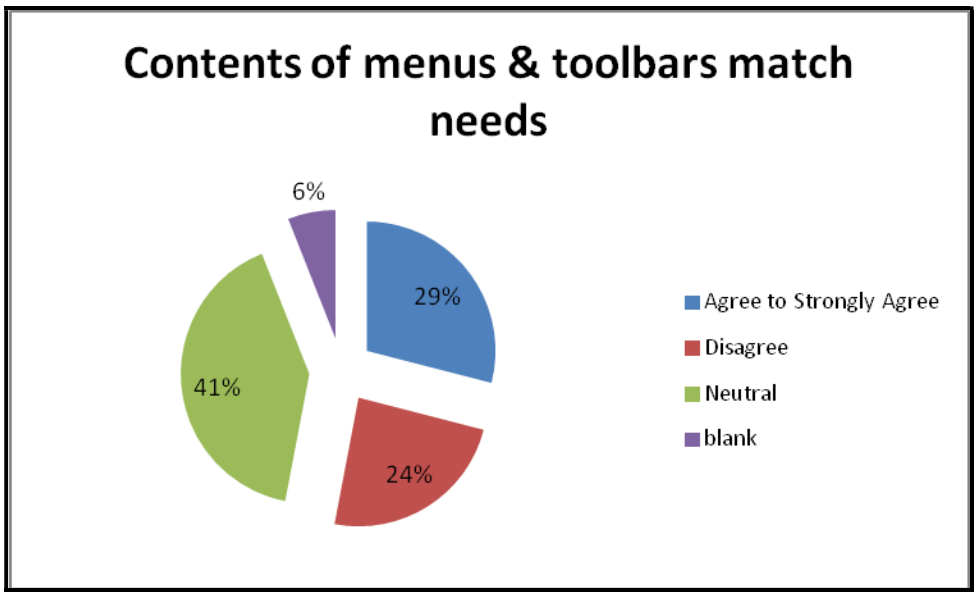

Figure 4-15: Contents of menus and toolbars match student needs

This means that the majority of students do not find much relevance as the menus and toolbars do not quite match their needs. This results in students having low interest in using the software. It is crucial to find mechanisms to instil interest for students to enjoy using the software. Storey (2005) suggests using adaptive interfaces which may perhaps be tailored to suit different kind of users and tasks. Software tools normally have many features which may be overwhelming not only to novice users but also to expert users.

4.4.3 OOP Misconceptions

Some students stated that they have not seen the application of JCreator IDE being applied practically in the real world of programming, while others cited that they consider the software as outdated and not used in the industry. Other students felt that JCreator is only good to orient students to the programming field, but not for practical use. These assertions or misconceptions are due to the fact that students do not properly understand programming. A strong orientation into the field of programming might be required. If the students remain with these beliefs it becomes very difficult, if not impossible, for them to develop much interest to learn programming let alone understand OOP concepts.

4.4.4 Motivation to learn OOP

The research findings show that students are motivated by various factors to learn OOP. Among the reasons stated, the most prominent ones were the mere pleasure, excitement and

satisfaction that they get when programming. They also cited that programming is in demand in this information age and it will be easy for them to accomplish more career wise. Furthermore, some students stated that it is easy to understand OOP concepts using Java; this motivates them to further pursue on learning OOP. It would appear that poor understanding of the basics of programming tends to de-motivate students, thus strategies need to be in place on how to boost student motivation. McDowell *et al*. (2003) confirmed that students who used pair programming produced better programs, were more confident in their solutions, and enjoyed completing the assignments more than students who programmed alone. Thus, the idea of pairing students when programming may enhance the quality of their programs and encourage them to pursue programming further. Paired programming has a high impact to reduce mistakes made by students such as misspelling Class names and variables, confusing the lower and upper case, and simply forgetting certain things.

4.5 Findings for campus using NetBeans

Below are the findings to the following research questions as presented in Table 4-5.

Table 4-5: Research Questions

Does the quality of use of an IDE enhance the comprehension of object-oriented programming concepts through teaching and learning of Java programming language? - Does the use of an IDE affect students' understanding of OOP concepts? - Does the use of a particular IDE increase students' confidence in learning Object – Oriented concepts? - What are the most common mistakes and misconceptions students make during program development in a particular IDE?

This section presents the findings from the campus using NetBeans on how students are faring and perceive learning Object-Oriented concepts using NetBeans.

4.5.1 Demographics

4.5.1.1 Gender

A total number of 21 students out of 29 fully answered the questionnaires for this study from the campus using NetBeans with females and males as shown in Figure 4-16.

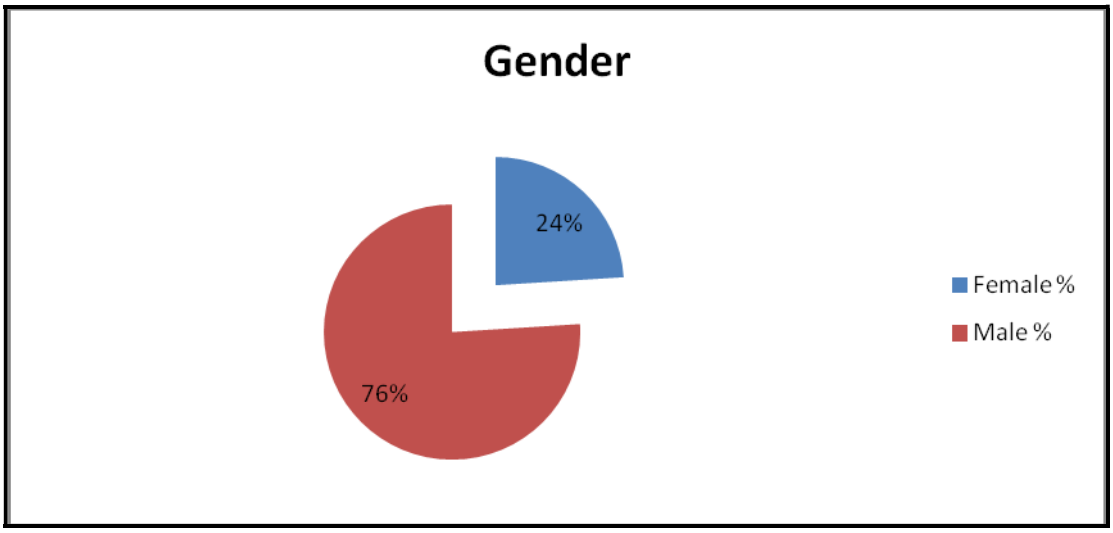

Figure 4-16: Gender

4.5.1.2 Age

Majority of the students were from the age category of 22 – 25 years as shown in Figure 4-17. The findings show that within the 22 – 25 years age category, 80% were males and 20% females.

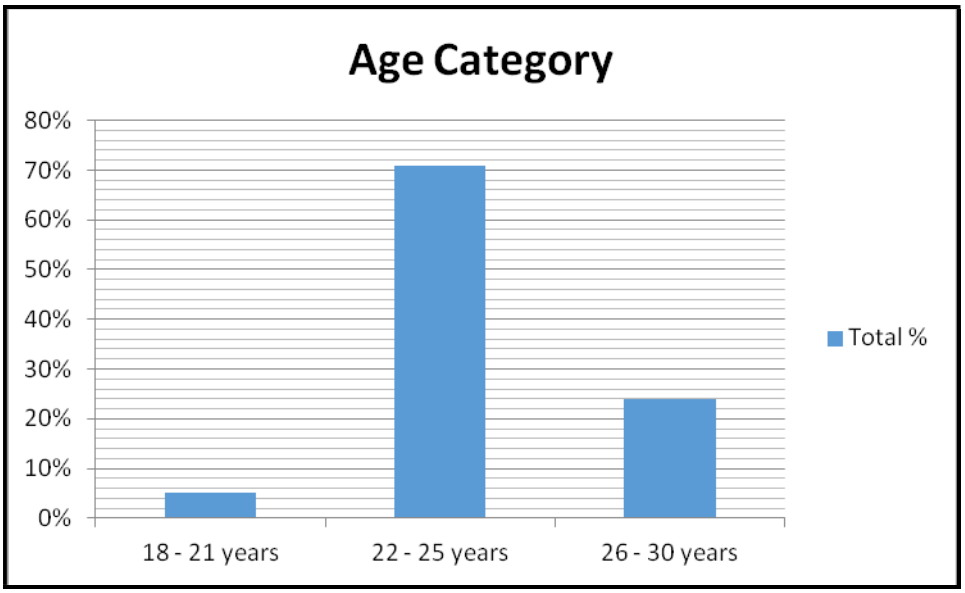

Figure 4-17: Age category

4.5.1.3 Timeframe of Programming

The research findings show that all the females have more than one year of programming experience, 88% of the males have more than one year programming experience, and only 13% have less than a year of programming experience as indicated in Figure 4-18.

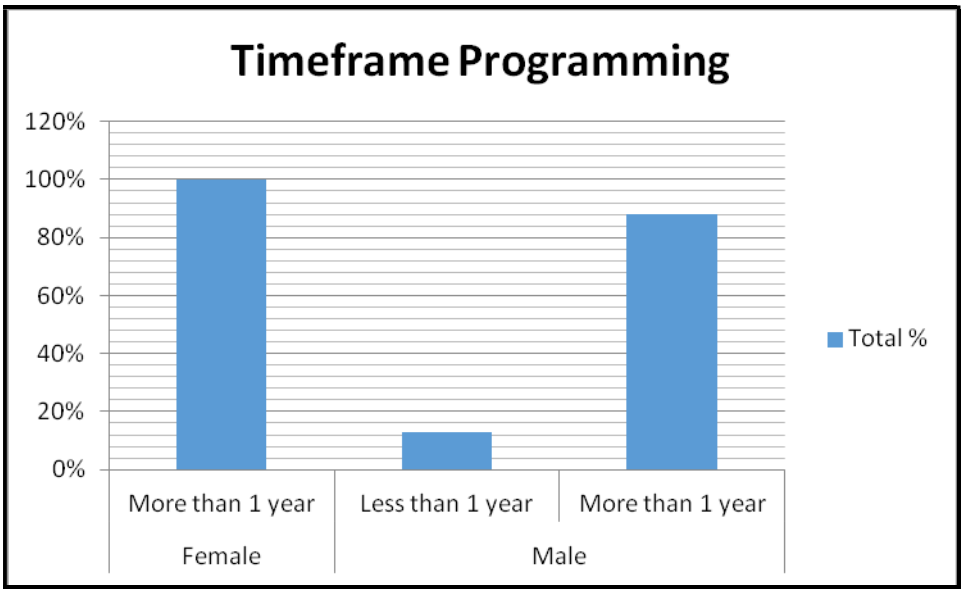

Figure 4-18: Timeframe of programming

Findings indicate that regardless of the number of years spent on programming, all 21 students have used Java software before. Students who have more than a year of programming experience have in addition used other programming languages such as Vb.Net, HTML; PHP and C++ as shown in Table 4-5.

Table 4-6: Programming languages used

Timeframe programming	Programming languages used before	Total
Less than 1 year	Dephi, Java and Vb.net	1
	Java and VB.net	1
More than 1 year	Java	5
	Java and C++	1
	Java and VB.net	7
	Java programming	1
	Java,c/c++,PHP and HTMl	1
	Python	1
	VB and Java	2
	VB.net	1
Grand Total		**21**

4.5.2 Understanding object-oriented programming concepts using NetBeans

The study aimed to understand whether students find it easy to understand Object-Oriented concepts using NetBeans software. The findings confirmed that students understand some of the concepts as 62% of the students stated that they find it easy to very easy to understand Class. 57% affirmed they find it easy to very easy to grasp Object, and only 43% respectively stated Method Overloading and Method Overriding. Merely 33% agreed to find Inheritance and also Encapsulation easy to understand. Polymorphism only had 24% of the students understanding the concept.

The results show that when it comes to understanding other oriented concepts, some students stated that they find it moderate to grasp OOP concepts using NetBeans. For instance, over 52% find it moderate to grasp Inheritance, and then 43% respectively stated Method Overloading, Encapsulation and Polymorphism. Method Overriding had 38% and Class had 33% of the students admitting that they find it moderate to understand these Object-Oriented concepts using NetBeans.

Few of the students reported that they find it difficult to very difficult to understand some of the Object-Oriented concepts. Only 10% of the students admitted that Inheritance, Encapsulation and Polymorphism Object-Oriented concepts are difficult to understand. Refer to Figure 4-19 for the summarised results on how students are faring in understanding OOP using NetBeans.

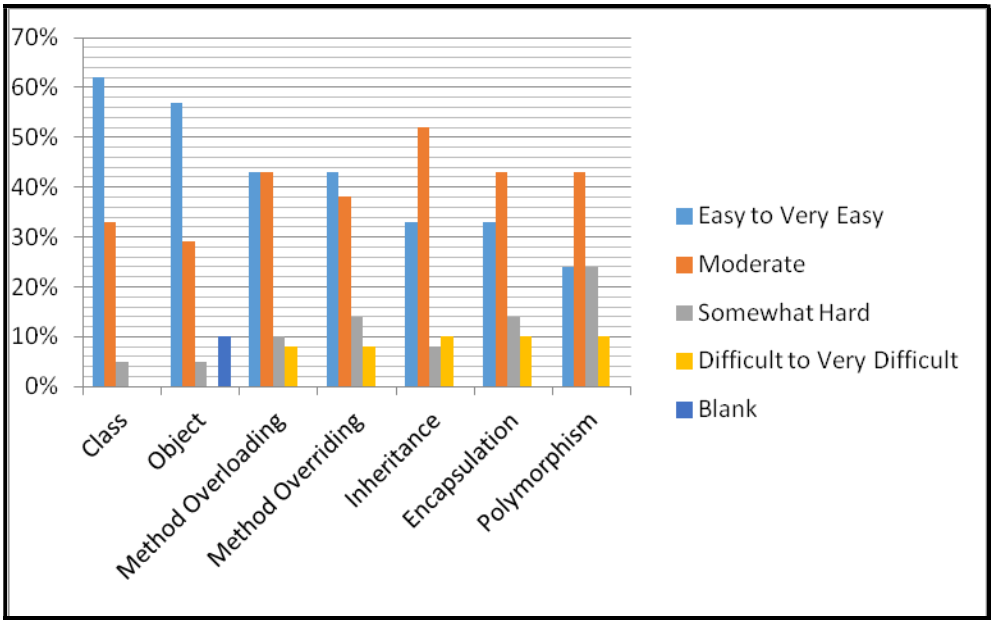

Figure 4-19: Understanding OOP concepts using NetBeans

The findings from the focus group discussions show that three quarters of the students (86%) find it easy to learn OOP. The students stated that although learning OOP concepts are not so difficult, much time practising and participating to understand the concepts is required. They further acknowledged that they need a lot of guidance to comprehend OOP concepts. Object-Oriented concepts were deemed to be challenging but exciting to learn. Only 7% reported that they find learning OOP neutral. In their exact words, they report that it is "not difficult and at the same time it is not easy as well". Only 7% admit that it is difficult to learn OOP concepts.

4.5.3 Easiness of NetBeans Software

The study enquired on whether the students found NetBeans software easy or difficult to understand. The results as presented on Table 4-7 shows that 67% of the students affirmed that the NetBeans software is easy to understand. 52% stated that the NetBeans software responds in time. When it comes to satisfaction levels, 48% of the students are satisfied with using NetBeans and only 43% find the software engaging.

Majority of the students (52%) confirmed that they feel neutral regarding if they will be able to learn everything that the software offers and finding options in menus and toolbars. When it comes to navigating through menus and toolbars, 48% of the students find NetBeans software neutral to use. In addition, 43% of the students reported that they find it neutral in getting started

with the NetBeans software and also on the software contents of menus and toolbars matching their needs.

The students (52%) acknowledged that discovering new features is not easy using the NetBeans software, and merely 10% find it easy to discover new features. 33% of the students disagree that the software is satisfying to use to accomplish tasks.

Table 4-7: Student perceptions on NetBeans IDE

%NetBeans	Question	Disagree	Neutral	Agree to Strongly Agree	Blank
1	Software easy to use	**0**	**33**	**67**	**0**
2	I am in control of contents of menus & toolbars	**14**	**48**	**38**	**0**
3	I will be able to learn all offered in software	**10**	**52**	**38**	**0**
4	Navigating through menus & toolbars is easy	**19**	**48**	**33**	**0**
5	Software is engaging	**19**	**38**	**43**	**0**
6	Contents of menus & toolbars match my needs	**19**	**43**	**38**	**0**
7	Getting started with the software version is easy	**24**	**43**	**33**	**0**
8	Finding options in menus & toolbars is easy	**24**	**52**	**24**	**0**
9	Software responds in time	**10**	**38**	**52**	**0**
10	Discovering new features is easy	**52**	**38**	**10**	**0**
11	Software is satisfying to use	**33**	**19**	**48**	**0**

4.5.4 Confidence gained using NetBeans on the following object-oriented concepts

The study found that very few students gained confidence in understanding Object-Oriented concepts using NetBeans. For example, the highest percentage who reported to have gained confidence in Class, Method Overriding and Method Overloading was merely 19%. This was followed by the 10% who stated all the other remaining Object-Oriented concepts namely Polymorphism, Encapsulation, Inheritance and Object as depicted in Figure 4-20.

Object had 52% of the students who reported that they gained confidence quite a bit, followed by the 43% who gained confidence quite a bit on Class. 38% stated that their confidence was boosted quite a bit on Inheritance.

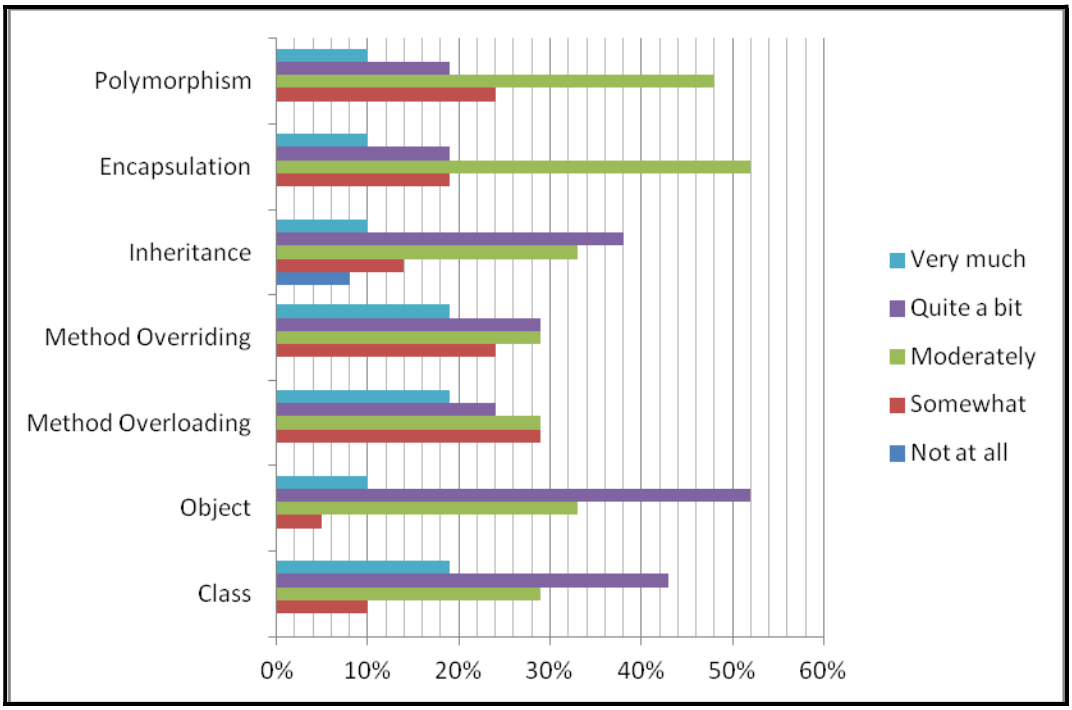

Figure 4-20: Confidence gained from using NetBeans

When it comes to being able to relate between a Class and an Object in NetBeans, 90% of the students affirmed that they can easily relate the two. Only 10% admitted that they cannot relate to and between Class and Object in NetBeans.

4.5.5 Recovering from errors and common mistakes using NetBeans

57% of the students find it easy to very easy to recover from system (IDE) error messages as shown in Figure 4-21. 19% of the students reported that they find it moderate, 5% find it somewhat hard, and the other 10% find it difficult to very difficult to recover from system (IDE) errors. The remaining 10% did not respond.

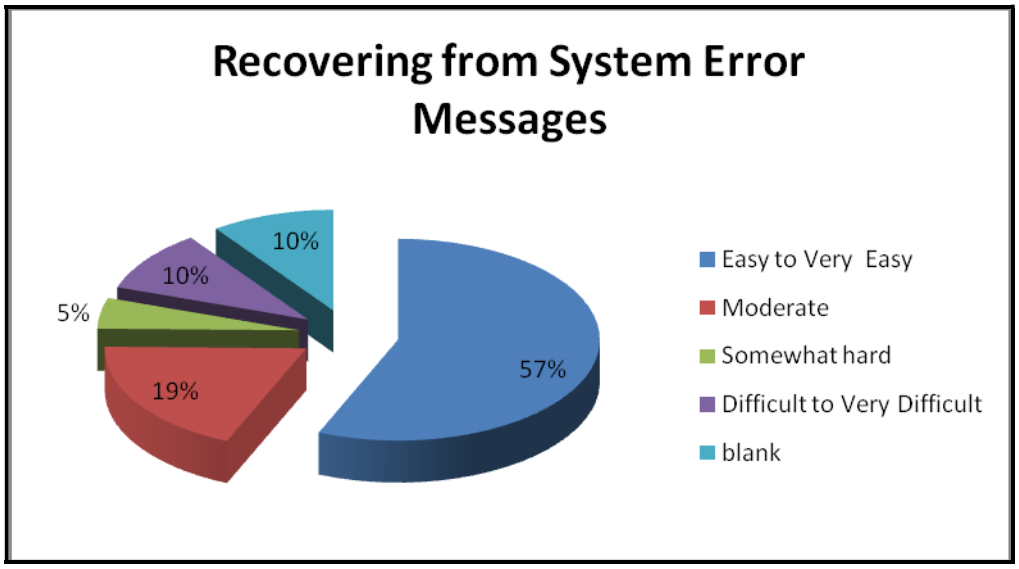

Figure 4-21: Recovering from system (IDE) messages in NetBeans

4.5.6 Recovering when output is not the movement expected

A few students (38%) find it easy to very easy to recover when output (animation) is not the expected movement. The majority, 48%, stated that they find it moderate to recover from unexpected animation. Only 10 % emphasised that it is somewhat hard and 8% stated that it is difficult to recover from unexpected output as shown in Figure 4-22.

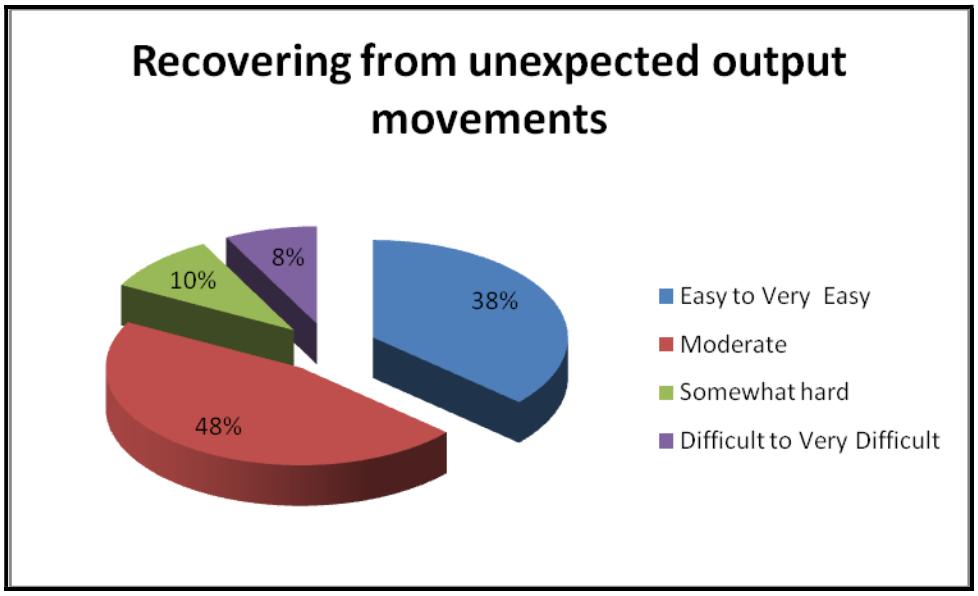

Figure 4-22: Recovering when output is not expected in NetBeans

The students reported that they do make mistakes when using IDE during focus group discussions. The most commonly cited mistakes include leaving out commas or sometimes using a semicolon instead of a comma. Another common mistake is using syntax of VB in Java; using the term or reserved words in an incorrect place. The students admitted that misunderstanding the features in IDE often leads them to make mistakes, for example having exceptions that they do not understand how they come about. They also cited that using the command prompt and program validation are usually challenging.

The focus group discussions confirmed that most of the mistakes made by students come from misunderstanding IDE features, as stated by 57% of the students during the focus group discussions. 43% refuted that their mistakes emanate from misunderstanding IDE features.

4.5.7 Using NetBeans to complete task

Only 38% of the students found it easy to complete tasks using NetBeans software. The majority, 43%, reported that they find it moderate and the remaining 19% stated that it is somewhat hard to complete tasks using NetBeans as indicated in Figure 4-23.

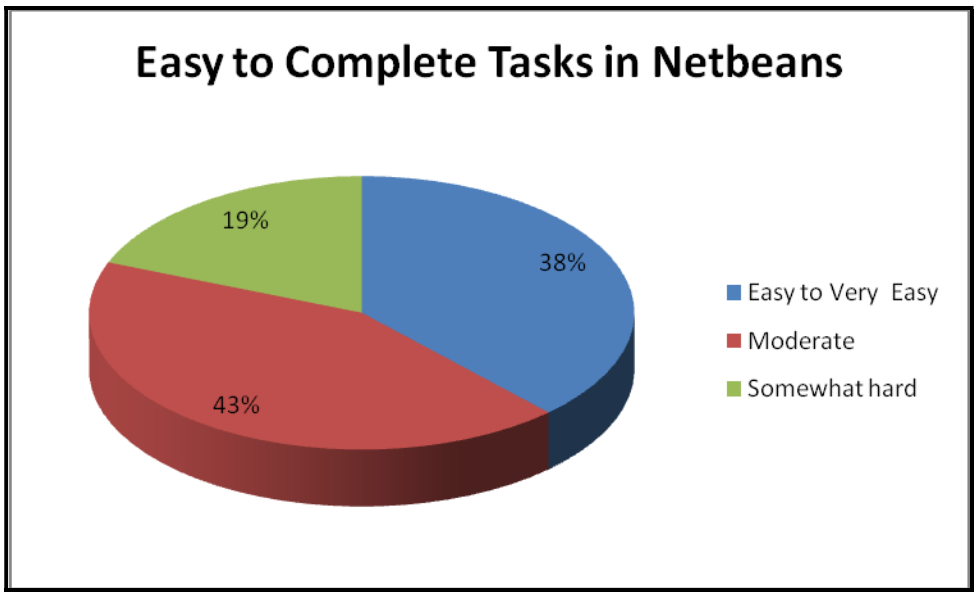

Figure 4-23: Completing tasks in NetBeans

4.5.8 Difficult Tasks to accomplish in JCreator

Students stated that the most difficult tasks to complete in NetBeans are: Method overloading (24%), Graphic User Interface (GUI 19%), and finding and correcting errors (14%). Some of the

tasks mentioned by 5% of the students respectively are namely: debugging, Inheritance, database connectivity, compilation complexities and classes.

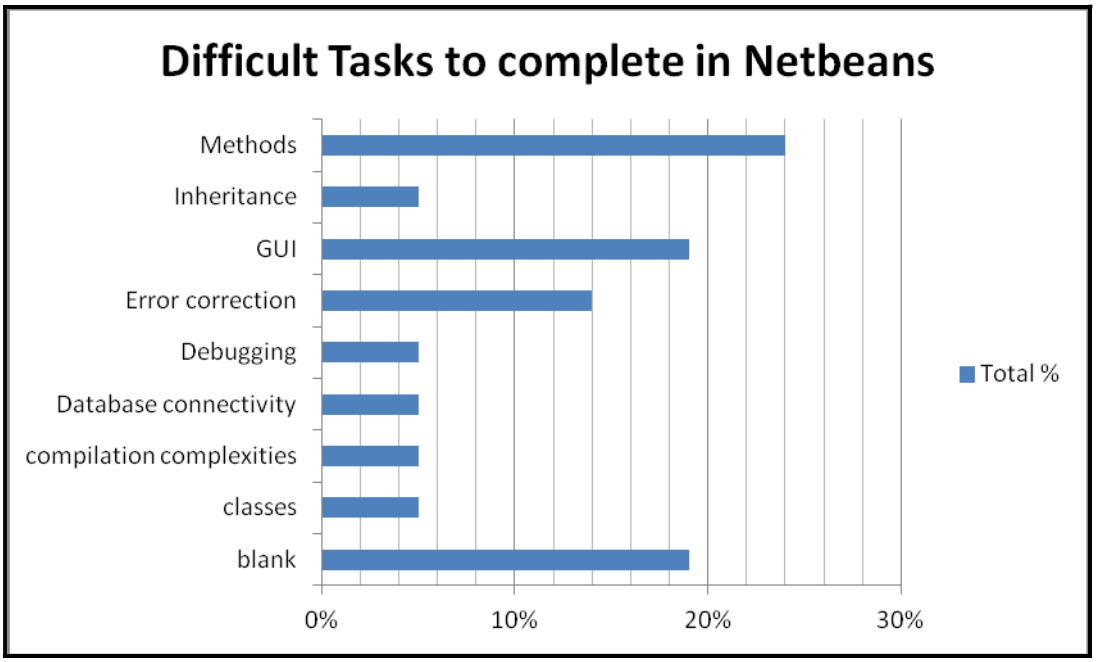

Figure 4-24: Difficult tasks to complete in NetBeans

4.5.9 Misconceptions

Students commented that NetBeans software is quite interesting to learn although it is challenging. For the beginner, it is hard to find the way around and it takes time to find an error. Other students reported that NetBeans software has been very easy to learn and use throughout the year, and they further stated that NetBeans makes programming interesting to continue to pursue. Another student reported that NetBeans matches most of their needs and it is understandable which makes it more interesting to continue learning.

4.5.10 Motivations to learn OOP using Java language

The findings on whether students have any motivation to learn OOP using Java language indicates that the majority of the students felt motivated to learn. Some of them stated that programming games is more motivating when using Java. Another student stated that they are motivated with the introduction of the different types of objects and the Graphic User Interface. In addition, the students cited that when learning OOP using Java, there is no limit; one can do so much with the software.

4.6 Data Analysis and Interpretation: Campus using NetBeans

4.6.1 Difficulties of learning programming using NetBeans

4.6.1.1 Demographics

Gender

It would appear that females find it much easier to use the NetBeans software than males with the findings indicating 100% acknowledgements from females as compared with 56% of the males stating that they find the software easy to use. The remaining 44% are neutral.

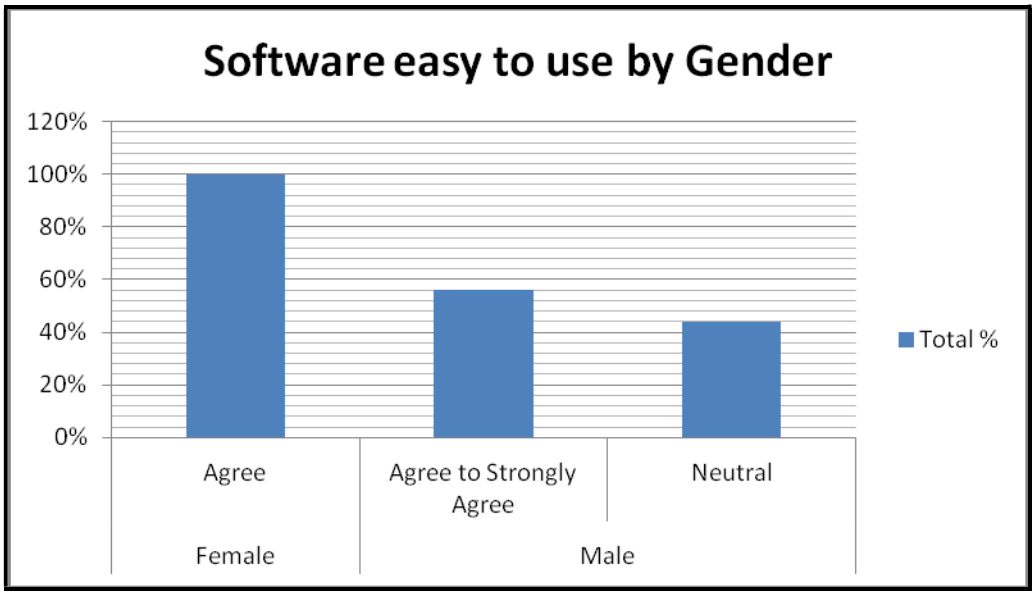

Figure 4-25: Gender analysis

Age

The age group category did not seem to have any effect on the rate of understanding the NetBeans software as both females and males fall between the age groups of 18 – 21, 22 – 25, and 26 – 30.

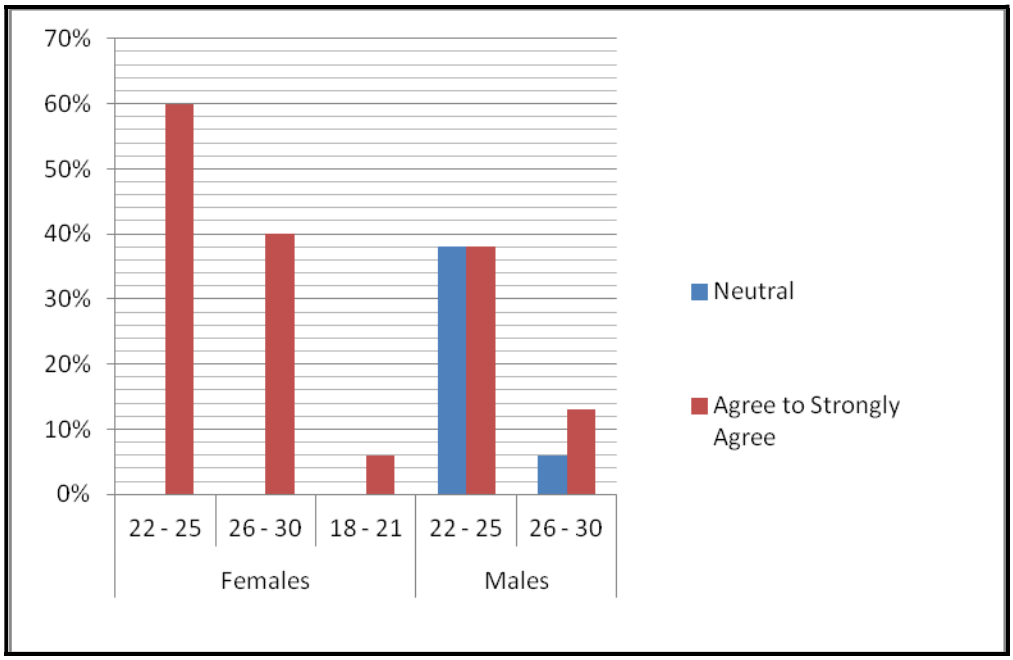

Figure 4-26: Software easy to use per age and gender

4.6.1.2 Programming timeframe and previous programming experience

The study noted that the timeframe of programming has a direct effect on how the students faired with using the NetBeans software. All the females stated that they find NetBeans software easy to use and they all have more than a year of programming experience. Only 44% of the males who stated that they find the NetBeans software neutral have less than one year of programming experience. This could possibly be due to the fact that they have to learn new concepts and they are still trying to familiarise themselves with the programming concepts.

Table 4-8: Software easy to use in relation to timeframe programming

Gender	Software easy to use	Timeframe programming	Total %
Female	Agree	More than 1 year	100
Female Total			**100**
Male	Agree to Strongly Agree	More than 1 year	44
	Neutral	More than 1 year	44
	Agree	Less than 1 year	12
Male Total			**100**

4.6.2 Object-Oriented concepts:

4.6.2.1 Understanding OOP concepts using NetBeans

Most students acknowledged that they find it easy to understand most OOP concept when using NetBeans such as: Class (62%), Object (57%), Method Overloading (43%), and Method Overriding (43%). The least cited OOP concept was Polymorphism with only 24% of the students stating that they find it easy to very easy to understand the concept.

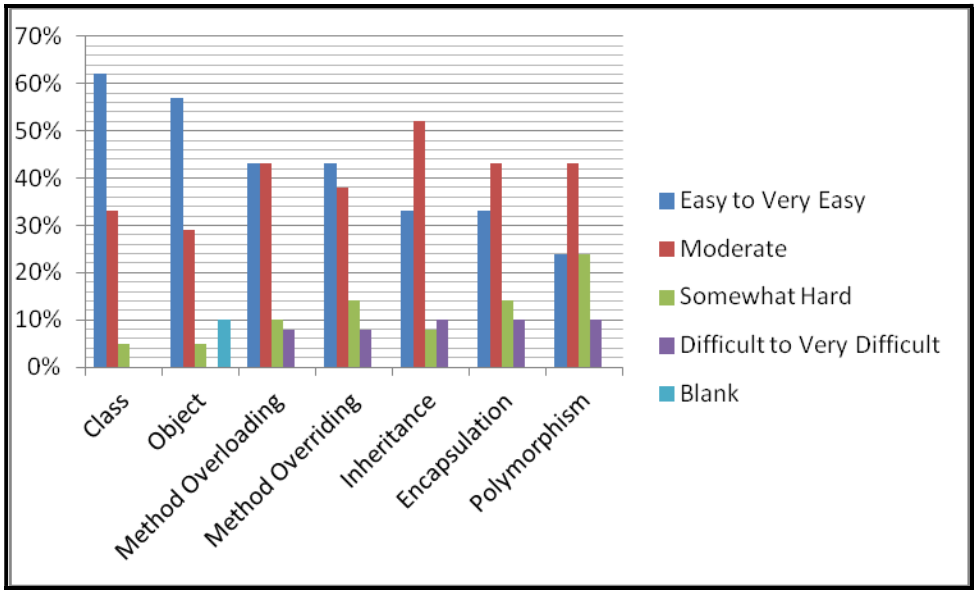

Figure 4-27: Understanding OOP concepts with NetBeans

4.6.2.2 Confidence gained

NetBeans software has been confirmed to have boosted students' confidence on few of the OOP concepts such as Method Overloading (19%), Method Overriding (19%) and Class (19%), all with only 19% of the students' acknowledgement. Most students mentioned that NetBeans has boosted their confidence quite a bit. It can be concluded that even the timeframe of programming has no influence on boosting the confidence gained on understanding OOP concepts. More practice and enthusiasm to use the NetBeans software is recommended to boost student confidence.

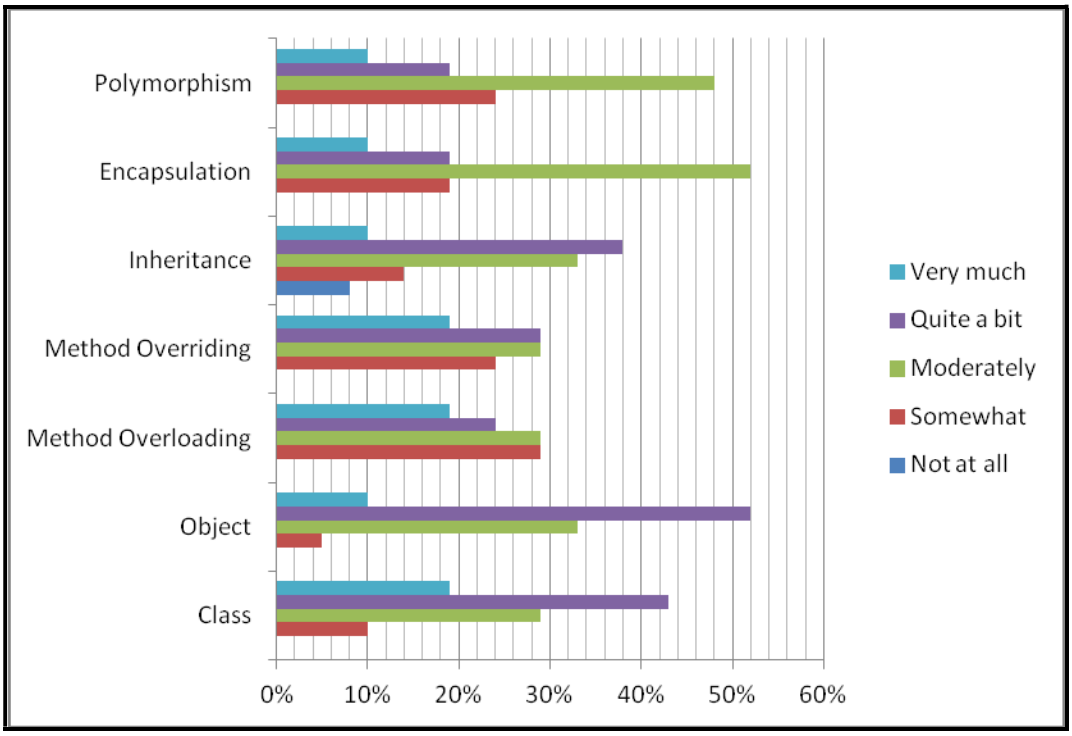

Figure 4-28: Confidence gained on OOP concepts

4.6.2.3 Common mistakes and difficult tasks to accomplish in NetBeans

The study results show that almost half of the students (57%) find it easy to recover from system (IDE) error messages. The majority (48%) find it moderate to recover from unexpected animation. This is supported by the fact that most students find it difficult to fully grasp Object-Oriented programming concepts, especially Polymorphism. Most scholars argue that it is vital for students to do programming by themselves while they are learning (Lahtinen *et al.,* 2005).

The students admitted that misunderstanding the features in IDE often leads them to make mistakes, for example having exceptions that they do not understand how they come about. They also cited that using the command prompt and program validation are usually challenging and they forget to insert the necessary commands such as commas. Such mistakes strongly suggest that students do not dedicate a lot of time to programming and they need to develop interest and determination.

4.6.2.4 Satisfaction in using NetBeans

Figure 4-29 shows that only 48% of the students find NetBeans software satisfying to use with 19% females and 29% males. This may be attributed to the fact that the contents of menus and

toolbars do not match the students' needs as only 38% reported that the contents of menus and toolbars match their needs. Furthermore, 52% of the students admit that they find it difficult to discover new features using NetBeans. This leads students to lack the satisfaction to use the NetBeans software. Storey (2005) suggests using adaptive interfaces which may perhaps be tailored to suit different kinds of users and tasks. This is because software tools normally have many features which may be overwhelming not only to novice users but also to expert users.

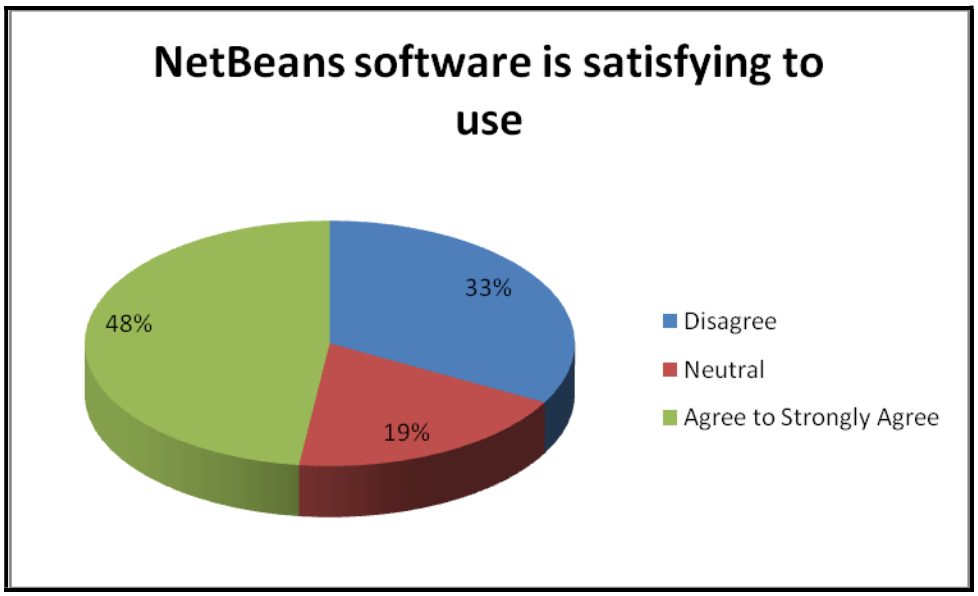

Figure 4-29: Satisfying to use

The study shows that only a few students (38%) will be able to fully learn all that is offered in NetBeans. Although the NetBeans software responds on time, it needs to be tailored more to suit students' expectations and more independent learning may assist to increase the student level of understanding the software. The study also shows that 62% of the students find it difficult to complete tasks using NetBeans software. This further contributes to the students finding NetBeans dissatisfying as they struggle to complete tasks.

4.6.3 Motivation to learn OOP

Students felt motivated to learn OOP using Java when they are working on work that they can easily visualise the outcomes, for instance games. Kasurinen *et al.* (2008) support this notion as they stated that students often lose interest in programming because complex models and structures have to be learned before anything visually impressive can be created. The students realise that they can do so much when learning OOP using Java.

4.7 Student results

The study went further to check on the student results from the different campuses, one that uses JCreator and the other that uses NetBeans. The study seeks to get an objective reflection on how the students are fairing with the software, other than basing the conclusions on students' perceptions.

Figure 4-30 depicts that for test 2 and test 3, students using JCreator scored the highest with an average of 80% as compared to the students using NetBeans who scored an average of 68% and 70%. However on the average year test marks, the findings show that both students using JCreator and those using NetBeans were at par scoring an average of 70%. Students using JCreator scored less compared to students using NetBeans with an average difference of 15.2% for test 4. Although the students from both campuses were performing well above average (50%), the final results show that students using NetBeans performed far better compared to the students using JCreator.

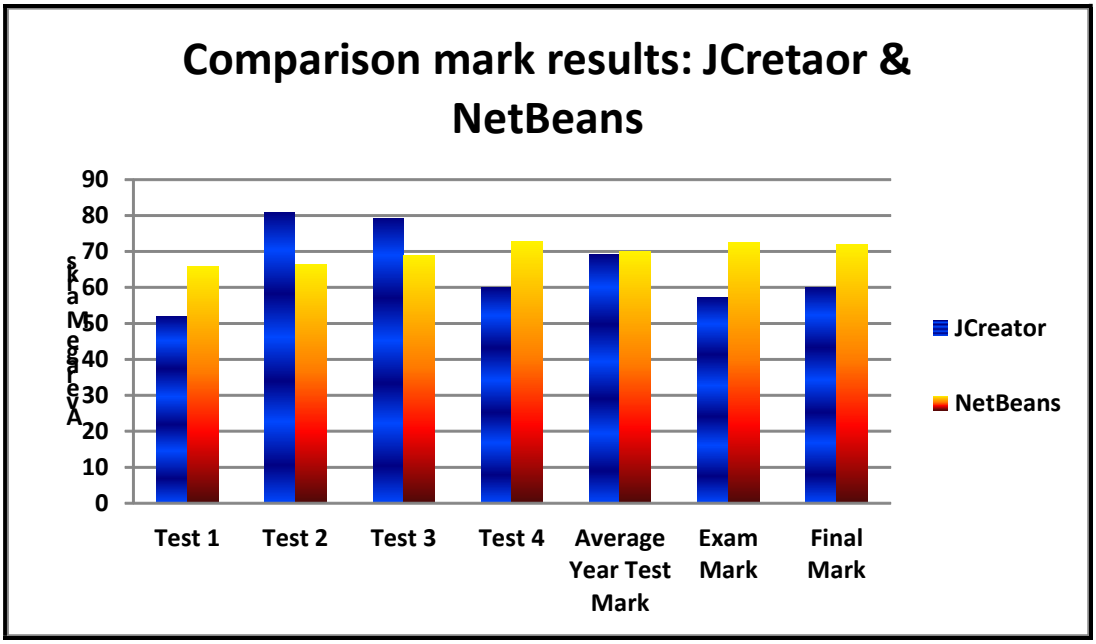

Figure 4-30: Tests and Exam comparison

It can be argued that since students from both campuses scored low on test 1 (slightly above average), the students were still learning the software and trying to grasp what the various object oriented programming concepts meant. As they proceeded doing other tests their results improved gradually, especially on JCreator. On the other hand, students using NetBeans

improved gradually as well and their results are more consistent as compared with the students using JCreator. Overall the findings indicate that students using NetBeans outperformed students that use JCreator.

The results from the actual student class marks, which indicate that students using NetBeans perform better than students using JCreator, are in agreement with the students' responses on how they find the software and other elements easy to use (see Table 4-9). Table 4-9 has 67% of the students using NetBeans rating the software as easy to use compared to the students using JCreator with only 56%. In addition, the students using NetBeans rated the software as responding on time (52%) and as engaging and satisfying to use compared to the students using JCreator. Although the results in other areas were below 50%, NetBeans recorded the highest on contents of menus and toolbars matching students' needs.

Table 4-9: IDE satisfaction

		JCreator	NetBeans
	Statement	Agree to Strongly Agree	Agree to Strongly Agree
1	Software easy to use	56	67
2	I am in control of contents of menus & toolbars	44	38
3	I will be able to learn all offered in software	47	38
4	Navigating through menus & toolbars is easy	50	33
5	Software is engaging	24	43
6	Contents of menus & toolbars match my needs	29	38
7	Getting started with the software version is easy	38	33
8	Finding options in menus & toolbars is easy	47	24
9	Software responds in time	41	52
10	Discovering new features is easy	24	10
11	Software is satisfying to use	32	48

The student test and exam results, however, conflict with the findings from the students' perception on how they understand the OOP concepts as shown in Figure 4-31. Students using

JCreator stated that they understand most Object-Oriented concepts better as compared to the students using NetBeans. Conversely, the class tests and exam results depict that students using NetBeans performed better than students using JCreator. Possibly the students using NetBeans underestimated their level of understanding of the OOP concepts.

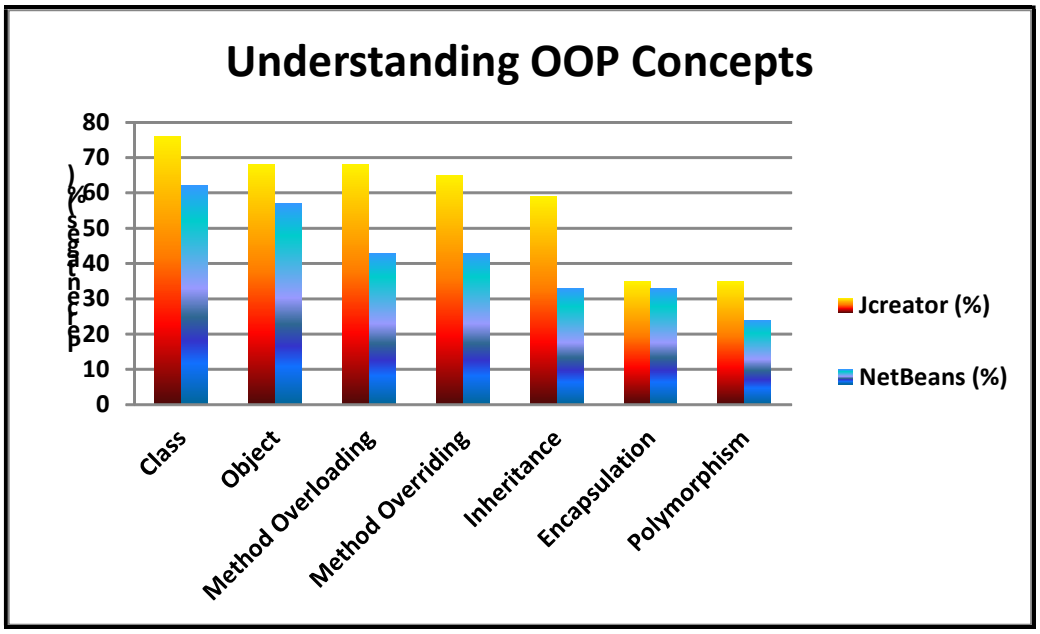

Figure 4-31: Understanding OOP concepts

It could also be that the lecturers who marked the NetBeans students' exams could have been lenient when assessing students' answers. Other scholars suggest that student performance on examinations is influenced by the level of difficulty of the questions(Sheard *et al.*, 2013).

Students could have misunderstood the exam questions or some were frightened by the mere fact they are seating for an exam. Moreover, the teaching methods of the lecturers could have influenced on both the perception and student class test and exam results. It is probable that the lecturers for the JCreator students are enthusiastic and motivate the students to enjoy programming, and as some result students end up believing that the JCreator software is easy to use. Whereas the lecturers for the NetBeans students maybe more concerned about students' achievements on tests and exams, but do not go the extra mile to assure students that the software is manageable and easy to use. This is evidently shown by more students using JCreator stating that they feel more confident to learn OOP concepts using JCreator compared to the students using NetBeans. So, it could be the different motivation levels that the students receive that have a direct influence on students' perceptions.

4.8 Empirical Findings

4.8.1 JCreator and NetBeans: a comparison

This section makes a comparison between JCreator and NetBeans based on the students' responses to the questionnaires, focus group discussions, class test marks and exam marks. Before presenting the comparison, the study made a comparison of student respondents' gender and age for the two campuses as shown in Table 4-10 to determine if there are any significant variances that may affect the results. For the campus that uses JCreator, there were more females than males. More males than females were recorded at the campus that uses NetBeans for the study. The age category 22 – 25 dominated for both campuses. There were slightly more students responding from the campus where they are using JCreator. The study sought to understand if the gender and age categories had any effects on how students understand programming and their understanding of the Object-Oriented concepts. Even though previous researchers indicated that females perform better than males, the study found that the test and exam mark results confirmed that students who used NetBeans performed better than the students using JCreator. The most interesting aspect to note is that there were more males than females at the best performing campus that uses NetBeans as compared to JCreator, thus the study concludes that the gender of the students were not much of a determinant factor on students' performance and understanding of OOP concepts.

Table 4-10: Age category and Gender

Age Category	JCreator		NetBeans	
	Male	Female	Male	Female
18 – 21	5	7	1	0
22 - 25	9	10	12	3
26 - 30	2	1	3	2
Total	16	18	16	5

4.8.2 Timeframe of programming

The study also made a comparison of students' timeframe of programming as it may have an effect on how the students understand OOP concepts. The results show that both campuses, as illustrated in Table 4-9, have more students with more than one year of programming. All the females from the campus using NetBeans had more than one year of programming experience. The results are in agreement with studies conducted by Byrne and Lyons (2001) and Allert (2004) which identified that students with previous programming experience tend to perform

better. It can be confidently concluded that previous programming experience has a positive influence on how students understand the OOP concepts. This is not only affirmed by the perception-based responses but also the students' final year mark results, which speaks volumes as in both campuses students passed with more than 50% marks.

Table 4-11: Timeframe of programming

	JCreator (%)		NetBeans (%)	
	Males	**Females**	**Males**	**Females**
Less than one year	**12**	**6**	**12**	**0**
More than one year	**88**	**94**	**88**	**100**

After having looked at the general students' demographics, the study then goes into detail through the various sub-headings to critically analyse the results as per each research question stated in chapter 1 under the sub sub-heading issue as follows.

4.8.3 First research sub-question

- Does the use of an IDE affect students' understanding of OOP concepts?

This sub-question focuses on investigating the students' perception of usability on IDEs and understanding of OOP concepts. A detailed discussion of the findings to the first research sub-question is provided below.

Understanding OOP Concepts using JCreator and NetBeans

The perception-based results derived from the study showed that the use of JCreator or NetBeans by student respondents in understanding the object-oriented concepts of Class and Object do not appear to differ. However, the use of JCreator to the concepts of Method Overloading, Method Overriding and Inheritance does appear to make a difference as majority of the students reported that they understand these Object-Oriented programming concepts better as compared to the students using NetBeans. For both campuses using JCreator or NetBeans, both IDEs do not seem to assist students to better understand Encapsulation and Polymorphism as both scored merely 33% of the students admitting understanding these OOP concepts. This means that the use of IDEs have a significant impact for students understanding OOP concepts, although not all of them as shown in Table 4-12.

Table 4-12: Understanding OOP concepts JCreator and NetBeans

Understanding OOP Concepts	JCreator (Out of 34 Students)		NetBeans (Out of 21 Students)	
	Easy to very easy to understand		Easy to very easy to understand	
	%	Actual no.	%	Actual no
Class	76	26	62	13
Object	68	23	57	12
Method Overloading	68	23	43	9
Method Overriding	65	22	43	9
Inheritance	59	20	33	7
Encapsulation	35	12	33	7
Polymorphism	35	12	24	5

Coping with Errors

Based on the perception-based results, Table 4-13 showed how respondents coped with errors using the programming tools. For System (IDE) Error Messages, NetBeans had 57% of the students stating that they find it easy to very easy to recover from system (IDE) error messages, unlike in JCreator which only recorded 44% of student acknowledgement. Recovering from wrong output expectedly was recorded to be difficult with few students agreeing that they can easily recover: JCreator (32%) and NetBeans (38%). The study gathered that since most students using NetBeans find it easy to very easy to recover from system (IDE) error messages and recovering from unexpected wrong output, it implies that they had a good and firm understanding of the IDE and the programming language which is an essential part of computer science and the key factor to succeed in programming.

Table 4-13: Coping with Errors

Types of Errors	JCreator Easy to very easy	NetBeans Easy to very easy
System (IDE) Error messages	44	57
Output	32	38
Completing Tasks	50	38

Interestingly, when it comes to being able to complete tasks, more students using JCreator (50%) admitted that they are able to complete tasks. Merely 38% of the students using NetBeans stated that they are able to complete tasks. This interesting perception-based result

on completing tasks could be attributed to the fact that students at the campus using JCreator have more confidence than students at the campus using NetBeans. In actual fact, the exam final mark results prove that even though the students at the campus using JCreator have more confidence in relation to the students at the campus using NetBeans, the NetBeans students have managed to grasp the programming better than the JCreator students, thus making NetBeans a more favourable software to use in understanding OOP concepts and to basically learn programming. The next section will address the second research sub-question which addressed the confidence gained in learning OOP concepts during IDE use of the research study.

4.8.4 Second research sub-question

- Does the use of a particular IDE increase students' confidence in learning Object–Oriented concepts?

This sub-question focuses on investigating whether the use of an IDE increases students' confidence in learning OOP concepts. A detailed discussion of the findings to the second research sub-question is provided below.

Confidence in Learning Object-Oriented Concepts

Based on the perception-based results, Table 4-14 described how JCreator and NetBeans increased respondents' confidence in learning object-oriented programming concepts. The results show that when it comes to Class Object-Oriented programming concept, more students using JCreator gained confidence (53%). For the Object-Oriented concepts Encapsulation and Polymorphism, students gained less confidence with the use of either JCreator or NetBeans.

Although JCreator scored slightly higher than NetBeans in increasing students' confidence, the scores were still too low all ranging between 32% – 38%. Thus, it can be safely concluded that for all object-oriented programming concepts, there was no significant difference between the uses of JCreator or NetBeans in increasing student respondents' confidence. The worse recorded was OOP concepts Encapsulation and Polymorphism. The notion that students from the campus using JCreator tend to be over-confident as compared to the students using NetBeans could be pointed to as the reason why the perception-based results depicted in Table 4-14 are like that. Students from the campus using NetBeans need to be highly motivated so that they can realise the great potential they have when it comes to programming and

understanding OOP concepts. The fact that majority of the NetBeans campus students are able to recover from system (IDE) error messages and unexpected outputs as well as scoring high marks based on the final exam marks, which is over 70%, is evidence enough to show their great potential.

Table 4-14: Confidence in learning OOP concepts

OOP Concepts	JCreator		NetBeans	
	Very Much (%)	Actual no	Very Much (%)	Actual no
Class	53	18	19	4
Object	32	11	10	2
Method Overloading	38	13	19	4
Method Overriding	38	13	19	4
Inheritance	38	13	10	2
Encapsulation	18	6	10	2
Polymorphism	15	5	10	2

Student perceptions on JCreator compared to NetBeans

Students from both campuses were asked to rate on the quality of the software installed on their machines, JCreator or NetBeans. NetBeans scored highest on being easy to use (67%) as compared to JCreator with 56%. NetBeans also had high scores on software responding on time (52%), engagement of the software (43%), and on satisfaction that students get using the software (48%) as shown in table 4-15. This is highly attributed to the fact that students using NetBeans software had a firm foundation in terms of understanding the basics of programming and they find it easy to apply the concepts that they learned as compared to the students using JCreator. When students understand the basic programming from its introductory level they tend to find it satisfactory to use.

Although below half compared to NetBeans, JCreator scored high in terms of students being in control of contents of menus & toolbars (44%); able to learn all offered in software (47%); navigating through menus & toolbars is easy (50%); and finding options in menus & toolbars is easy (47%).

When it comes to discovering new features; contents of menus & toolbars matching students' needs, and getting started with the software version is easy, both JCreator and NetBeans are lowly ranked. In as much as the NetBeans software may be the ideal software to use to understand OOP concepts, it also has its downside as reflected by the students' perception-

based responses on how they are faring with discovering new features, being in control of contents of menus & toolbars and the other aforementioned features.

Table 4-15: Students' perception of JCreator compared to NetBeans

		JCreator	NetBeans
	Statement	**Agree to Strongly Agree**	**Agree to Strongly Agree**
1	Software easy to use	**56**	**67**
2	I am in control of contents of menus & toolbars	**44**	**38**
3	I will be able to learn all offered in software	**47**	**38**
4	Navigating through menus & toolbars is easy	**50**	**33**
5	Software is engaging	**24**	**43**
6	Contents of menus & toolbars match my needs	**29**	**38**
7	Getting started with the software version is easy	**38**	**33**
8	Finding options in menus & toolbars is easy	**47**	**24**
9	Software responds in time	**41**	**52**
10	Discovering new features is easy	**24**	**10**
11	Software is satisfying to use	**32**	**48**

The following section addresses mistakes and misconception students face.

4.8.5 Third research sub-question

- What are the most common mistakes and misconceptions students make during program development in a particular IDE?

This sub-question focuses on mistakes and misconceptions students make while using IDEs. A table below discusses the findings to the third research sub-question.

Common Mistakes

JCreator	NetBeans
Findings from the focus group discussions had most students admitting that they usually make spelling mistakes. For instance, they misspell Class names, variables and many others since the IDE does not correct spellings. They also confuse the lower and upper case. When compiling, students make the mistake of running the program and they find it difficult to stop it. One of the critical errors stated was that the students fail to create the correct object. They only try to create it after they realise that it is needed. The other error is making all Class variables private. It was also stated that students find it difficult to debug. On the other hand, some students admitted that they simply forget certain things and they feel that they need more motivation and to remain focused so as to avoid these mistakes.	The students reported during focus group discussions that they do make mistakes when using IDE. The most commonly cited mistakes include leaving out commas or sometimes putting a semicolon instead of a comma. Another common mistake is using syntax of VB in Java, using the term or reserved words in an incorrect place. The students admitted that misunderstanding the features in IDE often leads them to make mistakes, for example having exceptions that they do not understand how they come about. They also cited that using the command prompt and program validation are usually challenging.
Mistakes come from misunderstanding IDE features	
Generally, the findings show that most students find it easy to learn using IDE. They acknowledge that although it can be challenging, it is quite interesting. IDE requires much attention and one needs to be willing to explore the tools, controls and features. Students stated that it would be much easier to learn using IDE if it could locate and fix errors.	The focus group discussions confirmed that most of the mistakes done by students come from misunderstanding IDE features as stated by 57% of the students. 43% refuted that their mistakes emanate from misunderstanding IDE features.

The mistakes stated by students from both campuses do not suggest usability problems in IDEs, but they show an outcry that students need to be motivated to grasp basic programming concepts. Certain activities, for instance spelling mistakes and forgetting to put commas and other commands where they are necessary, show that the students are not dedicating much of their time to make an effort to grasp the vital skills. However, on the other side, the IDEs must at least have in-build capabilities to at least try to limit the number of mistakes and errors that students make while using the software packages. The next section provides an analysis from the experts lecturing Object-Oriented concepts in Java.

4.9 Expert review

4.9.1 Understanding object-oriented concepts using NetBeans

The research went further to interview the lecturers (experts) on how students are faring with understanding the Object-Oriented concepts. The findings show that 67% of the experts stated that students find it easy to very easy to understand Class, Method Overloading and Method Overriding. Only 33% reported that they find it easy to very easy to understand Object, the remaining 67% find it moderate.

Only 33% of the lecturers stated that students find it somewhat hard to understand OOP concepts such as Class, Inheritance and Polymorphism. Most of the lecturers (67%) interviewed acknowledged that Encapsulation is difficult to very difficult to understand. The remaining 33% respectively cited Method Overloading, Method Overriding, Inheritance and Polymorphism as being difficult to very difficult to understand as depicted in Figure 4-32.

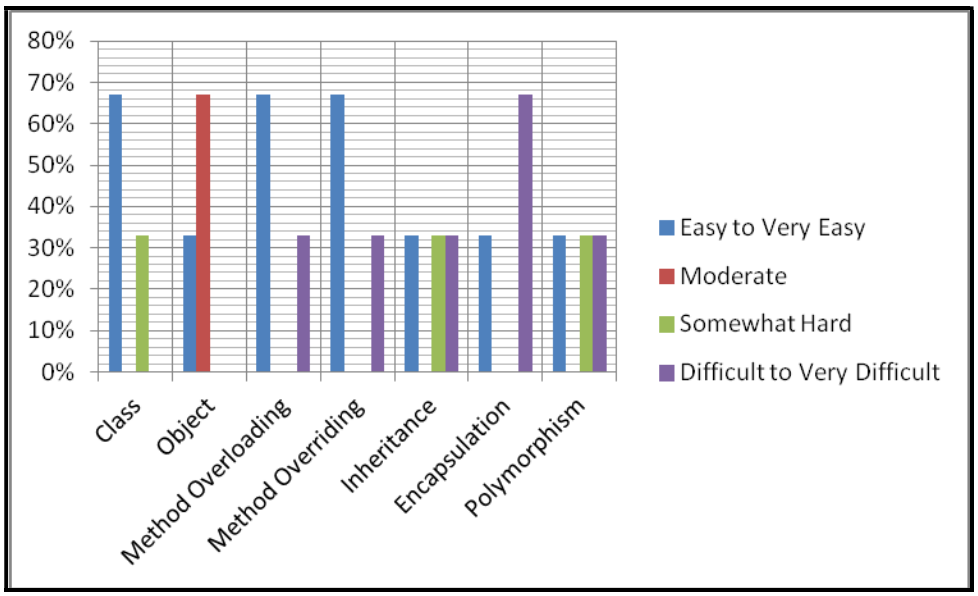

Figure 4-32: Understanding OOP concepts from experts

4.9.2 Recovering from errors and common mistakes using IDEs

The lecturers admitted that it is not easy to recover from system (IDE) errors with 33% stating that it is moderate and 33% reporting that it is somewhat hard. Only 33% reported that it is easy to recover from system (IDE) errors.

Table 4-16: Recovering from unexpected output

System (IDE) error messages	Total
Easy	33%
Moderate	33%
Somewhat hard	33%
Grand Total	99%

Most of the lecturers (67%) who responded to the questionnaire stated that they find it easy to very easy to recover when output is not the result expected, and only 33% stated that it is moderate.

4.9.3 Difficult tasks to accomplish in JCreator

The lecturers highlighted Inheritance and testing classes as some of the most difficult tasks to accomplish when using IDE.

The findings from the experts do not tally with students' perceptions. This symbolises a different level of understanding by experts and students. The observation made by lecturers carries more weight as these are the people with more knowledge in the field.

4.9.4 Comments

IDEs like NetBeans help students identify syntax errors, assist students in construction of code segments and visual forms designer. The lecturers emphasised that programming requires practicing and comprehensive approach whether it is on structures or object-oriented. They further advised that it is important for students to experience an active learning environment which is a result of pedagogical shifts, learning and authentic problem alignments. This curbs the fact that similar features are not always understood by students.

4.10 Summary

In this chapter the results from the interview schedule, questionnaire, test and exam marks were analysed and discussed. These results were compared to the findings of the critically reviewed literature to find out whether the results of the primary data are consistent with the explanations of the findings. This was achieved through inductive analysis and the results of the primary data are found to be consistent with some of the explanations.

The next chapter (Chapter 5) gives a full summary of the research findings and its contribution.

CHAPTER FIVE : CONCLUSION

5.1 Introduction

The aim of this research has been to contribute towards the teaching and learning of OOP concepts by interpreting the usability of an interactive Object-Oriented programming (OOP) development environment for teaching and learning of Java programming language that enhances OOP comprehension through quality of use. To accomplish the previously stated goal, the thesis adopted a single case strategy to analyse in detail the factors that influence OOP concepts comprehension. It is argued that some students are incapable of fully understanding and utilising the feature set of Integrated Development Environments (IDEs), thus affecting the comprehension of OOP concepts.

5.2 Overview of the research

The preceding chapters consist of a comprehensive introduction of the study, reviewed literature, research methodology, findings and interpretations of data were discussed. The identified problem and main objective of this study are explained in Chapter 1. Chapter 2 provides a thorough investigation into the nature of teaching and learning of object-oriented programming. This was achieved by first outlining difficulties of learning computer programming; this was further supported by looking at the Object-Oriented paradigm and thereafter highlighting on student cognition with regard to their behaviour and conceptualisation of concepts. Secondly, the various programming tools and specific IDEs were explored. Lastly, a discussion on ISO 9126 as a framework to evaluate the quality of use of IDEs was deliberated. These deliberations led to the formulation of a problem conceptualisation diagram on which this research was concentrated. The procedures carried out in collecting the data are presented in the research methodology in Chapter 3. Chapter 4 presented the findings, data analysis and interpretation obtained from the questionnaire and interview schedule conducted on students and experts. The interpretations of the student perceptions were further compared with the tests and exam results in an effort to have an objective overview of students' performance.

All these chapters have assisted in addressing the main objective of this study which is to examine the perceptions of students on interactive environments for teaching Object-Oriented concepts using the Java programming language in two integrated development environments.

This chapter concludes the research effort by analysing how each chapter has contributed towards addressing the research questions. The next section continues with the discussion of the research contributions and their implications. The contribution of the study is assessed using criteria formulated by Whetten (1989). The final section discusses opportunities for further research.

5.3 Research contributions

This section reviews the theoretical, the methodological and the practical contributions of the research.

5.3.1 Theoretical Contributions

The ISO 9126 model was used in this study in order to evaluate software quality of use. ISO 9126 is defined by six software quality characteristics: functionality, reliability, effectiveness, usability, maintainability, portability and 22 sub-characteristics. However, to test student usage of IDEs and appreciation of OOP concepts, only four characteristics and twelve sub-characteristics were tested on two IDEs. This consisted of selecting generic external system quality characteristics and sub-characteristics that fit student evaluation of an IDE. The proposed model was applied on NetBeans and JCreator LE 5.0 as IDEs that are used for the development of Java programs using OOP concepts. This study has proven that applying selected characteristics and sub-characteristics of the ISO 9126 theoretical model can be used to ascertain and evaluate software only in certain areas of interest that benefit the user. A brief discussion including primary data used to validate the model is provided in the research methodological contribution section below.

5.3.2 Methodological Contributions

This study employed the mixed method approach for collecting and analysing data. The research philosophies and research methodologies followed in this study combined both positivistic and phenomenological approaches. These philosophies contributed to gathering the information required. A large part of the work focused on the intended users using a single case analysis. This was conducted on two groups of students doing Java programming at second year Computer Studies (CS2) on two different campuses at the selected University. The integration of these two paradigms provided a broader context to the students' perceptions and a better understanding of the different angles in which the research problem was handled. The

model was validated by collecting in-depth primary data through questionnaires, semi-structured interviews with 34 students using JCreator, 21 students using NetBeans, and 3 experts in Java programming; their responses were compared to the common tests and exam students had written and supported by literature. This study used an Excel spreadsheet to present the data numerically and graphically. It followed an inductive approach to analyse the collected data so as to discover whether theoretical explanations of literature support or oppose the findings of primary data. It was found that the findings of primary data are consistent with the literature findings. The study focused on a detailed understanding of a specific environment and groups from the selected University and may look to offer generalisable knowledge to other similar settings. The next section gives a detailed discussion of the practical contributions which are core to this research.

5.3.3 Practical Contributions

5.3.3.1 JCreator IDE software

Generally, students find it easy to use JCreator software, with females recording the highest. The results indicate that females aged 18 to 21 easily understand and grasp concepts and as they grow older the level of understanding drops. Conversely, males perform better when they are in the age group category of 22 – 25. Students who had previous programming experience tend to perform better than other students with no programming experience.

The study found that generally students understand most of the OOP concepts, especially Class, Object, Method Overloading, Method Overriding and Inheritance. A good understanding of Object tends to give the student a good potential to carry out most tasks.

The study realised that students struggle to understand Encapsulation and Polymorphism. The fact that JCreator does not fully build confidence for students to understand most of the OOP concepts could be the reason why most students find it difficult to fix errors and to recover from unexpected movements.

The results indicate that the use of IDEs does not guarantee that students will be able to fully understand OOP concepts. Moreover, students struggle to save multiple files of classes and solving exceptions as well as Polymorphism.

The fact that students do not fully understand the JCreator software leads them to believe that

JCreator IDE is more theoretical and difficult to be practically implemented in the real world. In as much as students stated that they do not find much satisfaction in using JCreator, the study found that JCreator has great potential to capacitate students to comprehend programming and moreover understand OOP concepts as almost half of the students acknowledged that if properly mentored, they will be able to use all that the software has to offer.

It would appear that poor understanding of the basics of programming tends to de-motivate students, thus strategies need to be in place on how to boost student motivation. The idea of pairing students when programming may enhance the quality of their programs and encourage them to pursue programming further. More efforts need to be effected to further motivate students to unlock their potential and sharpen their skills of comprehension and memorisation abilities which are crucial to programming.

5.3.3.2 NetBeans IDE software

The study found that students find it easy to use the NetBeans software with 100% females and 56% males' acknowledgement. The age group category did not seem to have any effect on the rate of understanding of NetBeans IDE software. The timeframe of programming had a direct effect on how the students faired using the NetBeans software with students with more than one year of programming experience reporting that they find NetBeans software easy to understand.

NetBeans had most students acknowledging that they find it easy to understand most OOP concepts except Polymorphism. On the other hand, it can be concluded that even the timeframe of programming have no influence on boosting the confidence gained in understanding OOP concepts. More practice and enthusiasm to use the NetBeans software is recommended to boost student confidence.

More than half of the students find it easy to recover from system (IDE) error messages and the majority find it moderate to recover from unexpected output. The study findings also affirmed with most scholars that in addition to paired programming, it is vital for students to do programming by themselves while they are learning (Lahtinen *et al*., 2005).

Misunderstanding the features in IDE often leads students to make mistakes, for example having exceptions that they do not understand how they come about. Mistakes strongly suggest that students do not dedicate a lot of time to programming and they need to develop interest and determination.

Very few students find the NetBeans software satisfying to use and they also find it difficult to complete tasks. This may be attributed to the fact that the contents of menus and toolbars do not match the students' needs. In addition, the study also shows that only a few students will be able to fully learn all that is offered in NetBeans. Although the NetBeans software responds on time, it needs to be tailored more to suit students' expectations and more independent learning may assist to increase the student level of understanding the program. Students feel motivated to learn OOP using Java when they are working on tasks that they can easily visualise the outcomes for, for instance games.

5.3.3.3 Summary of practical contributions

The results derived from the study showed that the use of JCreator or NetBeans by student respondents in understanding the object-oriented concepts of Class and Object do not appear to differ. Both campuses using JCreator or NetBeans do not seem to assist students to better understand Encapsulation and Polymorphism OOP concepts.

Most students find it easy to recover from system (IDE) error messages using NetBeans unlike in JCreator. Recovering from wrong output unexpectedly was recorded to be difficult using JCreator or NetBeans software.

More students using JCreator admitted that they are able to complete tasks as compared to the students using NetBeans. Although JCreator scored slightly higher than NetBeans in increasing students' confidence, the scores were still too low. Thus, it can be safely concluded that for all object-oriented programming concepts, there was no significant difference between the uses of JCreator or NetBeans in increasing student respondents' confidence. The worse recorded was OOP concepts Encapsulation and Polymorphism.

NetBeans scored highest on being easy to use as compared to JCreator. NetBeans also had high scores on software responding on time, engagement of the software and on satisfaction that students get using the software. When it comes to discovering new features; contents of menus & toolbars matching students' needs, and getting started with the software version is easy, both JCreator and NetBeans are lowly ranked.

The study findings are a bit conflicting as the students' perception results show that JCreator is much better to use as programming IDE and the student test and exam marks show that

NetBeans is easier as most students using NetBeans had better results compared to students that use JCreator.

The study concludes that quality of use of an IDE increases the confidence in students to learn Object-Oriented concepts. In this study NetBeans is better programming software based on the class tests and exam results as the results give an objective picture of how students fared. The perception findings point out that NetBeans scored highest on being easy to use (67%) as compared to JCreator (56%). Moreover, 57% of the students using NetBeans acknowledge that it is easy to recover from system (IDE) error messages compared to JCreator with only 44%. This signifies that students using NetBeans have a better understanding of the software compared to students using JCreator as they struggle to recover from system (IDE) error messages. Inasmuch as perceptions are important, they tend to be subjective. However, further work needs to done on various IDEs to accommodate other programming environments to show how they enhance Object-Oriented concepts comprehension in students learning to program in Java.

In the next section, the contributions in this research are assessed.

5.3.4 Assessing the contributions

Whetten (1989) identified four aspects to be taken into account as part of an assessment of the contribution made by a research study to the body of knowledge in the particular field. These are as follows:

1. What? What factors and concepts should be included as part of the explanation of the contribution? This involves the inclusion of all relevant factors and parsimony. However, it excludes those factors that have little role to play in improving the understanding of the contribution.
2. How? Subsequent to the identification of the factors and concepts which are part of the contributions, the researcher should reflect on how these factors are interrelated.
3. Why? Why select certain factors? That is, what are the underlying assumptions of the theory or model? This means that the logic of the proposed conceptualisation should be of interest to other researchers.
4. Who, where and when? These enquiries define the boundaries for generalisation.

Given Whetten's (1989) framework for the assessment of the contribution of this research, the following questions are asked to help assess the contribution.

What is new? What is new in this research study which makes a significant contribution to current thinking?

This study has contributed in the following ways: Firstly, the relevant literature was reviewed that pertains to students' difficulties in understanding OOP concepts. This was supported by rereading the cognitive skills which affects knowledge retention in a student. Examining various programming tools with their characteristics and how they help a student to program OOP concepts was also a major contribution.

Secondly, the contribution gives an insight into the single case study by looking at students' perception towards programming OOP concepts in this environment. It shows that poor understanding of the basics of programming tends to de-motivate students, thus strategies need to be in place on how to boost student motivation. This study also contributes by suggesting the idea of pairing students when programming to enhance the quality of their programs and encourage them to pursue programming further. More efforts need to be effected to further motivate students to unlock their potential and sharpen their skills of comprehension and memorisation abilities which are crucial to programming.

Lastly but not the least, the contribution lies in the application of the ISO 9126 model to evaluate software quality of use and the role software (IDE) play to encourage students to program. However, the ability of a student to use a certain IDE effectively does not portray their true knowledge of OOP concepts comprehension. This is because students who had difficulty in using the software had better exam results.

So what? Is the theory likely to change in the way teaching and learning of OOP concepts with particular IDEs?

Relying on the understanding that proper use of IDEs can encourage students to program and understand OOP concepts could be coupled with the fact better motivated students make good programmers. Thus carefully analysing the findings of this study and applying them in a learning environment would improve the way OOP concepts teaching

and learning is perceived.

How so? Are the underlying logic and supporting evidence compelling?

Chapter 1 reviewed the research problem from various angles, It outlined students' inability to fully utilise IDEs and further stated with supporting evidence how students fail to interpret the errors messages from their program compilation. This also makes them not to relate the real-life experience with programming. In Chapter 2 the context of existing literature on OOP comprehension was presented. In Chapter 3 different research approaches were also discussed. This led to the choice of the interpretive approach and case study strategy to conduct this study. In Chapter 4 the findings from the data collected were analysed and interpreted. The contribution of this study in Chapter 5 was therefore derived from this solid base of evidence.

How well does the research work reflect seasoned thinking, convey completeness and thoroughness?

The research problem, as well as the results of the case studies, was viewed from different angles using triangulation. This included students' perceptions, expert reviews, student tests and exam results. The various research approaches were discussed in Chapter 3 and the interpretation of the results was undertaken from multiple perspectives in Chapter 4. The last chapter of the thesis is used to review the research, and, in particular, the contributions made by the research study. This indicates thoroughness and reflection on the part of the researcher.

How well is the thesis written? Does it flow logically? Are the central ideas easily accessed?

In Chapter 1 the thesis was introduced, followed by detailed background information to the research problem. Chapter 2 further supported Chapter 1 by examining the current findings in the literature that relates to the teaching and learning of OOP concepts. This chapter led to a research gap and various research variables were identified. Throughout the thesis, the central theme of the thesis was in focus. Research approaches were then discussed in Chapter 3 where a single case study was found more benefiting for this type research. After data was devised from the research approaches used, Chapter 4 gave a detailed interpretation of the data. The

interpreted data provided the contributions given in Chapter 5. The Table of Contents and the Glossary allow for easy access to the central ideas.

Why now? Why is this topic of contemporary interest to scholars and practitioners in this area?

The current throughput of good programmers from universities has been disappointing. It is due to this fact that measures are taken to ensure the industry that is in need of well-groomed programmers capable of solving real world programs is catered for.

Who cares? Who amongst academia would be interested in this topic?

This research is of much interest to students currently learning to program in OOP concepts. Besides the researcher, it also applies to institutions that are willing to restructure and improve on the existing curriculum in learning OOP concepts. This also highly benefits lecturers that are currently teaching programming.

5.4 Direction for future research

Further research can be undertaken to explore more factors that can improve student cognition towards learning OOP concepts. In addition, research might also explore the relationships between various teaching practices in the classroom and how it supports assimilation of OOP concepts by students.

REFERENCES

Ala-Mutka, K., 2012. Problems in Learning and Teaching Programming. Codewitz Needs Anal.

Allert, J., 2004. Learning Style and Factors Contributing to Success in an Introductory Computer Science Course. IEEE, pp. 385–389. doi:10.1109/ICALT.2004.1357442

Alrawashdeh, T.A., Muhairat, M., Althunibat, A., 2013. Evaluating the quality of software in erp systems using the iso 9126 model. Int. J. Ambient Syst. Appl. IJASA 1, 1–9.

Altinay, L., Paraskevas, A., 2008. Planning Research in Hospitality and Tourism. Routledge.

Baxter, P., Jack, S., 2008. Qualitative Case Study Methodology: Study Design and Implementation for Novice Researchers. Qual. Rep. 13, 544–559.

Bennedsen, J., Caspersen, M.E., 2007. Failure Rates in Introductory Programming. ACM SIGCSE Bull. 39, 32–36.

Bodemer, D., Ploetzner, R., Feuerlein, I., Spada, H., 2004. The Active Integration of Information During Learning with Dynamic and Interactive Visualisations. Learn. Instr. 14, 325–341. doi:10.1016/j.learninstruc.2004.06.006

Börstler, J., Bruce, K., Michiels, I., 2003. Sixth Workshop on Pedagogies and Tools for Learning Object Oriented Concepts, in: ECOOP. pp. 84–87.

Bryman, A., 2012. Social research methods, 4th ed. ed. Oxford University Press, Oxford; New York.

Byrne, P., Lyons, G., 2001. The Effect of Student Attributes on Success in Programming, in: ACM SIGCSE Bulletin. ACM, pp. 49–52.

Carlisle, M.C., 2009. Raptor: A Visual Programming Environment for Teaching Object-Oriented Programming. J. Comput. Sci. Coll. 24, 275–281.

Caspersen, M.E., Bennedsen, J., 2007. Instructional Design of a Programming Course: A Learning Theoretic Approach, in: Proceedings of the Third International Workshop on Computing Education Research. ACM, pp. 111–122.

Chege, F.N., Likoye, F., Nyambura, S., Guantai, H.K., 2013. Declining Boys' Participation and Performance in Kenyan Schools: Are Girls' Education Projects Influencing New Forms of Masculinities? CICE 叢書 5 Afr.-Asia Univ. Dialogue Educ. Dev. Final Rep. Phase II Res. Results1 Gend. Equity 1–17.

Cohen, L., Manion, L., Morrison, K., 2007. Research Methods in Education, 6 edition. ed. Routledge, London; New York.

Creswell, J.W., 2012. Educational Research: Planning, Conducting, and Evaluating Quantitative and Qualitative Research, 4th ed. ed. Pearson, Boston.

Creswell, J.W., 2009. Research Design: Qualitative, Quantitative, & Mixed Methods Approaches, 3rd Edition, Inc.,2009. ed. Sag Publications.

Creswell, J.W., 2008. Research Design: Qualitative, Quantitative, and Mixed Methods Approaches, 3rd edition. ed. SAGE Publications, Inc, Thousand Oaks, Calif.

Creswell, J.W., 2007. Qualitative inquiry & research design: choosing among five approaches, 2nd ed. ed. Sage Publications, Thousand Oaks.

DeLine, R., Rowan, K., 2010. Code Canvas: Zooming Towards Better Development Environments, in: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2. ACM, pp. 207–210.

Denny, P., Luxton-Reilly, A., Carpenter, D., 2014. Enhancing Syntax Error Messages Appears Ineffectual. ACM Press, pp. 273–278. doi:10.1145/2591708.2591748

Dominick, R.D.W. and J.R., 2006. Mass Media Research: An Introduction, Eighth Edition edition. ed. Thomson, Beijing.

Eckerdal, A., 2009. Novice Programming Students' Learning of Concepts and Practise.

Eckerdal, A., Thuné, M., Berglund, A., 2005. What Does It Take to Learn'programming Thinking'?, in: Proceedings of the First International Workshop on Computing Education Research. ACM, pp. 135–142.

Field, A., 2005. Discovering Statistics Using SPSS, Second Edition. ed. Sage Publications Ltd, London.

Flick, U., 2007. Managing Quality in Qualitative Research, The sage qualitative research kit. Sage Publications, Thousand Oaks, CA.

Flowers, *P*. (*2009a*). *Research Philosophies Importance* and *Relevance*, *MSC Research Leading*. *Learning* and *Change, Cranfield School* of *Management: Issue 1*

Flowers, *P*. (*2009b*). *Research Philosophies Importance* and *Relevance*, *MSC Research Leading*. *Learning* and *Change, Cranfield School* of *Management: Issue 1*

Gaspar, A., Langevin, S., 2012. An Experience Report on Improving Constructive Alignment in an Introduction to Programming. J. Comput. Sci. Coll. 28, 132–140.

Ghanim, S.A., Al-khafaji, N.J., 2014. Using the Literature to Develop a Preliminary Conceptual Model for the Student Success Factors in a Programming Course: Java as a Case Study.

Gomes, A., Mendes, A.J., 2007a. Learning to Program-Difficulties and Solutions, in: International Conference on Engineering Education–ICEE.

Gomes, A., Mendes, A.J., 2007b. An Environment to Improve Programming Education, in: Proceedings of the 2007 International Conference on Computer Systems and Technologies. ACM, p. 88.

Goosen, L., Pieterse, V., 2005. Students Perceptions of Learning Object-Oriented Programming.

Granell, X., 2014. Multilingual Information Management: Information, Technology and Translators. Chandos Publishing.

Gray, D.E., 2013. Doing Research in the Real World. Sage.

Hartmann, B., MacDougall, D., Brandt, J., Klemmer, S.R., 2010. What Would Other Programmers Do: Suggesting Solutions to Error Messages, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, pp. 1019–1028.

Hertz, M., 2010. What Do CS1 and CS2 Mean?: Investigating Differences in the Early Courses, in: Proceedings of the 41st ACM Technical Symposium on Computer Science Education. ACM, pp. 199–203.

IBM, 2006. Eclipse Platform Technical Overview [WWW Document]. URL https://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html#figure3 (accessed 11.8.14).

Kanellopoulos, Y., Antonellis, P., Antoniou, D., Makris, C., Theodoridis, E., Tjortjis, C., Tsirakis, N., 2010. Code Quality Evaluation Methodology Using the ISO/Iec 9126 Standard. Int. J. Softw. Eng. Appl. 1, 17–36. doi:10.5121/ijsea.2010.1302

Kasurinen, J., Purmonen, M., Nikula, U., 2008. A Study of Visualization in Introductory Programming, in: Proc. 20th Annual Meeting of Psychology of Programming Interest Group, Lancaster, UK.

Kaucic, B., Asic, T., 2011. Improving Introductory Programming with Scratch?, in: MIPRO, 2011 Proceedings of the 34th International Convention. IEEE, pp. 1095–1100.

Kerlinger, F.N., 1986. Foundations of Behavioral Research, 3rd ed. Rinehart and Winston, New York,Holt.

Kinnunen, P., Malmi, L., 2006. Why Students Drop Out Cs1 Course?, in: Proceedings of the Second International Workshop on Computing Education Research. ACM, pp. 97–108.

Kölling, M., Quig, B., Patterson, A., Rosenberg, J., 2003. The Bluej System and Its Pedagogy. Comput. Sci. Educ. 13, 249–268.

Kumar, R., 2005. Research methodology: a step-by-step guide for beginners, 2nd ed. ed. SAGE, London ; Thousand Oaks, Calif.

KydiamS, 2012. Statistics on Visual Learners - College Essay - Kydiams [WWW Document]. StudyMode. URL http://www.studymode.com/essays/Statistics-On-Visual-Learners-1211593.html (accessed 2.2.14).

Lahtinen, E., Ala-Mutka, K., Järvinen, H.-M., 2005. A Study of the Difficulties of Novice Programmers, in: ACM SIGCSE Bulletin. ACM, pp. 14–18.

Lee, M.J., Ko, A.J., 2011. Personifying Programming Tool Feedback Improves Novice Programmers' Learning, in: Proceedings of the Seventh International Workshop on Computing Education Research. ACM, pp. 109–116.

Lister, R., Adams, E.S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J.E., Sanders, K., Seppälä, O., others, 2004. A Multi-National Study of Reading and Tracing Skills in Novice Programmers. ACM SIGCSE Bull. 36, 119–150.

McDowell, C., Werner, L., Bullock, H.E., Fernald, J., 2003. The Impact of Pair Programming on Student Performance, Perception and Persistence, in: Proceedings of the 25th International Conference on Software Engineering. IEEE Computer Society, pp. 602–607.

McLeod, S., 2014. Structured and Unstructured Interviews | Simply Psychology [WWW Document]. URL http://www.simplypsychology.org/interviews.html (accessed 2.11.15).

Mouton, J., 1996. Understanding Social Research, 1st ed., 3rd. impression. ed. Van Schaik Publishers, Pretoria.

Murphy, L., Fitzgerald, S., Hanks, B., McCauley, R., 2010. Pair Debugging: A Transactive Discourse Analysis, in: Proceedings of the Sixth International Workshop on Computing Education Research. ACM, pp. 51–58.

Nienaltowski, M.-H., Pedroni, M., Meyer, B., 2008. Compiler Error Messages: What Can Help Novices?, in: ACM SIGCSE Bulletin. ACM, pp. 168–172.

Oates, B.J., 2005. Researching Information Systems and Computing, 1 edition. ed. SAGE Publications Ltd, London ; Thousand Oaks, Calif.

Olivero, F., Lanza, M., D'Ambros, M., Robbes, R., 2011. Enabling Program Comprehension Through a Visual Object-Focused Development Environment, in: Visual Languages and Human-Centric Computing (vl/Hcc), 2011 Ieee Symposium On. IEEE, pp. 127–134.

ORACLE, 2013. NetBeans IDE - Overview [WWW Document]. URL https://netbeans.org/features/index.html (accessed 11.13.14).

Or-Bach, R., Lavy, I., 2004. Cognitive Activities of Abstraction in Object Orientation: An Empirical Study. ACM SIGCSE Bull. 36, 82–86.

Padayachee, I., Kotze, P., van Der Merwe, A., 2010. Iso 9126 External Systems Quality Characteristics, Sub-Characteristics and Domain Specific Criteria for Evaluating E-Learning Systems. South. Afr. Comput. Lect. Assoc. Univ. Pretoria South Afr.

Parnin, C., Rugaber, S., 2012. Programmer Information Needs After Memory Failure, in: Program Comprehension (Icpc), 2012 Ieee 20th International Conference On. IEEE, pp. 123–132.

Porter, S., 2003. The A-Z of Social Research. SAGE Publications, Ltd, 1 Oliver's Yard, 55 City Road, London England EC1Y 1SP United Kingdom.

Ragonis, N., Ben-Ari, M., 2005. On Understanding the Statics and Dynamics of Object-Oriented Programs, in: ACM SIGCSE Bulletin. ACM, pp. 226–230.

Rogerson, C., Scott, E., 2010. The Fear Factor: How It Affects Students Learning to Program in a Tertiary Environment. J. Inf. Technol. Educ. Res. 9, 147–171.

Rouse, M., 2007. What Is Integrated Development Environment (ide)? - Definition from Whatis.com [WWW Document]. URL http://searchsoftwarequality.techtarget.com/definition/integrated-development-environment (accessed 11.8.14).

Ryan, MA, K., Filene, MPH, J., 2012. Selecting Appropriate Single Case Designs for Evaluating Miechv Funded Home Visiting Programs March 2012.

Salkind, N.J., 2012. 100 questions (and answers) about research methods. SAGE Publications.

Salman, A., 2010. Advantages and Disadvantages of Using IDE. Consult. Knowl. Base.

Samantha Magwashu, 2013. 20 Years into Democracy but Still a Huge Difference Between "model C" and Public Schools. wsusna.

Saunders, M., Lewis, P., Thornhill, A., 2009. Research Methods for Business Students. Prentice Hall, New York.

Seffah, A., Donyaee, M., Kline, R.B., Padda, H.K., 2006. Usability measurement and metrics: A consolidated model. Softw. Qual. J. 14, 159–178. doi:10.1007/s11219-006-7600-8

Segedy, J.R., Loretz, K.M., Biswas, G., 2013. Model-Driven Assessment of Learners in Open-Ended Learning Environments, in: Proceedings of the Third International Conference on Learning Analytics and Knowledge. ACM, pp. 200–204.

Settle, A., Vihavainen, A., Sorva, J., 2014. Three Views on Motivation and Programming. ACM Press, pp. 321–322. doi:10.1145/2591708.2591709

Sheard, J., Carbone, A., Chinn, D., Clear, T., Corney, M., D'Souza, D., Fenwick, J., Harland, J., Laakso, M.-J., Teague, D., others, 2013. How Difficult Are Exams?: A Framework for Assessing the Complexity of Introductory Programming Exams, in: Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136. Australian Computer Society, Inc., pp. 145–154.

Sicilia, M.-Á., 2006. Strategies for Teaching Object-Oriented Concepts with Java. Comput. Sci. Educ. 16, 1–18.

Simons, H., 2009. Case Study Research in Practice. SAGE Publications Ltd, Los Angeles ; London.

Sirkiä, T., Sorva, J., 2012. Exploring Programming Misconceptions: An Analysis of Student Mistakes in Visual Program Simulation Exercises, in: Proceedings of the 12th Koli Calling International Conference on Computing Education Research. ACM, pp. 19–28.

Sorva, J., 2013. Notional Machines and Introductory Programming Education. ACM Trans. Comput. Educ. 13, 1–31. doi:10.1145/2483710.2483713

Stamouli, I., Huggard, M., 2006. Object Oriented Programming and Program Correctness: The Students' Perspective, in: Proceedings of the Second International Workshop on Computing Education Research. ACM, pp. 109–118.

Storey, M.-A., 2005. Theories, Methods and Tools in Program Comprehension: Past, Present and Future, in: Program Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop On. IEEE, pp. 181–191.

Tan, P.-H., Ting, C.-Y., Ling, S.-W., 2009a. Learning Difficulties in Programming Courses: Undergraduates' Perspective and Perception. IEEE, pp. 42–46. doi:10.1109/ICCTD.2009.188

Tan, P.-H., Ting, C.-Y., Ling, S.-W., 2009b. Learning Difficulties in Programming Courses: Undergraduates' Perspective and Perception. IEEE, pp. 42–46. doi:10.1109/ICCTD.2009.188

Techopedia, 2014. What Is an Integrated Development Environment (ide)? - Definition from Techopedia [WWW Document]. Techopedia.com. URL http://www.techopedia.com/definition/26860/integrated-development-environment-ide (accessed 3.25.14).

Thyer, B.A., 1993. The Handbook of Social Work Research Methods. SAGE Publications, Inc., 2455 Teller Road, Thousand Oaks California 91320 United States of America.

Van Haaster, K., Hagan, D., 2004. Teaching and Learning with Bluej: An Evaluation of a Pedagogical Tool, in: Information Science+ Information Technology Education Joint Conference, Rockhampton, QLD, Australia.

Vihavainen, A., Paksula, M., Luukkainen, M., 2011. Extreme Apprenticeship Method in Teaching Programming for Beginners. Association for Computing Machinery, New York, N.Y.

Wahid, A., Mustafa, K., Khan, R.A., 2008. Wbis for Computer Programming., in: CSREA EEE. pp. 413–419.

Walsham, G., 2006. Doing Interpretive Research. Eur. J. Inf. Syst. 15, 320–330. doi:10.1057/palgrave.ejis.3000589

Watson, C., Li, F.W.B., 2014. Failure Rates in Introductory Programming Revisited. ACM Press, pp. 39–44. doi:10.1145/2591708.2591749

Whalley, J.L., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, P.K., Prasad, C., 2006. An Australasian Study of Reading and Comprehension Skills in Novice Programmers, Using the Bloom and Solo Taxonomies, in: Proceedings of the 8th Australasian Conference on Computing Education-Volume 52. Australian Computer Society, Inc., pp. 243–252.

Whetten, D.A., 1989. What Constitutes a Theoretical Contribution? Acad. Manage. Rev. 14, 490–495.

Xinogalos, S., 2012. Programming Techniques and Environments in a Technology Management Department, in: Proceedings of the Fifth Balkan Conference in Informatics. ACM, pp. 136–141.

Xinogalos, S., 2009. Guidelines for Designing and Teaching an Effective Object-Oriented Design and Programming Course, in: Hijon-Neira, R. (Ed.), Advanced Learning. InTech.

Xinox Software, 2010. JCreator — Java Ide [WWW Document]. URL http://www.jcreator.com/about.htm#a6 (accessed 11.13.14).

Zeiss, B., Vega, D., Schieferdecker, I., Neukirchen, H., Grabowski, J., 2007. Applying the ISO 9126 Quality Model to Test Specifications. Softw. Eng. 231–242.

APPENDICES

APPENDIX A: Questionnaire NetBeans IDE

Specific questions. Section A (participant's profile)

1. Name…….. (Optional)

2. To which age category do you belong to?

   ☐ Less than 18
   ☐ 18 - 21
   ☐ 22 - 25
   ☐ 26 - 30
   ☐ 31 -35
   ☐ 35 +

3. Gender:

   ☐ Male
   ☐ Female

4. How long have you been programming?

   ☐ None at all
   ☐ Less than 1year
   ☐ More than 1 year

5. What programming languages have you used before? _____

   _____

   _____

NetBeans

Section A: Specific questions to be completed during and/or after software use

1. In this section, answer to your satisfaction of the IDE indicated.
    a) With respect to the version of NetBeans currently installed on your machine, please indicate the extent to which you agree or disagree with the following statements:

1=Strongly Disagree 2=Disagree 3=Neutral 4=Agree 5= Strongly Agree					
This software is easy to use.	○ 1	○ 2	○ 3	○ 4	○ 5
I am in control of the contents of the menus and toolbars.	○ 1	○ 2	○ 3	○ 4	○ 5
I will be able to learn how to use all that is offered in this software.	○ 1	○ 2	○ 3	○ 4	○ 5
Navigating through the menus and toolbars is easy to do.	○ 1	○ 2	○ 3	○ 4	○ 5
This software is engaging.	○ 1	○ 2	○ 3	○ 4	○ 5
The contents of the menus and the toolbars match my needs.	○ 1	○ 2	○ 3	○ 4	○ 5
Getting started with this version of the software is easy.	○ 1	○ 2	○ 3	○ 4	○ 5
Finding the options that I want in the menus and toolbars is easy.	○ 1	○ 2	○ 3	○ 4	○ 5
The software responds in time.	○ 1	○ 2	○ 3	○ 4	○ 5
Discovering new features is easy.	○ 1	○ 2	○ 3	○ 4	○ 5
This software is satisfying to use.	○ 1	○ 2	○ 3	○ 4	○ 5

Section B:Please rate the following statements

Statements	Very easy (1)	Easy (2)	Moderate (3)	Somewhat hard (4)	Difficult (5)	Very difficult (6)
How easy is it to use NetBeans in completing your tasks?						
How easy is it to use NetBeans in understanding the following object-oriented concepts?						
Class						
Object						
Method Overloading						
Method Overriding						
Inheritance						
Encapsulation						
Polymorphism						
How easy is it to recover from the following in NetBeans?						
System (IDE) error messages						
Output(animation) is not the movement you expected						

Statements	Not at all (1)	Somewhat (2)	Moderately (3)	Quite a bit (4)	Very much (5)
How much has NetBeans increased your confidence in learning the following Object-Oriented Concepts?					
Class					
Object					
Method Overloading					
Method Overriding					
Inheritance					
Encapsulation					
Polymorphism					

Section C:Please complete the following sentences.

1. Can you relate between a Class and an Object in NetBeans (Yes/No)? ……..

2. What is the most difficult task to accomplish in NetBeans?…….

3. Comments

APPENDIX B: Questionnaire JCreator IDE

Specific questions. Section A (participant's profile)

1. Name…….. (Optional)

2. To which age category do you belong to?

   ☐ Less than 18
   ☐ 18 - 21
   ☐ 22 - 25
   ☐ 26 - 30
   ☐ 31 -35
   ☐ 35 +

3. Gender:

   ☐ Male
   ☐ Female

4. How long have you been programming?

   ☐ None at all
   ☐ Less than 1year
   ☐ More than 1 year

5. What programming languages have you used before? _____

   _____

   _____

JCreator LE 5.0

Section A: Specific questions to be completed during and/or after software use

1. In this section, answer to your satisfaction of the IDE indicated.
    b) With respect to the version of JCreator LE currently installed on your machine, please indicate the extent to which you agree or disagree with the following statements:

1=Strongly Disagree 2=Disagree 3=Neutral 4=Agree 5= Strongly Agree					
This software is easy to use.	○ 1	○ 2	○ 3	○ 4	○ 5
I am in control of the contents of the menus and toolbars.	○ 1	○ 2	○ 3	○ 4	○ 5
I will be able to learn how to use all that is offered in this software.	○ 1	○ 2	○ 3	○ 4	○ 5
Navigating through the menus and toolbars is easy to do.	○ 1	○ 2	○ 3	○ 4	○ 5
This software is engaging.	○ 1	○ 2	○ 3	○ 4	○ 5
The contents of the menus and the toolbars match my needs.	○ 1	○ 2	○ 3	○ 4	○ 5
Getting started with this version of the software is easy.	○ 1	○ 2	○ 3	○ 4	○ 5
Finding the options that I want in the menus and toolbars is easy.	○ 1	○ 2	○ 3	○ 4	○ 5
The software responds in time.	○ 1	○ 2	○ 3	○ 4	○ 5
Discovering new features is easy.	○ 1	○ 2	○ 3	○ 4	○ 5
This software is satisfying to use.	○ 1	○ 2	○ 3	○ 4	○ 5

Section B: Please rate the following statements

Statements	Very easy (1)	Easy (2)	Moderate (3)	Somewhat hard (4)	Difficult (5)	Very difficult (6)
How easy is it to use JCreator in completing your tasks?						
How easy is it to use JCreator in understanding the following object-oriented concepts?						
Class						
Object						
Method Overloading						
Method Overriding						
Inheritance						
Encapsulation						
Polymorphism						
How easy is it to recover from the following in JCreator?						
System (IDE) error messages						
Output(animation) is not the movement you expected						

Statements	Not at all (1)	Somewhat (2)	Moderately (3)	Quite a bit (4)	Very much (5)
How much has JCreator increased your confidence in learning the following Object-Oriented Concepts?					
Class					
Object					
Method Overloading					
Method Overriding					
Inheritance					
Encapsulation					
Polymorphism					

Section C:Please complete the following sentences.

1. Can you relate between a class and an Object in JCreator (Yes/No)? ……..

2. What is the most difficult task to accomplish in JCreator?…….

3. Comments

APPENDIX C: Interview schedule student

Participant's profile

Specific questions. Section A (participant's profile)

1. Name…….. (Optional)

2. To which age category do you belong to?

   ☐ Less than 18
   ☐ 18 - 21
   ☐ 22 - 25
   ☐ 26 - 30
   ☐ 31 -35
   ☐ 35 +

3. How long have you been lecturing Java programming?

   ☐ Less than 1year
   ☐ For a Year
   ☐ More than 2 year

4. Gender:

   ☐ Male
   ☐ Female

Questions to be answered by the interviewee

1. Do you think learning OOP is difficult?
2. From your own observations, are you motivated to learn OOP using Java programming language?

3. Which mistakes do you make normally when using the IDEs?

4. Do you find it difficult to learn and use the IDEs?

5. Is there any other misconception that you encounter when learning OOP?

APPENDIX D: Interview schedule expert

Participant's profile

Specific questions. Section A (participant's profile)

1. Name…….. (Optional)

2. To which age category do you belong to?

   ☐ Less than 18
   ☐ 18 - 21
   ☐ 22 - 25
   ☐ 26 - 30
   ☐ 31 -35
   ☐ 35 +

3. How long have you been lecturing Java programming?

   ☐ Less than 1year
   ☐ For a Year
   ☐ More than 2 year

4. Gender:

   ☐ Male
   ☐ Female

Questions to be answered by the interviewee

1. Do you think learning OOP is difficult?

2. From your own observations, are students motivated to learn OOP using Java programming language?

3. Which mistakes do students make normally when using the IDEs?

4. Do students find it difficult to learn and use the IDEs?

5. Is there any other misconception that students encounter when learning OOP?