USING AN E-LEARNING TOOL TO OVERCOME DIFFICULTIES IN LEARNING OBJECT-ORIENTED PROGRAMMING

by

Saadia Fahim Essa

submitted in accordance with the requirements for
the degree of

MASTER OF SCIENCE

in the subject

COMPUTING

at the

University of South Africa

Supervisor: Prof. Ian Sanders

March 2016

DECLARATION BY STUDENT

Student number: 30965799

I declare that Using an e-Learning Tool To Overcome Difficulties In Learning Object-Oriented Programming is my own original work and that all the sources that I have used or quoted have been indicated and acknowledged by means of complete references.

March 2016

SAADIA ESSA

DATE SUBMITTED

ACKNOWLEDGEMENTS

Above all, thank you to my Creator, Cherisher and Sustainer, the Almighty Allah (God), Lord of the worlds, Master of the Day of Judgement. Thee alone do we worship and Thee alone do we ask for help. Without Your guidance, this study would certainly have not been possible. “Glory be to Allah (God), the Exalted.”
O our Lord, accept (this service) from us. Verily You and You alone are the Hearer, the Knower (Holy Quraan, 1:127).

I would like to express my gratitude to the following people:

- My supervisor, Prof. I. Sanders, as he has been a constant source of inspiration to me during this challenging and onerous period. His continued encouragement, motivation and academic contribution contributed to the completion of my dissertation. May God bless him.
- Lecturer Ronell van der Merwe, for her support and assistance in setting up an online questionnaire, reviewing the questionnaire and encouraging students’ participation.
- I am grateful to officials of UNISA who granted me permission to conduct this study and administer the questionnaire on the students.
- To the students, for responding to the questionnaires.
- To my parents, for their support, encouragement and prayers.
- To my husband, for his moral support and continued encouragement throughout this journey. His sacrifice and motivation were highly appreciated.
- To my children, who were a source of strength during the time when we burnt the midnight oil.
- To my editor, Khomotso Bopape of Let’s Edit (Pty) Ltd, for assisting with the editing of this dissertation.

ABSTRACT

This study was motivated by the need to overcome the pedagogical hindrances experienced by introductory object-oriented programming students in order to address the high attrition rate evident among novice programmers in distance education.

The initial phase of the research process involved exploring a variety of alternative visual programming environments for novices. Thereafter the selection process detailed several requirements that would define the ideal choice of the most appropriate tool. An educational tool Raptor was selected. Lastly, the core focus of this mixed method research was to evaluate undergraduate UNISA students' perceptions of the Raptor e-learning tools with respect to the perceived effectiveness in enhancing novices' learning experience, in an attempt to lower the barriers to object-oriented programming.

Students' perceptions collectively of the Raptor visual tool were positive and despite the fact that the sample size was too small to achieve statistical significance, these quantitative and qualitative results provide the practical basis for implementing Raptor in future. Thus providing learning opportunities suited to learner interests and needs, can lead to an enormous potential to stimulate individuals' motivation and development in creating a more positive learning experience to overcome barriers in programming and enhance concept understanding to address the diverse needs of students in distance education that could lead to a reduced dropout rate.

Key terms:
E-learning; Object-Oriented Programming; Distance Education; Mixed Methods; Visualisation Tools; Algorithm Visualisation Tools; Raptor; Novice Programmers; Web-based Learning; Computer Science

TABLE OF CONTENTS

CHAPTER 3
FOCUS

CHAPTER 4
TOOLS PLATFORM

CHAPTER 5
RESEARCH DESIGN AND METHODOLOGY

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

3D	Three-dimensional
AV	Algorithm Visualisation
CIMEL	Constructive, collaborative, Inquiry-based Multimedia E-Learning
CMU	Carnegie Mellon University
CS	Computer Science
DE	Distance Education
GUI	Graphical User Interface
I/O	Input or Output
IDE	Integrated Development Environment
ITS	Intelligent Tutoring System
LMS	Learning Management System
OLE	Online Learning Environment
OO	Object-Oriented
OOP	Object-Oriented Programming
PV	Program Visualisation
RAPTOR	Rapid Algorithmic Prototyping Tool for Ordered Reasoning
SCS	Selection control statement
SFC	Structured Flow Chart
UNISA	University of South Africa

CHAPTER 1
ORIENTATION TO THE STUDY

1.1 INTRODUCTION

In today's rapidly changing world, information technology society roles of teaching and learning, the knowledge of media and the perceptions of technology and tools used in education are transforming (Duan & Jiang, 2008). The key to gaining an educationally competitive advantage is '*knowledge and the way it is deployed by an organization'* (Ivanov & Peneva, 2007:IV.8-1). Abdelhakim and Shirmohammadi (2007) asserted that many universities are adopting innovative educational technologies to broaden their markets and enhance flexibility in their offerings.

Distributed education is convenient with respect to accessibility and is also characterised by its flexibility in terms of time, location and pace of learning (Weissman, 2002). The University of South Africa (UNISA) is a distance education institution which aims to address Africa's education and developmental problems (UNISA, 2012). UNISA is an eminent, comprehensive, flexible (time and place) and an accessible open distance learning institution (UNISA, 2012). This higher-level organisation offers internationally recognised qualifications and has optimal resources (UNISA, 2012).

The persistent and escalating need for novel approaches in distributing education has fuelled remarkable transformations in learning technology and institutions (Zhang, Zhao, Zhou, & Nunamaker, 2004). As a result, this has led to a transition – to a new era in distance education. The introduction of the Internet and World Wide Web technologies has made *e-learning* or web-based learning possible (Li, Lau, Shih & Li, 2008).

Due to these technical advances, progressive ways of thinking as well as the demand for learning, this has resulted in a critical pressure point for education institutions and organisations in understanding the e-learning phenomena and making tactical decisions on how to implement e-learning techniques within their environments (Zhang et al., 2004). E-learning is most significant with respect to its role in distance education and as a supplementary learning tool used predominantly in the higher education arena (Duan and Jiang, 2008). The emphasis on e-learning

tools to supplement training at many universities and companies unlocks new approaches to personalise the learning experience (Wolf, 2012). Vrasidas (2004) maintained that e-learning and distributed education is characterised by tremendous growth.

As a result of the increase in accessibility of fast, multimedia-capable computers and the rapid uptake of broadband internet, this has led to new opportunities to deliver "rich media content" via the Internet (Wolf, 2012). Web-based learning applications have grown with the wide use of the Internet. Thus, digital learning content in education and distributed education courses such as e-learning platforms are experiencing rapid developments. Due to accessibility, usability and technology influences, one can enhance the learning experience.

Torrente, Del Blanco, A., Moreno-Ger, Martinez-Ortiz and Fernandez-Manjon (2009a), explained that e-learning comprises of rich content that includes multimedia and interactive experiences that engage students, thus resulting in increased motivation to learn. This can be achieved with the support of multimedia technology, which encompasses information visualisation tools and computer graphics; this results in engaging individuals, hence leading to an enhanced learning experience. Visualisation tools create a positive educational experience, enabling students to visually express and construct meaning (Vrasidas, 2004). The emergence of visual-based instructional technology provides an enriched form of teaching-learning environment and can bring about powerful changes to the education system of e-learning in distance education. Thus the Internet and visualisation tools are reforming the way knowledge is deployed.

Major changes in development and disposition of information technologies force many transformations in computer science curricula for all educational degrees (Ivanov & Peneva, 2007). Due to the dynamics in computing, students need to be stimulated with additional resources. These additional specialisations could be accomplished by self-directed and distance learning (Ivanov & Peneva, 2007).

Programming is one of the fundamental courses required in any computer science curriculum, but for most students, it is also one of the most challenging. Dillon, Anderson & Brown (2012) highlighted that in order to improve the novice experience with learning to program, visual environments can be implemented. Both university and industry have placed increasing importance on the early exposure of students to

object-oriented programming (OOP) (Georgantaki & Retalis, 2007). There is a demand for software engineers who are proficient in using the OOP paradigm to analyse and develop systems (Georgantaki & Retalis, 2007).

There are varied dynamics associated with the difficulty experienced by novice programmers. Also, there are various programming languages, which require a substantial amount of time to learn. Carlisle, Wilson, Humphries and Hadfield (2005) highlighted the concern that the use of a particular programming language in an introductory computing course focuses on syntactic difficulties encountered by novice programmers instead of focusing on the fundamental issues of object-oriented programming concepts of classes and algorithmic problem-solving. Furthermore, in the learning process, novices face challenges in comprehending the fundamentals of object-oriented design and programming concepts (Moritz, Wei, Parvez & Blank, 2005). Teaching computer programming to novices is difficult, as students find programming overwhelming when they attempt to understand abstract and fundamental complex concepts (Dorn & Sanders, 2003). Gomes, Santos and José Mendes (2012) further noted that difficulty in learning introductory programming leads to dropouts or poor performance. Stephen, Franklin, Patrick, Peter and Elizabeth (2012) also agree and recognised pedagogical approaches used in the instruction process as an underlying cause of the poor performance and misconceptions in programming. Gomes et al. (2012) attributed the following influences as a contributing cause towards the poor performance and comprehension in programming: contextual knowledge and attitudes, pedagogical approaches used in the instruction process and social circumstance.

According to Truong, Bancroft and Roe (2005), difficulties experienced in both learning and teaching OOP lead to a lack of confidence and a reduced perception of programming. Furthermore, students are inclined to lose self-confidence, particularly due to the abstract nature of concepts; also, it is difficult to visualise or understand concepts (Dorn & Sanders, 2003).

Moreover, in the learning process, novices face challenges in Distance Education (DE) introductory programming. They face challenges not only in understanding fundamental concepts but also in addressing the needs of students of diverse backgrounds. Esteves, Fonseca, Morgado and Martins (2009) noted that the diversity of experience levels and backgrounds in distance education students' knowledge results in the complex traditional teaching process. Further, due to lack of

personalised attention in DE, this ultimately leads to an inadequate learning experience and increased likelihood of failure (Truong et al., 2005). This certainly is not an optimal way to promote students' self-efficacy, confidence and create an enhanced perception in novice's ability to learn programming. Clearly, programming is complex for students; this is particularly evident from the high attrition rates in introductory programming and it remains one of the most urgent challenges (Hundhausen, Narayanan & Crosby, 2007).

These learning difficulties cannot be fully solved by instructors; thus, new procedures and tools are needed to assist students with diverse learning styles in mastering the fundamental object-oriented software development concepts (Moritz et al., 2005; Wei, Moritz, Parvez & Blank, 2005). Current technological advances have unlocked new learning opportunities (Esteves et al., 2009). Several authors (Georgantaki & Retalis, 2007; Matthews, Hin, & Choo, 2009) have noted that various researchers have conducted studies on the effective instructional processes for Object-Oriented (OO) design and programming. Those researchers focused on the utilisation of educational tools in helping students to mitigate learning barriers in conceptualising the OOP philosophy and its concepts (Georgantaki & Retalis, 2007). Programming and visualisation environments have been explored, enabling users to develop, visualise and interact with concrete visual representations of computing processes in order to dissociate computing concepts from programming languages (Hundhausen et al., 2007).

Furthermore, Minton, Boyle and Dimitrova (2004) emphasised that, the primary reason behind the use of e-learning technology is diversity, as e-learning is motivated by learner needs. To address the diverse needs of learners, interactive, personalised and adaptive technology is required (Minton et al., 2004). The rise and establishment of e-learning technology has led to the development of modern comprehensive e-learning systems that are targeted at alleviating the prevailing issues (Torrente, Moreno-Ger, Martínez-Ortiz & Fernandez-Manjon, 2009b). Some of these issues include lack of motivation and the high dropout rates as a result of the separation between students and instructors, which is characteristic of distance education. Piccolli, Ahmad and Ives (2001) also emphasised that the high flexibility of e-learning with respect to time, place and pace is a positive motivational factor. Thus, this provides learning opportunities suited to learner interests and needs. Additionally, it can lead to an enormous potential of stimulating individuals' motivation and development in creating a more positive learning experience in order to

overcome barriers in programming and enhance concept understanding to address the diverse needs of students in DE. There has been a positive impact of these tools that have been implemented in various universities and other institutions. This will be elaborated on in Chapter 2.

This study was motivated by the need to overcome the pedagogical hindrances experienced by introductory OOP students and to address the high attrition rate evident among novice programmers. The study aims to provide the necessary tool support to effectively support knowledge construction and learning in distributed education instructional processes in order to ease the difficulty of OOP design and programming among novices. The approach that was utilised involved the use of an e-learning tool to ease programming difficulty with the aim of supporting instruction of OOP design and programming. This can lead to the creation of content rich applications and an enhanced learning experience that are useful for computer science programming students at UNISA. Further, it is the aim of this study to enhance understanding and to create an enhanced perception in novice's ability to learn programming, ensuring they master the fundamental concepts of OOP software development.

This research entailed exploring a variety of alternative visual programming environments for novices in an attempt to lower the barriers to OOP programming. Therafter, the study involved selecting an appropriate tool to ease novices' understanding of OOP fundamental concepts. The selection process detailed several requirements that would define the ideal choice of the most appropriate tool. The primary goal of the selected tool is to ease OOP difficulties experienced by novices through improving student problem solving skills and minimising syntactic complexity. The key feature is ensuring the tool is well suited to distance education novice programmers. Primarily, this study addresses students' perceptions of whether the selected computing technology supplemented with distance education can stimulate an individual's motivation and development. This study will achieve this goal with the support of multimedia technology, which includes information visualisation tools and computer graphics in engaging individuals to enhance their learning experience in order to ease OOP difficulties experienced by novices. Creating an engaging, e-learning programming experience could result in an increase in students' learning confidence and thus could result in a reduced dropout rate.

1.2 PROBLEM STATEMENT

In this era that is thriving with various technology-mediated forms of learning, there is a need not only for more effective online instructional methods but also the need to use technology to enhance learning acquisition and facilitate teaching effectiveness (Wang, 2011). Brinkman, Payne, Patel, Griggin and Underwood (2007) suggested that the success of educational technologies is dependent on ease of use and the ability to motivate and engage students and adapt to their needs. With the current technology focusing on both academic and professional developments with regard to object-oriented design and programming, it is acknowledged that object-oriented design and programming are still challenging for students. The question that will be addressed is: Will the approach of using technology-enhanced e-learning tools improve the student experience by assisting to circumvent pedagogical hindrances experienced by DE students in introductory OOP?

The scope of this study involves three parts, as illustrated in Figure 1.1. First, it explores and presents a variety of alternative appropriate novice programming e-learning tools. This phase entails research conducted on what forms of technology-enhanced e-learning tools can be used to supplement the instructional process of OO design and programming. Further, the research investigates the impact of e-learning tools in assisting to create content-rich, engaging and easily accessible applications that are suitable for addressing diverse issues that impede novice programmers' understanding and to master the fundamental concepts of object-oriented software development. This sub-question is addressed in Chapter 2.

The second phase addresses the selection process, through detailing criteria for the selection process. Thereafter, a detailed explanation of why the tool was selected is given followed by an explanation of what the Raptor tool is about. An educational tool Raptor that is specifically developed to lower novice hindrances when learning to program was selected and presented. The Raptor tool supports instruction of the OOP programming paradigm through limiting syntax and offering concrete visual representations. Further, it provides a personalised learning approach and requires greater learner autonomy, thus creating an enhanced learning experience and resulting in a positive attitude towards learning to program (Esteves et al., 2009). This phase is addressed in Chapter 4.

Lastly, the core focus of this research is to gain an understanding of novices' perceptions, through gathering experimental empirical data on the subjective views of novices' experience and effectiveness of learning to program with asynchronous and web-based visual learning tool Raptor. This phase investigates the impact on students' perception of the process of integrating the *selected* technology-enhanced tool approach in distance education. It is hoped that student perceptions regarding the impact of a visual e-learning tool support environment will be determined to overcome difficulties in learning OOP in distance education. This will be addressed in Chapters 5 and 6. Further, this phase entails two aspects:

a) Will the selected visually engaging e-learning tool approach that illustrates solutions to algorithm design problems assist novices by motivating them through creating a more positive, enhanced user learning experience with technology? Further, this study seeks to determine novices' perceptions of Raptor with respect to its usability, engaging user experience and learning capability of the tool to support efficient and effective learning in understanding fundamental OOP concepts.

b) To what extent does the student believe the tool environment has contributed to overcoming difficulties in learning OOP?

A great deal of research shares the commitment of this study, in rigorously evaluating the educational benefits of learning tools involving visualisation technology. The focus in the study is on the use of interactive, personalised and adaptive visualisation e-learning tools in stimulating individuals' motivation to reduce complexity experienced by novices. A key element offering opportunities for teaching and learning in order to reinforce and ease the difficulty of OOP among novices is to offer a more personalised and engaging learning approach, thus resulting in an enhanced positive learner experience. The contribution of this dissertation is on the emphasis of an educational methodology based on the recognition of supplementing distance education prescribed works with a technology-mediated tool. The purpose of this is to reinforce and teach fundamental concepts of OOP so as to address barriers novices face when learning to program. This will enable students to augment existing learning resources, to bridge knowledge gaps or to get more suitable explanations.

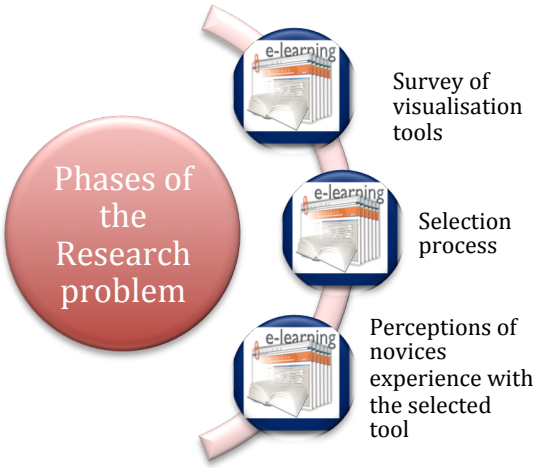


Figure 1.1: Phases of research

1.3. RESEARCH QUESTIONS AND ASSOCIATED SUB-QUESTIONS

1.3.1 MAIN RESEARCH QUESTION

Will the approach of using technology-enhanced e-learning tools improve the student experience by assisting to circumvent pedagogical hindrances experienced by DE students in introductory OOP?

1.3.2 SUB-QUESTIONS

In addition to the main research question, the following sub-questions are worth bringing to the fore:

1. What is an appropriate e-learning tool for teaching OOP?

2. What criteria are involved in the selection process to define an appropriate tool?

3. What are students' perceptions of the tool?

A detailed discussion of the research questions and associated sub-questions of the study on the impact that e-learning tool support has in easing the difficulty of OOP among novices is addressed in chapter 3.

1.4 SCOPE OF THE STUDY

This study was delimited to the semester module Interactive Programming (ICT2612) as well students studying Web Development (ICT1513) in the undergraduate Diploma in Information Technology in 2014 and Interactive Programming (ICT2612) in 2015 at the tertiary DE institution of UNISA. The total population of respondents for the study was 985 students over the two years. Students are of diverse backgrounds and cultures and from various areas that are not limited to South Africa.

1.5 LIMITATIONS OF THE STUDY

In this study, distributed education and distance education were used synonymously, since both refer to a self-directed learning process. Quality through e-learning in this study refers to using e-learning as an approach to improve educational opportunities, access and learning (Mehdi & Feiznia, 2011).

This study still constitutes a mixed methods study even though it does not involve the rich context of open-ended question analysis and detailed information from participants, as it collects both quantitative and qualitative data.

A limitation of this study was a poor response rate, contributing to a small sample size, resulted in low statistical power. Consequently, the scope for generalisation was limited, as a small sample was used. Another limitation is the placebo effect, as students might report changes because they are expected to and not because of their actual experience (Mouton, 2001).

The study was limited to UNISA therefore generalisations were only restricted to this distance learning institution. Another limitation of the present study was that it was not possible to conduct further surveys on the same courses in order to progressively build more significant, quantitative and qualitative measures over an increased population size due to time constraints. Furthermore due to the paucity of literature on difficulties of OOP of students in UNISA, the findings may not fully explain the impact of an e-learning tool to overcome the difficulties in learning OOP of the participants in the study.

1.5 STRUCTURE AND OUTLINE OF DISSERTATION

Table 1.1: Structure of dissertation

Chapter 1: Orientation to the Study	Chapter 1 provides background to the study, including a discussion of the research problem regarding the impact of e-learning tool support in addressing pedagogical hindrances experienced by distance education novice OOP students. This is followed by the research questions and associated sub-questions; scope and limitations of the study. Thereafter the remaining chapters in the dissertation are outlined.
Chapter 2: Literature Review	This chapter focuses on the initial phase that involves investigating and finding various tools to ease novice programmers' understanding of fundamental OOP concepts. This chapter provides the background and motivation for conducting the research in addressing the importance of the impact of e-learning tools in order to supplement distance education for novice programmers and to address difficulties of OOP at UNISA.
Chapter 3: Focus	This chapter focuses on the research objective, significance of the study and detailed discussion of the research questions and associated sub-questions of the study on the impact that e-learning tool support has in easing the difficulty of OOP among novices.
Chapter 4: Tools Platform	Chapter 4 specifies the methodology for identifying an appropriate tool. This chapter focuses on the selected visualisation tools' platform in creating both a positive programming learning experience and assisting to lower a distance education novice programmer's hindrances when learning to program.
Chapter 5: Research Design and Methodology	The final phase of the research entails evaluating the subjective view of students with regard to the selected visual algorithmic e-learning tool selected. The procedure of how the data is presented and interpreted is discussed in this chapter. The research design and methodology for evaluating the tool are also discussed.

	Aspects related to mixed methods are outlined and will include a framework on analysis. This chapter focuses on the philosophical assumptions, research design and methodology approach undertaken by the research study so that valid data could be collected and analysed to create meaning.
Chapter 6: Results and Discussion	The final phase of the research entails evaluating the subjective view of students with respect to the selected visual algorithmic e-learning tool. The final phase discussing the results of survey and data analysis and interpretation are covered in this chapter. This chapter presents the data collected and the analysis. The processes include quantitative and qualitative frameworks within them. Quantitative and qualitative data was gathered from concurrent questionnaires, which were administered to the distance education computer science students.
Chapter 7: Conclusion and Recommendations	Chapter 7 focuses on the summary of findings; conclusions are drawn and recommendations are made on issues raised.
List of References	
Appendices	

1.6 CONCLUSION

Chapter 1 provided background to the study, including a discussion of the research problem regarding the impact of e-learning tool support to address pedagogical hindrances experienced by distance education novice OOP students. This is followed by an outline of the dissertation.

The next chapter focuses on a literature review. It provides the background and motivation for conducting the research in addressing the importance of the impact of e-learning tools to supplement distance education for novice programmers in addressing difficulties of OOP at UNISA.

CHAPTER 2
LITERATURE REVIEW

2.1 INTRODUCTION

This chapter focuses on literature relevant for this study. This chapter provides the background and motivation for conducting the research in addressing the importance of the impact of e-learning tools to supplement distance education for novice programmers in an effort to address difficulties of OOP at UNISA. This chapter focuses on the initial phase that involves investigating and finding various tools to ease the difficulty of OOP design and programming among novices.

According to Bijlani, Jayahari and Mathew (2011), the advent of novel technologies has created a paradigm shift in the education system. The way people are learning is evolving due to the rapid advancement of computer technology, potentially unlocking new learning opportunities (Esteves et al., 2009). Web-based learning applications have grown with wide use of the Internet. Thus, computer technology-mediated learning and distributed education programmes such as e-learning platforms are experiencing rapid developments.

Major changes in development and disposition of information technologies force many transformations on computer science curricula for all educational degrees (Ivanov & Peneva, 2007). Due to the dynamics in computing, students need to be stimulated with additional resources; these additional specialisations could be accomplished by self-directed and distance learning (Ivanov & Peneva, 2007). Novices experience difficulty learning and understanding object-oriented design and programming principles and fundamental concepts. These learning difficulties cannot be fully solved by instructors; thus, new procedures and tools can be used to assist diverse students to understand the fundamental object-oriented programming (OOP) concepts (Moritz et al., 2005; Wei et al., 2005).

DE students are characterised by diversity in knowledge, thus resulting in a complex learning process. The focus in this study is on the utilisation of educational tools to assist students to alleviate the learning barriers in understanding the OOP philosophy and its concepts. It is hoped that a more personalised, motivating and enhanced learner experience will be achieved in order to effectively support

knowledge construction and learning in DE to ease the difficulty of OOP design and programming among novices.

The remainder of the literature review is organised as follows:

- Distance education and e-learning characteristics
- Role of distance education and e-learning in enhancing learning
- Motivation to do object-oriented programming
- Difficulties experienced in teaching and learning programming
- Survey of visualisation tools relating to OOP pedagogy to address difficulties in OOP
- Why visualisation tools
- Role of visualisation in tool selection

2.2 DISTANCE EDUCATION AND E-LEARNING CHARACTERISTICS

To meet the needs of people, tertiary institutions and administrations in developing countries have implemented programmes in various forms for students, adults and businesses (Yukselturk, Yazici, Sacan & Kaya, 2010). Distributed education can offer these programmes, as it is convenient with respect to accessibility, as well as characterised by its flexibility in terms of convenience, location and pace of learning Weissman (2002). Wang (2011) highlighted that distributed education is thriving at an phenomenal rate, fuelled by accelerating advances in the remarkable ability to break through time and space constraints. According to Weissman (2002), the aim of distributed education was to extend the traditional universities to overcome their inherent problems of inadequacy and exclusivity.

The persistent and escalating need for novel approaches to distributing education has fuelled remarkable transformations in learning technology and institutions (Zhang et al., 2004). Zhang et al. (2004) stated that, the progression of digital and networking technologies is offering various methods to individualise and enhance flexibility, portability and accessibility in learning, due to the economy requiring people to acquire new information efficiently and effectively. As a result, this has led to a transition – a new era in distance education (*e-learning* or web-based learning), which has been enabled with the introduction of the Internet and World Wide Web technologies (Li et al., 2008). Due to economical and real-time distribution, the Internet has emerged to become the prevailing method of distributing information and

knowledge (Zhang et al., 2004). Due to accessibility, usability and technology influences, one can enhance the learning experience.

Li et al. (2008) emphasised that distance education is beneficial for students in time and space. Moreover, distributed education provides efficiency for educators, as it enables support of larger learner populations, up-to-date coursework and to track and administer online assessments progress (Li et al., 2008). According to Zhang et al. (2004:76), e-learning advantages include a more "learner-centred & self-paced" approach, "time and location flexibility, cost-effective for learners, potentially available to global audience, unlimited access to knowledge, archival capability for knowledge reuse and sharing". Therefore, students are able to acquire knowledge and skills through methods that are more conducive to individual needs so as to enhance learning. Brinkman et al. (2007) maintained that badly designed educational technology systems can end up becoming an obstacle to learning, causing frustration and boredom, whereas a well-designed system would lead to in-depth understanding approaches and would not encourage the application of surface level approaches.

Within an Online Learning Environment (OLE), challenges in student learning are more complex due to the following:

- OLE students are typically older, have occupations and family responsibilities (Weissman, 2002).
- Students are interested in obtaining the degree for better job opportunities or to retain existing jobs (Weissman, 2002).
- Distributed education isolates students, hence removing motivational factors that involve interaction in a peer group (Weissman, 2002).
- Students are also from diverse backgrounds (Weissman, 2002).
- Some are not computer literate, thus are challenged with course work and the delivery vehicle (Weissman, 2002).
- "Lack of immediate feedback in asynchronous e-learning" (Zhang et al., 2004:76).
- To some people, e-learning can be uncomfortable, frustrating and lead to anxiety and confusion (Zhang et al., 2004)
- There are challenges in the design of online learning, which include the need to address diversity in students' experience that is characteristic among greater student populations (Howard, Johnson & Neitzelm, 2010).

- A further challenge in OLE, as pointed out by Zhang et al. (2004), is an increase in preparation time for the instructor. Piccolli et al. (2001) noted that the facilitator is expected to be flexible due to excessive time and energy demands for effective e-learning.

Piccolli et al. (2001) explained that, e-learning is divided into *three dimensions* (Figure 2.1):

- Learner: forms part of the human dimension. Parker (2003) noted that e-learning increases learner use of technology, and there is more responsibility and control that students assume.
- Facilitator: forms part of the human dimension. The role of the facilitator has been described by Piccolli et al. (2001) as essential. Further, the facilitator is expected to be flexible, due to high time and energy demands for effective e-learning.
- Technology: forms part of the design dimension. Wells (2000) maintained that online instruction delivery is highly dependent on network technologies.

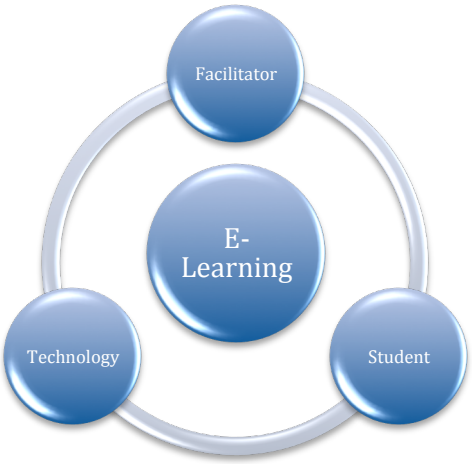


Figure 2.1: Dimensions of e-learning

The difference between e-learning and traditional learning lies in the attention given to students (Duan & Jiang, 2008). Hence, the centre of education in e-learning has transferred to the *lecturer* paying more attention to the students' individual character demands (Duan & Jiang, 2008). Moreover, e-learning offers a learner-centred approach. In comparison to conventional programmes, e-learning is characterised by a higher dropout rate, as it requires more maturity and self-discipline (Zhang et al., 2004).

Zhang et al. (2004) explained that the constructivist theory is characterised by students actively constructing their own knowledge, by applying existing knowledge and real-world experience. Clearly, “learning is an active process, conducted in a self-directed fashion” (Zhang et al., 2004:77). Graven and MacKinnon (2009) have noted that students prefer and will opt for an active learning process even after encountering some difficulties, as active learning is simply more enjoyable. In addition to that, actively involved students tend to learn more and have more positive learning experiences than passive listeners (Graven and MacKinnon, 2009). Zhang et al. (2004), have asserted e-learning supports resource-rich, student-centred and interactive learning and thus provides support of opportunities for constructivist learning.

2.3 ROLE OF DISTANCE EDUCATION AND E-LEARNING IN ENHANCING LEARNING

Vrasidas (2004) highlighted that e-learning and distributed education are characterised by tremendous growth. The author further claimed that within an online environment, *technology* plays a vital role. Weissman (2002) suggested that the success of online learning students is attributed to the enthusiasm for learning with technology, which stimulates thinking processes.

Howard et al. (2010) have emphasised that online education is motivated by broad *asynchronous accessibility*. The nature of a modern e-learning strategy is characterised by a more distributive environment that is independent of time and spatial constraints. Piccolli et al. (2001) stated that asynchronous accessibility provides learners with the ability to choose to learn at their own pace. Furthermore e-learning offers a more flexible learning environment that focuses on learner needs; thus, the approach is more learner-centred, self-directed and self-paced (Piccolli et al., 2001).

The rise and establishment of e-learning technology has led to the development of modern comprehensive e-learning systems (Torrente et al., 2009b). These e-learning systems are targeted to address issues such as lack of motivation or the high dropout rates resulting from the separation between students and instructors, which is characteristic of distance education students. Piccolli et al. (2001) also indicated that high flexibility of e-learning with respect to time, place and pace is a positive motivational factor as far as e-learning is concerned.

E-learning has become an imperative aspect of many learning practices; as such, there is a necessity for “high-quality content” built with concrete pedagogical principles (Torrente et al., 2009b). As e-learning emerges, it is providing support for lifelong learning, as well as providing efficient, effective learning that is crucial to ensuring that students are equipped with up-to-date knowledge and progressive abilities (Zhang et al., 2004).

2.4 MOTIVATION TO DO OBJECT-ORIENTED PROGRAMMING

Govender and Govender (2012) highlighted that there has been a necessity to integrate OOP into any computer science introductory programming course as dictated by global software companies. OOP is defined as a contemporary method to computer programming which mimics real-world entities (Stephen et al., 2012). The emphasis of this approach is the use of objects, which includes both functions and data; objects interact by passing information. According to Haupt, Perscheid, Hirschfeld and Kessler (2010), OOP enables the modelling of a real-world environment such as software objects; therefore, it is considered a paradigm that is effortlessly comprehensible to beginners. The OOP paradigm is becoming a prevailing paradigm, as it is shown to be superior in its organisation of software complexity, which requires a varied set of skills and knowledge (Georgantaki & Retalis, 2007). Govender and Govender (2012) noted that OOP programmers use the central language concepts of inheritance and polymorphism, and design class hierarchies, focusing on encapsulation of class constructs.

Both university and industry have placed increasing importance on the early exposure of students to OOP (Georgantaki & Retalis, 2007). There is a demand for software engineers who are proficient in using the OOP paradigm to analyse and develop systems (Georgantaki & Retalis, 2007). Govender and Govender (2012) noted that another reason for OOP inclusion in the curriculum or course is due to its flexibility and shifts from the inflexibility of the procedural programming paradigm. Georgantaki and Retalis (2007) further stressed the growing emphasis placed on newer OOP languages and tools such as Java by the software industry.

2.5 DIFFICULTIES EXPERIENCED IN TEACHING AND LEARNING PROGRAMMING

Matthews et al. (2009:396) pointed out that learning introductory programming languages requires understanding "the underlying programming paradigm, syntax, logic and the structure". However, Esteves et al. (2009) emphasised that, typically, learning to program entails intensive problem-solving skills. This involves recognising a problem, developing an algorithm to address the problem, and coding the algorithm (Esteves et al., 2009). The algorithm is coded in a specific programming language whose syntax and semantics must be studied (Esteves et al., 2009). Georgantaki and Retalis (2007) further highlighted that students' learning goals should focus on the strengthening of programming practices and problem-solving skills, not on the particular programming language syntax and semantics elements. Programming involves numerous skills and processes (Matthews et al., 2009). Furthermore, students are required to reason methodically, reflect innovatively, rationally and conceptually (Fletcher & Guillen, 2012). Matthews et al. (2009) indicated that novice programmers are required to have a thorough understanding of basic programming concepts so, cognitive understanding is vital for program understanding and writing.

Clearly, programming plays a pivotal role, as it is applicable in both academic and professional development (Stephen et al., 2012). There are varied dynamics associated with the difficulty experienced by novice programmers. Esteves et al. (2009) stated that a wide range of research has focussed on difficulties associated with learning or teaching a programming language.

Researchers have attributed various reasons that have been associated with programming difficulties. Lahtinen, Ala-Mutka and Jarvinen (2005) suggested that programming difficulty is attributed to its abstract nature. Haupt et al. (2010) agreed and reported that teaching objects in OOP could be challenging, as initially these concepts are abstract and not straightforward. The central concepts of OOP include inheritance, casting and polymorphism (Mussai and Liberman, 2012). These concepts are abstract and complex.

Moritz et al. (2005) further highlighted that in the learning process, students face challenges in comprehending the fundamentals of object-oriented design and programming concepts. Teaching computer programming to novices is difficult, as students find programming overwhelming as they attempt to understand abstract and

fundamental complex concepts (Dorn & Sanders, 2003). As with previous studies, Santos (2011) agreed that learning and teaching OOP is still seen as a difficult task. Stephen et al. (2012) maintained that teachers and students both face challenges and complexities involved in programming. Barriers novices face in learning to program include learning the syntax as well as the model of computation (Hundhausen, Farley & Brown, 2006). Stephen et al. (2012) also agreed and recognised pedagogical approaches used in the instruction process as an underlying cause of the poor performance and misconceptions in programming. However, Gomes et al. (2012) attributed the following influences to students' learning performance in programming: contextual knowledge and perceptions, instructional approaches, and social circumstance.

According to Truong et al. (2005), because of the difficulties experienced in both learning and teaching OOP, the result is lack of confidence and reduced perception of programming. In addition, students are inclined to lose self-confidence, particularly due to the abstract nature of concepts and also it is difficult to visualise or understand concepts (Dorn & Sanders, 2003). An additional influence is lack of motivation due to difficulties in basic concept understanding as well as applying them accurately in more complex constructs. Esteves et al. (2009) further emphasised that it is vital to assimilate knowledge of concepts and approaches to use when solving problems within the learning process. A further challenge encountered by novice programmers is that of sustaining students' interest in learning to program (Matthews et al., 2009). The reason attributed to the aforementioned challenge is that students, probably due to misconceptions about programming activities, are exposed to video games and graphical user interfaces, whereas normally the programming activities that they were involved in are text-based Input or Output (I/O), possibly masked with simple messages and dialog boxes (Dorn & Sanders, 2003).

Matthews et al. (2009) attributed the programming learning environment as a contributing factor causing difficulties in learning programming. Truong et al. (2005) suggested that the difficulties novices experience could possibly be due to program text editor usage knowledge, installing a language compiler, and how to compile and debug the program as well as understanding the error messages displayed.

Gomes et al. (2012) further noted that difficulty in learning introductory programming leads to dropping out or poor performance. Programming is difficult particularly for novices, resulting in a high level of failure in introductory programming courses

(Esteves et al., 2009). Esteves et al. (2009) attributed the failure rate of novice programmers to difficulties in understanding basic programming concepts. These authors emphasised one of the reasons for high attrition rate in introductory CS is that students struggle with designing a solution to a problem, partitioning code into simple sub-components, and testing for error situations. Moreover, programming at a higher level also requires analysis of a problem, comprehension of a solution, coding in a specific computer language code, and testing and correcting errors (Esteves et al., 2009). Hundhausen et al. (2006:1), on the other hand, attributed factors that affect attrition rate as follows: "individual student differences; a lack of a sense of community; deficient pedagogical approaches; inadequate novice programming environments".

Truong et al. (2005) stated that diversity of experience levels in students' knowledge and lack of personalised attention in DE students ultimately leads to inadequate learning experience and increased likelihood of failure. Clearly, programming is complex; this is particularly evident from the high attrition rates in introductory programming, and it remains one of the most urgent challenges (Hundhausen et al., 2006).

2.6 SURVEY OF VISUALISATION TOOLS RELATING TO OOP PEDAGOGY TO ADDRESS DIFFICULTIES IN OOP

From the observation of Esteves et al. (2009), the diversity of experience levels in distance education students' knowledge results in a complex traditional teaching process. Current technological advances have unlocked new learning opportunities (Esteves et al., 2009). Georgantaki and Retalis (2007) as well as Matthews et al. (2009) noted that various researchers have conducted studies on effective instructional processes for OO design and programming, focusing on the implementation of educational tools to assist in mitigating learning barriers for conceptualising the OOP philosophy and its concepts (Georgantaki & Retalis, 2007).

Georgantaki and Retalis (2007) have identified and presented tools for teaching OOP into seven main categories. The categories are as follows: "programming micro worlds, educational programming environments, tools for enhancing the capabilities of programming environments, software tools supporting the 'objects-first' teaching approach, gaming environments, and tools based on specific educational programming languages tools" (Georgantaki & Retalis, 2007:112).

In this research, a literature review of various e-learning tools designed for supporting the instructional process of OOP so as to ease novice programmers' difficulties was conducted. There are several visualisation tools with a diverse range of functionalities. Also, there is a rich set of visualisation tool paradigms for novice programmers to choose from, which are discussed in the next sub-sections; these range from those allowing students to control both symbols and code to those that reject text-based programming.

2.6.1 AGUIA

Santos (2011) implemented an approach that consisted of new methods for understanding OOP through utilising tool support. The author presented a pedagogical tool called AGUIA/J tool for interactive experimentation and visualisation of object-oriented Java programs. The researcher proposed this tool, as it exploits graphical conceptual metaphors based on graphical user interface (GUI) elements in order to understand OOP. This study, demonstrated using AGUIA/J evaluation within the course in preliminary study groups, contributed significantly to enhancing motivation and lowering dropout rates.

2.6.2 ALICE

Alice (http://www.alice.org) is a 3D (three-dimensional) programming animation environment which includes 3D characters, objects and animations, developed at Carnegie Mellon University (CMU) in which objects, classes, methods, events and their behaviours are visualised (Georgantaki & Retalis, 2007; Giordano & Carlisle, 2006). Alice enables novice programmers to construct 3D virtual worlds, including animations and games, as they learn to construct programs visually by selecting statements from available operations (Rößling et al., 2008). To control the movement and activities of objects, students can visually create programs through a series of drag and drop functions. Thereafter, these programs can be viewed and executed in a 3D micro world, thus preventing syntax errors (Hundhausen et al., 2006; Li & Watson, 2011; Radošević, Orehovački & Lovrenčić, 2009).

Alice was designed to accelerate student accomplishment in problem-solving skills in order to design and implement software solutions (Fletcher & Guillen, 2012). Fletcher and Guillen (2012) explained that problem-solving demands methodical, creative, logical and abstract reasoning. Stephen et al. (2012) indicated that, the purpose of

developing the Alice tool was to create an amusing and appealing initial introductory programming experience as well as to encourage girls into the computer science field. The strength of Alice is in its visual feedback environment, enabling students to understand essential concepts as well as be actively involved in the learning process (Esteves et al., 2009).

Stephen et al. (2012) highlighted that Alice's focus is on introducing OOP concepts using the syntax of Java, C++ and C#. Alice supports object-oriented programming by writing simple scripts in which its users can control their object's appearance and behaviour (Esteves et al., 2009). Program Visualisation (PV) enables visualisation of how animation programs run, enables ease of understanding of the association between the programming statements and constructs and the behaviour of the animations, thus enabling understanding the fundamentals of computing (Cooper, Nam & Si, 2012; Radošević et al., 2009).

Fletcher and Guillen (2012) developed a programming course using Alice, where the outcome of assignments is an animated movie. This research involved an assignment requiring students to apply OOP and design skills in creating an animation. Students were required to follow good programming style without concern for syntax, implement solutions using loops and conditional logic, and to create documentation using flowcharts. Students worked in a 3D environment with immediate visual feedback. The results of the research proved to be positive; students were able to apply object-oriented concepts, and even enjoyed the course, which enhanced learning and served to be a motivation in pursuing fields in computer science.

2.6.3 ALVIS

ALVIS Live! provides immediate feedback through an algorithm editing and visualisation model. Furthermore, it offers an interface for producing object creation declarations through direct manipulation, this enables easy layout of program objects in an animation window using a set of tools. ALVIS also caters for standard graphical illustrations of variables, arrays, and array indexes (Hundhausen & Brown, 2005).

Hundhausen and Brown (2005) explored the importance of a studio-based pedagogical approach, which is an actively engaged algorithm visualisation learning technology to teach novices to program. The context in which this research was

carried out was in a Computer Science 1 (CS1) introductory module on algorithmic problem-solving. This research involved pairs of students who were required to solve particular algorithm design problems involving constructing algorithmic solutions and accompanying visualisations; thereafter, visualisations were presented for feedback and discussion with different tools to construct their visualisations (Hundhausen & Brown, 2005). The tools that were used in the Art Supply and ALVIS sessions included art supplies and the ALVIS programming and visualisation environment respectively. However, in the next semester, students used ALVIS Live! an updated version of the ALVIS software (Hundhausen & Brown, 2005).

Hundhausen and Brown (2005) used a diverse set of ethnographic field techniques and collected a detailed set of data with respect to the pedagogical value and the type of technology that would be most suitable for the tasks. Primary field techniques included *participant and videotaping* in all sessions. Secondary field techniques included *artefact collection,* which evaluated semantic accuracy in participants' code as well as analysed students' visualisations. *Interviews* were also conducted where emerging themes were transcribed. Finally, they used *open-ended questionnaires* based on the student's approach and perceptions to tasks completed in their Studio Experiences (Hundhausen & Brown, 2005).

Students were involved in three primary activities in the Studio Experiences as observed by Hundhausen and Brown (2005):

- *Algorithm development and accompanying visual representations:* Due to the immediate visual feedback specified by the tool, this resulted in less reliance by an instructor as well as students developing a significantly more correct and faster coding process. Constructing personalised visual depictions, algorithmic design problems resulted in increased engagement and enjoyment by students. However, results were inconclusive with respect to the benefit in their understanding of algorithms.
- *Visualisation and story development:* The ALVIS tool supported the development of visualisations with deeper narratives and more elegant graphical elements.
- *Visualisation presentation and discussion:* Presentation sessions' results indicated that student presenters made a larger contribution. Nevertheless, students that used the ALVIS tool seemed to encourage discussions with a stronger emphasis on the particular algorithm behaviour aspects, this resulted

in more cooperative associations and repairing of semantic errors.

2.6.4 CIMEL ITS

"Objects-first" is an effective approach used in the instruction process of OOP by introducing key concepts of OOP. However, it does not assist students in learning fundamental concepts of software development that include problem-solving skills and design, as this approach emphasises coding (Wei et al., 2005; Moritz et al., 2005). "Design first" approach in introductory courses involves students learning the fundamental aspects of object-oriented analysis and design as problem-solving skills (Wei et al., 2005; Moritz et al., 2005).

Wei et al. (2005) and Moritz et al. (2005) suggested that in order to assist students, students should learn object-oriented design and programming and Java programming, using elements of UML before coding. These authors propose Constructive, collaborative, Inquiry-based Multimedia E-Learning (CIMEL) Intelligent Tutoring System (ITS), an intelligent tutoring system that offers personalised tutoring to assist novices with different learning styles in a CS1 course. CIMEL ITS interfaces with both the Eclipse integrated development environment (IDE) and CIMEL multimedia to give students a personalised, appropriate support as they are learning and applying their knowledge in problem-solving tasks (Wei et al., 2005; Moritz et al., 2005).

Thus, CIMEL ITS interacts with students through two systems. These systems include Eclipse IDE and CIMEL. The Eclipse IDE is explicitly selected to support the design-first curriculum where students design and code their project tasks. It is an open-source IDE that is extensively utilised by developers in both universities and businesses. The ITS will detect the students' progress and offer adapted individual support based on pedagogical approaches. The CIMEL multimedia courseware is effective in assisting with introductory OOP concepts. The CIMEL ITS refers students to review sections for reinforcement; it also establishes a student's level of understanding based on performance in interactive activities within CIMEL (Moritz et al., 2005).

2.6.5 BLUEJ

The BlueJ tool is one of the most extensively utilised educational programming environments (www.bluej.org) (Ben-Ari, Ragonis & Ben-Bassat Levy, 2002). It is an

IDE specifically designed for teaching OOP in Java to secondary and first-year post-secondary students (Ben-Ari et.al, 2002). BlueJ uses an "objects-first" approach in teaching Java; this is centred on the basis that students study OOP from the beginning in order to avoid a change of programming paradigm (Ben-Ari et.al, 2002). Among the important features of BlueJ are its simplicity and static visualisation of the class structure as a UML diagram. Another important feature of BlueJ includes the transition in environment from visual to code view and ease of compilation with a combination of error messages and source editor (Georgantaki & Retalis, 2007).

BlueJ graphically presents OO UML-like diagrams of software project structure; thus, it is an ideal tool to present OOP fundamental concepts such as data abstraction and encapsulation, inheritance and polymorphism, message passing including others which are typically challenging for students to comprehend (Stephen et al., 2012). Further, its "visualisation" capability of projects, objects and classes enables users to "interact" with classes constructing objects and invoking methods. Also, users can examine the state, the implementation and interface view for the classes (Georgantaki & Retalis, 2007).

Georgantaki and Retalis (2007) designed a technology-enhanced learning script for teaching OOP based on the "modelling first" approach in order to investigate and confirm findings of the comparison between the combination of educational tools with professional environments for teaching OOP and design. These authors conducted an OOP seminar, which had been devised and established on a technology-enhanced learning script. The script consists of learning activities that take place during three central stages of the learning process: (i) the first phase entailed observation of the development of an OOP example, (ii) the second phase entailed problem-solving tasks with direction, and (iii) the third phase involved autonomous problem-solving tasks in developing an OO software application for submission. During the seminar, the following tools were used: ArgoUML CASE tool, BlueJ educational environment, and SUN One Studio IDE. The ArgoUML CASE tool (http://argouml.tigris.org/) was used for design process. The BlueJ educational environment was used for further implementation of examples. The SUN One Studio IDE (http://java.sun.com/) was used in the instruction process of the application (Georgantaki & Retalis, 2007). Georgantaki and Retalis (2007) used two types of questionnaires and focus group interviews to collect students' perceptions; additionally, they analysed students' activities provided during the second phase. The second questionnaire comprised mostly quantitative questions measured on a

five-point Likert-type scale, and a few qualitative questions were used to assess the seminar's effectiveness (Georgantaki & Retalis, 2007).

The results of the evaluation of the OOP seminar were positive and indicated that students valued the BlueJ tool for its simplicity and usability. The analysis of students' assignment indicated that the assignments completed using BlueJ during the seminar improved their knowledge, skills and understanding of OOP concepts (Georgantaki & Retalis, 2007).

2.6.6 GREENFOOT

Greenfoot is a pedagogical IDE for novice programmers; it includes all standard IDE elements. Among these are the editor, compiler, execution mechanism and class browser. The platform incorporates the standard tools with pedagogical tools, which include interactive object invocation, inspection and class visualisation. The platform provides support for highly visual and interactive programs, providing both a framework and an animated environment. Moreover, various visual applications can be created, for example, simulations and games (Kölling, 2008; Henriksen, 2006).

The primary aim of Greenfoot was to express object-oriented concepts and principles in a clear, simple and comprehensible approach (Gallant & Mahmoud, 2008). Georgantaki and Retalis (2007) indicated that the Greenfoot environment is constructed with visual interactions between the world and the objects. This programming environment is a complete open-source Java environment. This approach enables an interactive, graphical and stimulating method to learning OOP in Java by enabling simple creation of objects' visual representation, thus ensuring that students can focus on the application logic, as the interface encourages experimentation with objects (Gallant & Mahmoud, 2008).

According to Georgantaki and Retalis (2007), the Greenfoot design was motivated by mixing beneficial characteristics of some existing tools. Some of these characteristics are highlighted below.

1) As with Karel the Robot, environment is simple and includes the exceptional visualisation aspect of objects, their state and behaviour (Georgantaki & Retalis, 2007).
2) As with the BlueJ environment, the Greenfoot design offers good

support for direct object interaction (Henriksen & Kölling, 2004; Georgantaki & Retalis, 2007). It had been developed by the same company that developed BlueJ (Gallant & Mahmoud, 2008).

The tool can be used to support diverse education levels, as it can be adjusted according to the significance of the specific group and subject areas. With high school/CS0 courses, Kölling (2008) noted that associating learning to program to recognised applicable domains, for example, computer games, could be beneficial in increasing motivation. The authors noted that pedagogical tools are beneficial for CS introductory students in comprehending OOP concepts. For higher levels of education, the Greenfoot environment can also be utilised, as there is no limit to the complexity that programs may take, as the language is full Java (Kölling, 2008).

Vilner, Zur and Tavor (2011) used Greenfoot, as it provided a visual environment that catered for programmers to create and manipulate objects seeing immediate changes in properties and behaviour. The aim of this research was to simplify the programming of complex examples and increase student motivation; thus, the authors looked for an environment that enabled presentation of visual examples. Greenfoot enabled students to focus on the core issues without having to deal with the aspects of GUI, through the use of scenarios and the creation of complex examples, maintaining good use of visual representations. The Greenfoot environment demonstrates the subjects of inheritance and polymorphism.

The research of Vilner et al. (2011) entailed two aspects. Concerning the pedagogical aspects, the authors considered students' attitudes towards engaging with the Greenfoot environment. The second aspect dealt with students' satisfaction with respect to the technical aspects. The results demonstrated that the impact of implementing a Greenfoot environment proved to be a positive experience, as students enjoyed and were stimulated when working with Greenfoot; students felt concepts of inheritance were better understood and it was easy for students to use Greenfoot. Clearly, from this research, Greenfoot was very successful when integrated into CS 1.

To address the challenges and difficulties encountered in programming, Gallant and Mahmoud (2008) emphasised that students need to be challenged, tested and amused, as well as be actively involved in their own learning process. The approach entails students learning fundamental skills stimulating them in such a way that they

are unaware that they are learning (Gallant & Mahmoud, 2008). Thus, this provides an approach for assisting educators in creating an engaging, positive and motivating experience for novices to increase retention and to learn the fundamentals of computer programming using the Java programming language and the Greenfoot programming environment (Gallant & Mahmoud, 2008).

Gallant and Mahmoud (2008) conducted a study which included concepts of Java-based experiments, interrelated labs and a capstone project utilising the Greenfoot programming environment for teaching introductory Java to post-secondary students in their first year of tertiary education. To realise this, these authors designed, developed and implemented a completely new working scenario called Going to the Moon into the Greenfoot programming environment. This scenario proved to be effective in teaching Java in CS1, as it ensured students had a comprehensive base of Java knowledge to prepare them for further Java studies (Gallant & Mahmoud, 2008). Gallant and Mahmoud (2008) highlighted that due to Greenfoot being self-contained in a simple environment, it is favoured as a good distance education tool; as such, they are also exploring applying research to include distance education students.

2.6.7 GAME-BASED APPROACH TO LEARNING PROGRAMMING

Learning to program is challenging and complex; nevertheless, it is still an essential skill required to achieve a computer science degree (Li & Watson, 2011). Li and Watson (2011) utilised a game-based learning virtual environment to support novice programmers in order to improve learning and retention through engagement of the learner. Graven and MacKinnon (2009) found that students' experience with a game context resulted in an engaging, enjoyable and useful way to present learning material. These authors highlighted that game construction tasks are beneficial, as the relationship between syntax and output can be clearly seen; accordingly, it is easier for students to understand as they graphically experiment with various programming concepts (Li & Watson, 2011).

Li and Watson (2011) utilised an environment that uses blending programming learning game construction assignments in order to make basic programming more intuitive, comprehensive and an easier method to learn programming. Their research aim was to assist in learning programming at each of the four stages of learning programming concepts. These stages include Receiving, Visualising, Reinforcing,

and Applying and Synthesising. A tile-based game, which comprised of visualisation techniques and task completion was used; therefore, students could visualise and learn fundamental programming concepts through game object manipulation. The tile-based game is simple and easy for novices to comprehend. Furthermore, it is also flexible, as it enables development of different types of games built upon the same core principles. Thus, this approach enabled students to apply their new knowledge and to construct innovative applications. This is essential, as it assists in creating a more motivating experience and in aiding students to have a deeper understanding of the learning process (Li & Watson, 2011).

Li and Watson (2011) have indicated that game construction tasks form a natural parallel with central programming concepts, hence are suitable for programming in education. Therefore, these authors have closely aligned game and program construction tasks along with meaningful graphical visualisation, as they believe it could provide a profound effect towards effective learning (Li & Watson, 2011). The methodology emphasises the learning of programming concepts through Java, a "real" programming language. The environment could be portrayed as an exploratory learning tool where students interact and inspect the state of visual objects they created (Li & Watson, 2011). The authoring-based game-based learning utilised in this study is superior to other existing approaches, as Li and Watson (2011) offer methodologies to assist in the understanding of variables or data structures and the development of simple programming logic. The authors believe that this will have a profound impact on programming education (Li & Watson, 2011). Quantitative results were mostly positive, indicating this technique is appropriate in teaching novice programmers elementary concepts (Li & Watson, 2011).

2.6.8 JEROO

Jeroo is an integrated development environment and micro world. Jeroo and its user documentation can be accessed at http://www.nwmissouri.edu/~sanders/Jeroo/Jeroo.html. The design of Jeroo was aimed to assist educators to teach fundamental programming skills. This involves problem analysis and decomposition, incremental development of algorithms, and formulation of pre- and post-conditions. Jeroo assists in increasing confidence in a student's ability to write programs due to the personalised nature of this IDE. The feedback offered by animated execution with immediate highlighting and programs can be executed stepwise or continuously at different speeds. Due to

the game-like nature of Jeroo and its graphically rich environment, this results in more enthusiasm with learning basic concepts (Dorn & Sanders, 2003).

Jeroo is based on the metaphor of a rare kind of Australian kangaroo-like animal bouncing throughout an island, picking flowers and dodging nets (Sanders & Dorn, 2003). The metaphor is non-technical, non-violent and familiar thus students can comprehend and feel content with it (Dorn & Sanders, 2003).

Dorn and Sanders (2003) noted that the Jeroo tool has four major components. First, the user interface has a single-screen development environment in which everything is always visible and easily grasped. Secondly, the Jeroo language syntax was designed to reflect the standard syntax of Java, C++, and C#. Further, it was intended to emphasise on objects, methods, and fundamental control structures. Thirdly, the editor feature of the Jeroo tool enables ease of editing source code and island layout. Lastly, the IDE incorporates a runtime module that demonstrates the association between source code and visible action, providing a rich instructional environment. Syntax and runtime errors are readily identified; also, the semantics of the control structures are easily evident, and the interaction between methods is shown (Sanders & Dorn, 2003).

Dorn & Sanders (2003) have tested Jeroo in Northwest Missouri State University, and the results indicate Jeroo to be an effective tool for novices. The authors have categorised benefits of Jeroo as programming concepts, programming practices, and student satisfaction (Dorn & Sanders, 2003). Moreover, the authors felt that with respect to programming concepts, Jeroo has resulted in an enhanced understanding of control structures, methods, and objects. Thus, understanding of these fundamental concepts enabled confidence and a smoothed transition when moved into Java (Dorn & Sanders, 2003).

Further, Jeroo has benefited students to develop a more advanced programming style. Students are capable of decomposing problems, planning and also reading and tracing programs. Additionally, students are able to note the association between source code and changes in program state due to source code being highlighted during execution (Dorn & Sanders, 2003).

Student satisfaction was evaluated using a five-point Likert scale questionnaire after the students had made the transition to Java. In a Java programming class, it

resulted in substantial increase in student comfort and confidence levels, particularly evident among females. Irrespective of gender or programming experience, all students had experienced value added in using Jeroo in increasing confidence and comfort. Furthermore, experienced programming students felt that Jeroo is effective in teaching the fundamental concepts of Java (Dorn & Sanders, 2003).

2.6.9 JELIOT

Jeliot is an interactive open-source program visualisation tool aimed at assisting novice students to learn procedural programming and OOP (Moreno, Myller, Sutinen & Ben-Ari, 2004). The fundamental feature is in its visualisation of data and control flows of Java programs and OOP features (Moreno et al., 2004). Jeliot has a simple user interface with the aim of encouraging active experimental learning in the development of programs and simultaneously examining visual representation of program execution (Rößling et al., 2008). The Jeliot interface consists of four distinct areas: code view, animation view, control panel, and output console. This tool has the ability to visualise a huge subset of basic Java programs focusing on visualisation of complex concepts of inheritance and objects not easily grasped by novices (Bednarik, Moreno & Myller, 2006). Jeliot supports objects-first or fundamentals-first approaches in the teaching of introductory programming (Myller & Nuutinen, 2006). Additionally, it enables users to view the static history of an execution (Myller & Nuutinen, 2006).

Jeliot has a research-oriented development process; as such, it is based on the earlier versions' design and empirical evaluations (Myller & Nuutinen, 2006). Thus, it enables modularity and extensibility of the earlier versions to be implemented in different environments, therefore enabling either to extend the opportunities for visualisation or to support diverse user populations (Myller & Nuutinen, 2006). To support the extensions of Jeliot 3 the source codes of the system have been published under the GPL licence and is freely available at http://cs.joensuu.fi/jeliot/ (Myller & Nuutinen, 2006). The philosophy of Jeliot e-learning tools provides full access to the source codes, providing for both internal and external extensibility (Bednarik et al., 2006).

The opportunity for both academic research and feedback from the user community contributes to its development through adapting and extending the capabilities of this tool without constraints; this has had a positive effect on the tool (Bednarik et al.,

2006). Among the benefits to be derived is creating an enriched learning experience that supports internal motivation, which arises, as environments can be adapted to users' needs and learning preferences (Rößling et al., 2008). Thus, it can be used in distance education, as it can be adapted to user needs (Myller & Nuutinen, 2006).

The plug-in architecture between BlueJ and Jeliot 3 combines animation created by Jeliot visualisation with the standard UML notation and interaction with objects of BlueJ. The JeCo plug-in supports both cognitive and social processes in a distance education, which is a combination of Jeliot 3 and Woven Stories into a collaborative program visualisation programming environment. Therefore, it supports both synchronous and asynchronous methods of communication and learning (Myller & Nuutinen, 2006). The Editing Java Easily (EJE) is an additional plug-in. This development environment includes particular features enabling the use of a tool directly from a web page to enable ease of use among novices (Myller & Nuutinen, 2006).

2.6.10 JIVE

The Java Interactive Visualisation Environment (JIVE) software tool is an innovative method for runtime graphical notations of OOP (Gestwicki, 2004). Gestwicki (2004) utilised the JIVE prototype and its related visual representations in various courses ranging from undergraduate to more advanced level. JIVE includes the following three features of visualisation: visual semantics, interactive execution and effective drawing (Gestwicki, 2004).

With respect to visual semantics, JIVE uses two corresponding views to visualise an object structure, namely, an object diagram to view the program's current state and a sequence diagram to show the program's execution history. This notation enables the description of objects as environments of execution, provides visualisations of complex object-oriented features, for example, inheritance and method overriding, and it further simplifies Java semantics (Gestwicki, 2004).

With respect to interactive execution and queries, a database supports forward and backward interactive execution. Additionally, queries on program behaviour at runtime are supported, as program execution is highly dynamic.

Regarding effective drawing, JIVE produces clear, legible diagrams (Gestwicki, 2004). Gestwicki (2004) utilised the JIVE prototype and its related visual representations in various programmes.

2.6.11 JAVAVIS

Javavis uses a Java Debugging Interface to visualise the state of a program (Moreno et al., 2004). The strength of Javavis lies in its ability to visualise the state of the program and its changes during execution, making it possible to obtain information about the runtime behaviour of the programs (Moreno et al., 2004). This is shown using 'animated UML-like object and sequence diagrams' (Bednarik et al., 2006:51). However, Javavis is not suitable for novices; it should rather be used for more advanced programming courses, as the visualisations require users to understand UML and fundamentals of programming (Moreno et al., 2004).

2.6.12 STRUCTURED FLOW CHART

SFC (Structured Flow Chart) Editor is a structured flowchart editor (Carlisle, 2009). SFC emphasises structured flowchart style development and creates generic or C++ pseudocode (Giordano & Carlisle, 2006). Limitations of SFC are that students can only manipulate flowcharts and symbols, not pseudocode (Giordano & Carlisle, 2006). Further, SFC uses an imperative subset of C++ code it generates (Carlisle, 2009).

2.6.13 JGRASP

JGrasp is an educational programming environment supporting an objects-first teaching approach (Georgantaki & Retalis, 2007). JGrasp is available at http://www.eng.auburn.edu/grasp/. Stephen et al. (2012) described jGrasp as a Java IDE with visualisation capabilities. A further strength of jGRASP is its ability to visualise other languages even though it is Java based (Stephen et al., 2012). JGrasp utilises Control Structure Diagrams (CSDs) for program visualisation in order to view source code and program design (Stephen et al., 2012).

As a pedagogic IDE JGrasp is beneficial for teaching OOP; however, it does not offer true visual semantics or fully interactive execution (Gestwicki, 2004). Further, it uses static visualisations; therefore, it is not very interactive (Stephen et al., 2012). JGrasp is not suitable for novice programmers, as it requires programming knowledge, and there is no provision for code highlighting (Stephen et al., 2012).

2.6.14 VILLE

VILLE is a dynamic program visualisation tool with the key purpose of supporting the learning process of novice programmers. This tool can be used both in lectures and for independent learning. The principal purpose of VILLE is to support a higher level of abstraction focusing on a programming language independency paradigm. VILLE currently supports Java, pseudocode and C++. The pseudocode's definition can be changed according to instructor requirements. The built-in syntax editor enables users to add new languages or alter the syntax of built-in languages and to extend the language support of the tool. Further visualisations are displayed for any of the languages VILLE supports. To highlight the language independency paradigm, the tool has a parallel view, which simultaneously displays a program in two languages. Thus, a user can understand the different programming concepts instead of concentrating on syntax by discovering relationships in their basic functionalities (Rajala, Laakso, Kaila & Salakoski, 2008).

VILLE has the ability to trace program execution per line and observing outputs and changes in variable values. Furthermore, the visualisation available in VILLE is effective and easy to understand, as there is an automatically generated textual explanation of each line of program code, as well as a description of the role of variables. This enables interpretation of program execution, which is fundamental in learning to program, because students can comprehend the associations of programming concepts and structures (Rajala et al., 2008).

VILLE provides a set of predefined extendible examples permitting students to follow changes in execution and visualisation. These can be published on the web, thus enabling flexible engagement learning sessions independent of time and location. Another customisation feature offered by VILLE includes interactivity through design and use of pop-up questions (Rajala et al., 2008).

Rajala et al. (2008) conducted a study on the effectiveness of VILLE on learning basic programming concepts. The following research questions were addressed: (1) whether the tool would assist in learning to program and (2) whether there are any variances in learning when considering previous programming experience. The study was conducted at the University of Turku, Finland, in the fall of 2007 on students in their first programming course (Rajala et al., 2008).

For the first research question, students were randomly divided into two groups. The control group used conventional textual material, while in the treatment group, the equivalent material was extended with interactive examples using VILLE during the sessions. Results were compared between the two groups (Rajala et al., 2008). With respect to the second research question, students' learning performance was compared for both treatment and control groups where previous programming experience was taken into consideration (Rajala et al., 2008).

Overall results indicated that the tool enhanced students' learning of basic programming concepts (Rajala et al., 2008). Additional evidence revealed that the tool improved students' learning irrespective of previous programming experience. Furthermore, the VILLE tool was beneficial among novices, successfully closing the gap of previous programming experience in a very short training period (Rajala et al., 2008).

2.6.15 RAPTOR TOOL

Rapid Algorithmic Prototyping Tool for Ordered Reasoning (Raptor) was designed by Carlisle et al. (2005) primarily to minimise barriers to programming, focusing on limiting syntax complexity and offering concrete visual representations on which to work. Raptor runs in the .NET Framework and was developed using Ada, C# and C++ (Carlisle et al., 2005). The goal of Raptor is to design and execute algorithms independent of a programming language. Raptor is an iconic programming environment where programs are created visually using UML and flowcharts.

Raptor is a visual programming environment built with flowcharts (Brown, n.d.(a)). The core purpose of Raptor was to teach students and enable visualisation in order to assist students in understanding fundamental programming concepts of classes and methods in object-oriented programming and to become better at problem-solving through the use of algorithms (Fletcher & Guillen, 2012). Problem-solving entails reasoning systematically, to think creatively, logically and abstractly (Fletcher & Guillen, 2012).

Carlisle et al. (2005) noted that the use of a specific programming language in an introductory computing course disregards the teaching of fundamental issues of algorithmic problem-solving, by focusing on syntactic difficulties experienced by students. Carlisle et al. (2005) suggested that this is possibly due to the highly textual

nature of most programming environments instead of adopting a more visual approach. The purpose of the study was to evaluate Raptor, of which the primary goal was to improve student problem-solving through minimising syntactic complexity (Carlisle et al., 2005).

The study entailed comparing the outcome of three algorithmic design questions on the final exam of the compulsory course of "Introduction to Computing". The questions used included a succinct problem statement requiring students to compose and solve an algorithmic problem. Algorithms could be articulated in a flowchart, Ada, or MATLAB. The majority chose to use flowcharts (Carlisle et al., 2005).

Carlisle et al. (2005) developed a seven-point Likert scale survey consisting of nine questions that ranges from *strongly disagree* to *strongly agree*. This research aimed to assess ease of use; Help System; effectiveness in developing problems skills and understanding how computer programs operate; an engaging experience; and testing and debugging programs (Carlisle et al., 2005).

Results indicated that students performed considerably better when taught with Raptor than with Ada or MATLAB. This research utilised two-sample t-tests, which were conducted on the results. This study confirmed that students preferred and were more successful at designing algorithms visually when compared to the traditional language or writing of flowcharts, as it assisted students to easily track the program control flow and to solve problems.

Carlisle et al. (2005) concluded that Raptor offers a simple environment for developing algorithms. Raptor is flexible, as it enables instructors to modify the environment and develop more stimulating tasks by enhancing the built-in procedures. Furthermore, the study concluded that students opted to create algorithms visually using flowcharts. Further results indicate that students developed more enhanced problem-solving skills with Raptor in comparison to using a more traditional, non-visual language approach. However, these results did not cover problem-solving using arrays, as array handling in Raptor still required improvements at the time this research was conducted. The results were encouraging; thus, Carlisle et al. (2005) have implemented numerous additional ease-of-use features as well as students' ideas from the survey.

Giordano and Carlisle (2006) emphasised that in order to enhance learner engagement and pedagogical process, this can be achieved through the use of algorithmic visualisation systems and techniques. For a visualisation tool to be of an educational benefit, it should avoid being too cumbersome and having unnecessary overheads (Giordano & Carlisle, 2006).

Giordano and Carlisle (2006) explored the effectiveness of an introductory IT course by comparing "commercial-off-the-shelf" (COTS) diagram tool to Raptor. The COTS tool enabled algorithm visualisation, had a consistent approach and automated tool support. However, obstacles it resulted in were "unbounded problem space" and there was no feedback with respect to designs, students also encountered difficulties understanding flowchart basics. Though flowcharting methodology in combination with the COTS tool achieved the course goals and provided a well-understood framework, a search for a better tool was required, as evaluations at the end of the course considered the COTS tool as challenging to operate and cumbersome. Furthermore, to address the high attrition, this triggered a change in "design paradigms" (Giordano & Carlisle, 2006).

Giordano and Carlisle (2006) outlined the following requirements for the replacement tool:

1. Teach fundamentals in a straightforward manner.
2. Visualisation tool support leading to better higher-level Java programming skills.
3. Ensuring syntax and semantic correctness in Java programming.

To minimise the degree of transition, Raptor was integrated into all 32 sections of the course. The pilot study was undertaken to evaluate the effectiveness of Raptor objectively, even though most instructors felt Raptor was a superior tool. Flowcharting and Java programming were chosen as the objective measures of performance. Weaker students were identified and moved to a more conducive environment, thus allowing them to move at a slower pace and solve simpler problems still within course objectives; these students were in the regular sections. The other population consisted of students who were engaged in more challenging problems in the advanced section.

The questionnaire administered by Giordano and Carlisle (2006) included views on:

- Effectiveness as a design tool.
- Made programming in Java easier.
- Java programming in Raptor would be easier with first implementing the design in Raptor.
- Support course goals.
- Enhanced feelings about programming.

Giordano and Carlisle (2006) chose the Raptor flowchart tool as the superior tool from the survey of visualisation tools, as it provided the least disruption to their existing course when implementing it as well as complementing the current course content. The positive features and capabilities included:

- "Standard symbol template visible to user
- Constrained symbol placement
- Input and output handled automatically
- Real-time syntax checking
- Flowcharts actually 'run'; one may run a flowchart to completion or step through each symbol discretely
- Real-time variable inspection at run-time" (Giordano & Carlisle, 2006:117).

Disadvantages of the Raptor tool at the time when this research was undertaken showed the following:

- Raptor used different flowchart symbols than current methodology.
- The Raptor tool was mostly beneficial for novice programmers; it was not meant to lead to text-based programming in one semester.

Despite the foregoing disadvantages, researchers acquired a modified version that would be more feasible and in line with the current pedagogy and flowcharting methodology. This showed that the Raptor tool is flexible in its development for specific academic purposes; this was the USMA version of Raptor.

Giordano and Carlisle (2006) used both quantitative and qualitative data in supporting their decision to implement Raptor throughout the course. In comparison to COTS, Raptor was the preferred tool and easier for the development of Java

applications. Results showed that the Raptor visual programming environment when developing algorithms using flowcharts actively engaged students; on the other hand, COTS was found to be complex and cumbersome. Results showed that Raptor resulted in positive learning and enhanced learning outcomes. Furthermore, it is suitable for novices so as to enable them to write structured programs and reinforces design methodology. After this study was conducted, Raptor was used solely in this course at US Military Academy, which entails IT-enabled problem-solving algorithm design and development in Java (Giordano & Carlisle, 2006).

2.7 WHY VISUALISATION TOOLS

This section addresses why the complexity of the execution of OOP programs seems to require visualisation in improving novice's programming experiences as well as the type of visualisations environments to consider when selecting a web-based learning tool.

Rajala et al. (2008) defined visualisation as presenting the execution of programs or algorithms with visual elements. There are several visualisation tools with a diverse range of functionalities. Giordano and Carlisle (2006) suggested that there are two broad groups of visualisation tools. First, iconic programming that depicts to students some form of code or pseudocode containing the use of symbols, visual structures or flowcharts to demonstrate some lower-level syntactic process that are abstracted for the user. The second category of visualisation tools hides code from students.

Dillon et al. (2012) highlighted that to improve novices' programming experience, visual environments have been implemented in contrast to a command line environment. Carlisle et al. (2005) suggested this is possibly due to the "highly textual nature of most programming environments" as it "works against the learning style of the majority students" (Carlisle et al., 2005:176). Adopting a visual approach is preferred, as students are more inclined to learn visually through the use of flowcharts and iconic programming languages (Carlisle, 2009). These environments possess more assistive features for programming than a command line environment. Novice programming environments have various assistive features among them include, syntax highlighting, auto completion, or drag and drop coding.

Highly assistive environments include ample helpful programming features. These environments restrict the programmer to basic sets of programming skills and mostly

use drag and drop functionality rather than syntax-based programming. Further output is animated rather than textual. Another drawback of a highly assistive visual environment is that it prevents novice students from learning underpinning programming concepts, which include syntax checking, compilation and file systems (Dillon et al., 2012). This environment is characterised by non-flexible programming with a lower learning curve exclusively for early stages of learning to program. For novice programmers to further develop their programming skills, they are required to move over to less assistive environments.

Low assistive environments, on the other hand, provide fundamentals for programming, permitting flexibility in the learning process where the output of a program is usually textual. Dillon et al. (2012) noted that command line environments have also been used to teach novices programming due to "familiarity and personal beliefs" (Dillon et al., 2012:70); they are also believed to promote the achievement of useful mental models enhancing the understanding of wider programming basics. However, due to limited features, such environments enforce a higher learning curve, particularly among novice programmers (Dillon et al., 2012). Dillon et al. (2012) demonstrated in their study that students find it difficult to use a low assistive environment (command line) irrespective of their previous experience and confidence with programming. Command line environments are typically used to teach programming at the intermediate and advanced stages of CS curriculum capabilities (Dillon et al., 2012).

Dillon et al. (2012) further recommended that students need to use more moderate assistive environments. Moderate assistive environments provide more assistive programming features and include IDEs and feature-rich editors, which consist of various assistive features. Further, users are rarely restricted to a few capabilities (Dillon et al., 2012). Providing users with flexibility to program can also be used to teach at any stage in CS curriculum as with low assistive environments. Due to the assistive features, moderately assistive environment reduces the programming task, however it also inflicts a higher learning curve for novices because of the many features included (Dillon et al., 2012).

2.8 ROLE OF VISUALISATION IN TOOL SELECTION

This section assists in understanding and providing insight involving criteria to consider when selecting a web-based learning tool to improve understanding of OOP concepts among novices.

The field of "Software Visualisation" (SoftVis) includes both Algorithm Visualisation (AV) and Program Visualisation (PV) tools. SoftVis is aimed at assisting students in the education field. Further, it assists by graphically depicting the organisation of large systems or the progression of software over a period of time, it also assists in finding mistakes or discrepancies. AV and PV are one of the methods to support novice programmers' comprehension (Rößling et al., 2008).

Rößling et al. (2008) explained that AV displays the static and dynamic behaviour of an abstract description of software. The purpose of AV technology is to aid students in comprehending how computer algorithms work through visualising their dynamic behaviour. Current research focuses on empirically assessing the pedagogical effectiveness of AV technologies; thus, the key focus investigates approaches and technology to actively engage students. Hundhausen et al. (2007) agreed that in order to assist students to understand the dynamic behaviour of computer algorithms, pedagogical algorithm visualisations could be utilised.

Brown (n.d.(b)) defines an *algorithm* as a specific sequence of instructions with correct execution results in solving a task. Algorithms are encountered in daily life; however, one does not obviously think in terms of the steps required to solve the task, for example, driving a car or following a recipe (Brown, n.d.(b)). *Algorithmic thinking* involves the skill to understand and execute precise sequential instructions, to evaluate if the algorithm correctly and completely solves the task, and the ability to create algorithms precisely to solve the task (Brown, n.d.(b)). The *significance of algorithms* lies in their ability to precisely and explicitly accomplish a collection of interrelated tasks. Algorithm representation implies that a computerised automated solution exists for tasks (Brown, n.d.(b)).

PV tools, according to Rößling et al. (2008), display the behaviour of a program developed in a particular programming language. This is achieved by showing the effect of specific operations, for example, program state, which is visualised independently so that users can easily comprehend the impact of each operation

(Rößling et al., 2008). Thus, PV tools are used to graphically explain the run-time behaviour of computer programs (Bednarik et al., 2006).

A program visualisation tool is typically developed to be used to support the teaching and learning of programming concepts or to visually debug programs; through graphically depicting parts of a program, this includes a class diagram or animation of the program's execution behaviour. PV systems can be used to support novice programmers in learning programming concepts (Bednarik et al., 2006). Stephen et al. (2012) have further highlighted that PV technology is beneficial in the instruction process used in computer programming. There has been an increase in the development of PV tools, with the objective of assisting in enhancing the instructional process in introductory programming (Stephen et al., 2012).

In the computing and engineering fields of study, programming is an essential subject. PV tools are inevitable to ensure proper understanding of the pedagogical purposes. Stephen et al. (2012) have presented the taxonomy of PV tools with pedagogical focus in order to assist teachers and students in selecting the correct tool for learning/teaching based on their guidelines; the authors have indicated the taxonomy is developed on four classifications and subcategories, which are as follows:

1. Interface category: describes the visual representations (elements and objects) and user interaction with the tool.

    - Visual representation and visual prompts: This sub-category can include any one or more of the following:

        - Primitive Representation: This can be categorised into simple or composite objects and worlds.
        - Standardised Representation: This is universally acceptable visual cues to visualise the program, e.g. UML diagrams or flowcharts. For example, Raptor uses flowcharts.
        - Metaphorised Representation: This approach simulates the behaviour of a particular animal, human or character. For example, the Jeroo tool represents a Kangaroo-like animal.

- User Interaction: refers to PV tools graphical user interface. Sub-classifications within this category are as follows:

    - Predefined: This approach entails following a predefined succession of stages within the PV system. For example, Alice.
    - User-defined: The user has control over this approach, as the visualised program is typically developed by the user. This, for example, could include control over what the user wishes to visualise, the speed of operations, as well as the ability to modify any of the code.

- Enhanced Interactivity

    - The system supports interaction approaches; for example, the system generates questions and the user responds to assess students' understanding.

2. Pedagogy: This category consists of different educational elements that are important to consider in choosing the correct tool.

    - Programming Paradigm: refers to a programming style that a tool supports in order to visualise. These could possibly include the following:

        - Procedural programming language uses mathematical concepts of procedures or functions.
        - Object-oriented programming: This contemporary approach mimics the real-world environment, emphasising objects. Objects contain both actions and data and communicate by passing messages.
        - Logic Programming: utilises mathematical relations and logical inferences.

    - Platform: indicates which operating system the tool can run on/under – it can either be specific or cross-platform. For example, a Jeliot 3 can be supported by multiple platforms and can be downloaded for Linux systems, Windows and Apple systems. A good tool should ideally be cross-platform specific.

- Scope: refers to the accuracy of features of language that the PV tool visualises. It incorporates the following features:

    - Programming Constructs: Ability of the PV tool to support programming fundamentals.
    - OOP Concepts: Emphasis is placed on the tool's support for object-oriented features, which will affect visualisation of OOP languages. These features include inheritance, polymorphism and encapsulation that can be easily understood.
    - Language Support: For specific programming language(s), the PV tool visualises; more specifically, it should reflect the appropriate programming language syntax.
    - Language Compatibility: This is important to ensure ease of transfer to a real programming environment.
    - Multilingual Support: It refers to the ability of the tool to visualise single or multiple language support in converting the code. This feature is beneficial for both novices and advanced learners in learning another language.

3. Stephen et al. (2012) explained that visualisation is closely aligned to interface and refers to an approach in which the system presents the visual and audio indications. These researchers categorised visualisation into the following three sub-categories:

    - Dynamic: refers to controlled or uncontrolled animated visualisations when program code is executed.
    - Static: refers to visual cues fixed on the screen with no animation.
    - Multimedia: does not offer access to the code, and it entails audio (voice) or video and possibly some textual descriptions.

4. Meta-Language

    This category comprises all other independent fundamental elements that are beyond the PV system itself and that can be controllable by the system developers. It consists of the following sub-categories:

- Availability: refers to how the tool can be obtained by the envisioned users, either as open-source software, freeware or as proprietary software.
- Installation: refers to how to install the PV system on a computer for usage and its requirements. Consideration should be made of ease of installation, particularly for novices.
- Extensibility: refers to vital ability of the tool to widen or be customised. Perhaps this can be done by ensuring that the tool is open-source so that the tool's source code is available.
- Integration: ability of a pedagogical system to incorporate or support features of another PV system or IDE, either as an imported file or as a plug-in.

2.9 CONCLUSION

This chapter provided the background and motivation for conducting the research in addressing the importance of the impact of e-learning tools to supplement distance education for novice programmers in addressing difficulties of OOP. DE students are characterised by diversity in knowledge, resulting in a complex learning process. Novices experience difficulty learning and understanding object-oriented design and programming principles and fundamental concepts. These learning difficulties cannot be fully solved by instructors; thus, new procedures and tools can be used to assist diverse students to understand the fundamental OOP concepts (Moritz et al., 2005; Wei et al., 2005).

The tool support relating to OOP pedagogy to address difficulties in OOP among novices was extensively discussed. Initiatives of a few tool supports that have been successfully implemented at various institutions were presented. The purpose of the survey of visual environments focuses on their potential effect on teaching programming and an acceptable, effective tool to consider in order to ease OOP difficulties among novices.

Thereafter, the role of visualisation tools – which addresses why the complexity of the execution of OOP programs seems to require visualisation – was discussed. Further, the criteria to consider in selecting the visualisation tool of this study based on the taxonomy of visualisation tools outlined were explained.

In Chapter 3, the research process will be discussed. Aspects of the research objective, significance of the study and an outline of the research questions of this study will be discussed.

CHAPTER 3
FOCUS

3.1 INTRODUCTION

The preceding chapters emphasised that programming is one of the fundamental courses required in any CS curriculum; however, for most students, it is also one of the biggest challenges. Moreover Esteves et al. (2009) noted the diversity of experience levels in distance education students' knowledge, results in a complex traditional teaching process. Further, due to lack of personalised attention in DE, this ultimately leads to an inadequate learning experience and an increased likelihood of failure (Truong et al., 2005). Clearly, programming is complex for students – particularly evident from the high attrition rates apparent in introductory programming – and remains one of the most urgent challenges (Hundhausen et al., 2006). Georgantaki and Retalis (2007) and Matthews et al. (2009) made the observation that various researchers have conducted studies on effective instructional processes for the OO design and programming, focusing on utilisation of educational tools in helping students to mitigate learning barriers and to conceptualise the OOP philosophy and its concepts (Georgantaki & Retalis, 2007).

This chapter focuses on the research objective, significance of the study and a detailed discussion of the research questions and associated sub-questions of this study regarding the impact of e-learning tools as a support to ease the difficulty of OOP among novices.

3.2 RESEARCH OBJECTIVE

This research is aimed at providing insight into the current emphasis on asynchronous distance education and e-learning tools with the aim to achieve the following:

a) **To assess the influence of e-learning technology tools in enhancing the learning experience of novice OOP students by easing hindrances experienced by these students**

The primary purpose of this study is on a learning approach with effective learning tools as ways to ease students' programming learning difficulties and to address the

high attrition rate evident among novice programmers. The objective is to evaluate novice computing students' perception of the efficiency and effectiveness of the selected visually oriented programming environment in enhancing the students' understanding in learning introductory programming.

b) To determine students' perception of whether the selected tool assists or is beneficial in stimulating the individuals' engagement and motivation with technology

Another objective of this study is to address students' perceptions of whether the selected computing technology supplemented with distance education can stimulate an individual's motivation and development. The goal of the study will be achieved with the support of multimedia technology, which includes information visualisation tools and computer graphics in engaging individuals in order to enhance their learning experience to ease OOP difficulties experienced by novices. Creating an engaging, e-learning experience to learn to program could result in an increase in students' learning confidence and could thus lead to a reduced dropout rate.

c) To determine students perception of whether the tool is beneficial in addressing the diverse needs of students

Minton et al. (2004) highlighted that the primary reason behind the use of e-learning technology is diversity, as e-learning is motivated by learner needs. To address the diverse needs of students, interactive, personalised and adaptive technology is required (Minton et al., 2004). Thus, providing learning opportunities suited to learner interests and needs can lead to an enormous potential to stimulate individuals' motivation and development in creating a more positive learning experience to overcome barriers in programming and enhance concept understanding to address the diverse needs of students in DE.

3.3 SIGNIFICANCE OF THIS STUDY

The significance of combining e-learning tools and distributed education is its potential to transform the way one learns, improve the learning experience, improve quality in education, as well as providing support for lifelong learning. The significance of this research is in providing an effective, appealing e-learning experience by integrating the Raptor tool to enhance learner interactivity,

personalisation and to engage students. In this study, the value of such an environment for learning was investigated, and particular consideration was also made with respect to the impact in creating a more positive, enhanced user learning experience to overcome barriers in programming and to enhance concept understanding.

Mehdi and Feiznia (2011) emphasised that a decisive factor of e-learning is certainly its quality to create a rich learning experience. This study will particularly benefit IT educators about various tools researched as well as encourage them to be become more knowledgeable about the use of the Raptor tool in creating a rich learning experience to overcome OOP hindrances experienced among novices.

According to Mehdi and Feiznia (2011), e-learning has proved to be an effective tool in creating a more accessible, effective and efficient learning experience. In addition, learning that is transformed by technology enables accessibility to online education programmes that are independent of time, pace, place or physical location, and can be customised to students' needs.

3.4 RESEARCH QUESTIONS AND ASSOCIATED SUB-QUESTIONS

3.4.1 MAIN RESEARCH QUESTION

Will the approach of using technology-enhanced e-learning tools improve the student experience by assisting to circumvent pedagogical hindrances experienced by DE students in introductory OOP?

3.4.2 SUB-QUESTIONS

In addition to the main research question, the following sub-questions are worth bringing to the fore:

4. What is an appropriate e-learning tool for teaching OOP?

This question addresses research conducted on what forms of technology-enhanced e-learning tools can be used to supplement the instructional process of OO design and programming. Moreover, the current research indicates the impact of e-learning tools in assisting to create content-rich, engaging and easily accessible applications that are suitable for novices. Furthermore the tools should assist in addressing

novice programmers' understanding and to master the fundamental concepts of object-oriented software development. This sub-question is addressed in Chapter 2.

5. **What criteria are involved in the selection process to define an appropriate tool?**

This question addresses the selection process, through detailing criteria for the selection process. Thereafter, a detailed discussion will be given on why the tool was selected; this will be followed by an explanation of what the Raptor tool is about, which will be addressed in Chapter 4, focusing on tools platform.

6. **What are students' perceptions of the tool?**

The third research sub-question investigates the impact on students' perceptions of the process of integrating the *selected* technology-enhanced tool approach in distance education. This question will determine student perceptions in regard to the impact of a visual e-learning tool support environment in overcoming difficulties in learning. This sub-question will be addressed in Chapters 5 and 6, dealing with object-oriented programming in distance education.

The foregoing questions will address two aspects:

a) Will the selected visually engaging e-learning tool approach that illustrates solutions to algorithm design problems assist novices by motivating them through creating a more positive, enhanced user learning experience with technology? Further, novices' perceptions of the Raptor tool with respect to its usability was determined, engaging user experience and learning capability of tools to support efficient and effective learning in understanding fundamental OOP concepts. These aspects of the research question are discussed in terms of the quantitative results, which includes the statistical analysis for the particular questions in Section 6.4 of Chapter 6.
b) To what extent does the student believe the tool environment has contributed to overcoming difficulties in learning object-oriented programming? These aspects of the research question are discussed in terms of the qualitative open-ended questions results in Section 6.5 of Chapter 6.

3.5 CONCLUSION

The main components of the research are organised into three successive intersecting activities. The initial phase involves investigating and finding various tools to ease novice programmers' understanding of fundamental OOP concepts as discussed in Chapter 2. Thereafter, a tool is selected based on the criteria outlined in Chapter 4. The final phase of the research entails evaluating the subjective view of students with regard to the selected visual algorithmic e-learning tool selected. Additionally, the final phase, discussing the design and methodology for the evaluation, is covered in Chapter 5. This will be followed by results and analysis in Chapter 6.

CHAPTER 4
TOOLS PLATFORM

4.1 INTRODUCTION

The previous chapter discussed the research objective, significance of the study and detailed discussion of the research questions. This chapter now focuses on the selected visualisation tools' platform in creating both a positive programming learning experience and assisting to lower a distance education novice programmer's hindrances when learning to program. The following aspects are covered: why visualisation, criteria specifying the choice of a visualisation system, motivation for selection of Raptor, evaluation of tools, and the Raptor tool's platform.

4.2 WHY VISUALISATION

The focus of distributed learning is on assimilating people and learning resources that are distributed in diverse geographic areas (Li et al., 2008). Zhang et al. (2004) noted aspects that the impact on the effectiveness of e-learning is dependent on media characteristics, the learning environment, technology and learner characteristics. Parker (2003) indicated that various types of e-learning in tertiary education are attributed to the growth and use of the Internet. Due to the Internet, global access to various resources is possible, assisting in the learning process.

Due to the increase in the accessibility of fast, multimedia-capable computers and the radical infiltration of broadband internet, new opportunities can be opened to deliver "rich media content" via the Internet (Wolf, 2012). The content that universities rely on and hope to integrate into a content management strategy has become diverse with the initiation of internet technology, particularly the World Wide Web (Li et al., 2008). The environment in which content is produced and utilised involves a diverse range of different data formats, processing tools and operating systems (Li et al., 2008). This content includes the development of various visualisation techniques to assist in enhancing learning or teaching introductory courses in programming. Vrasidas (2004) suggests that visualisation tools create a positive educational experience that enables learners to visually express and construct meaning. The emergence of visual-based instructional technology provides an enriched form of teaching-learning milieu and can bring about powerful changes to the education system of e-learning in distributed education and the way knowledge

is deployed. A combination of these two technologies can lead to the creation of content-rich, engaging and easily accessible applications that are useful to education.

There are various programming languages, which require a substantial amount of time to learn. Dillon et al. (2012) highlighted that in order to improve novice experience with learning to program, visual environments have been implemented, in contrast to the command line environment. Carlisle et al. (2005) suggest this is possibly due to the highly textual nature of most programming environments as it "works against the learning style of the majority of students" (Carlisle et al., 2005:176). Adopting a visual approach is preferred, as students are more inclined to learn visually through the use of flowcharts and iconic programming languages (Carlisle, 2009). These environments have more assistive features for programming than the command line environment. Novice programming environments have various assistive features including syntax highlighting, auto completion, or drag and drop coding.

Carlisle et al. (2005) expressed the concern that the use of a particular programming language in introductory computing course focuses on syntactic difficulties encountered by novice programmers instead of focussing on the fundamental issues of OOP concepts of classes and algorithmic problem-solving. Programming and visualisation environments have been explored, thus enabling users to develop, visualise and interact with concrete visual representations of computing processes in order to dissociate computing concepts from programming languages (Hundhausen et al., 2007).

Tools should be interactive and engaging to support learning and enhance motivation. Empirical evaluating research on the pedagogical effectiveness of the PV technology identified the more actively and interactive engaged learners were involved in activities involving AV technology, the better they performed and were able to construct own knowledge (Hundhausen et al., 2007). Hence, the aim of this study was to explore approaches and technology that gets students more actively engaged with AV technology and increases their motivation, supports learning and level of interest in algorithms. This research involved exploring e-learning and a visualisation tool support in the distance education environment of UNISA. Visualisation influence on e-learning has resulted in a great potential for administering and distributing learning content online. This study aimed to highlight

that e-learning and visualisation can enhance engagement, accelerate retention, reinforce learning and serve as a catalyst for effective transfer of information to ease the difficulty of OOP among novices.

4.3 TOOL SELECTION PRINCIPLES

This research entailed exploring a variety of alternative visual programming environments for novices in an attempt to lower the barriers to OOP. Thereafter, the study involved selecting an appropriate tool to ease novices' understanding of OOP fundamental concepts. As such, several requirements that would define the ideal choice of the most appropriate tool were outlined. Consideration was given to the following requirements:

1) *Support of a visual algorithm tool representation* to assist novices by shielding some of the complexities of syntax and to address difficulties in OOP through enhancing algorithm problem-solving thinking. Henriksen (2006) emphasised that in order to create an engaging experience to learn about objects, visualisation and interaction techniques are utilised. Giordano and Carlisle (2006) suggest that to enhance learner engagement and pedagogical process, this can be achieved through the use of algorithmic visualisation systems and techniques. Visualisation of an algorithm or program visualisation provides the best method of demonstrating behaviour as well as being an exceptional studying resource for learning algorithms or programming for novices (Moreno et al., 2004). The reason for studying algorithms is that algorithmic thinking is "transferable" – this implies the ability to analyse programs in various domains (Brown, n.d.(b)).
2) *Use of syntax corresponding to Java* with some visualisation system. Both university and industry have placed increasing importance on the early exposure of students to OOP (Georgantaki & Retalis, 2007). There is a demand for software engineers who are proficient in using the OOP paradigm to analyse and develop systems (Georgantaki & Retalis, 2007). Govender and Govender (2012) noted that another reason for OOP inclusion in the curriculum or course is its greater flexibility and shifts from the rigidity of earlier procedural program writing styles. Georgantaki and Retalis (2007) further stressed the growing emphasis placed on newer OOP languages and tools such as Java by the software industry.

3) *Easy* for novice distance education students to use, in order to teach fundamentals in the most straightforward manner so as to prepare students to acquire further high-level programming skills.
4) *Minimalise cognitive overload of students*. Giordano and Carlisle (2006) stated that for a visualisation tool to be of educational benefit, it should be lightweight and less cumbersome.
5) *Availability,* referring to how the tool can be obtained by the students either as open-source software or freeware (Stephen et al., 2012). This was the key criterion that was used in identifying the tools that could be used in teaching object-oriented programming.
6) *Installation*, indicating how the tool will be installed in a computer for practice and its requirements (Stephen et al., 2012). Stephen et al. (2012) highlighted the importance in ensuring ease of installation, as novices will be using the tool in their learning process to understand fundamental complex OOP programming concepts.
7) *Feedback.* Truong et al. (2005) highlighted that when learning to program, it is crucial to receive constructive and corrective feedback.
8) *Tool should cognitively engage students* by actively engaging students and suit the population. Graven and MacKinnon (2009) have noted that students prefer and will opt for an active learning process even after encountering difficulties, as active learning is simply more enjoyable. Furthermore, actively involved students tend to learn more and have more positive learning experiences than passive listeners (Graven and MacKinnon, 2009). According to Zhang et al. (2004), e-learning supports resource-rich, student-centred and interactive learning and thus provides support of opportunities for constructivist learning.
9) *Flexibility with respect to learning processes* in order to meet diverse needs of individuals. Esteves et al. (2009) indicated the diversity of experience levels in distance education students' knowledge results in a complex traditional teaching process. Minton et al. (2004) emphasised that the compelling case for the use of e-learning technology is diversity, as e-learning is motivated by learner needs. To address the diverse needs of students, interactive, personalised and adaptive technology is required (Minton et al., 2004). Thus providing learning opportunities suited to learner interests and needs can lead to an enormous potential to stimulate individuals' motivation and development.

4.4 RAPTOR A SUITABLE CHOICE

Raptor was designed by Carlisle et al. (2005) primarily to minimise barriers to programming, focusing on limiting syntax complexity and offering concrete visual representations on which to work. The core purpose of Raptor was to teach students and enable visualisation in order to assist students in understanding fundamental programming concepts of classes and methods in object-oriented programming and to become better at problem-solving through the use of algorithms (Fletcher & Guillen, 2012). *Problem-solving entails reasoning systematically, to think creatively, logically and abstractly* (Fletcher & Guillen, 2012). *These features of Raptor apply to and satisfy criterion 1 outlined above.*

The goal of Raptor is to design and execute algorithms independent of a programming language. A frequent method used to express a sequence of ordered instructions without using a programming language is *flowcharts* (Brown, n.d.(a)). A flowchart provides a visual depiction of a sequential instruction process. Carlisle (2009) stated that visual representations result in more successful learning of programming concepts when offered using iconic or flowchart methods. Raptor is an iconic programming environment where programs are created visually using UML and flowcharts. This enviroment enables execution and testing of the validity of flowcharts (Carlisle, 2009). Giordano and Carlisle (2006) noted that students are prevented from entering syntactically incorrect flowcharts in the flowchart symbols.

Raptor reduces the complexity for students, as it provides a simple easy-to-use visual environment for OOP algorithm design, development and problem-solving (Carlisle et al., 2005). *Thus, this feature of Raptor applies to and satisfies criterion 3.* This reduces the complexity of writing programs, hence allowing students to focus on the problem to be solved and engaging them more actively in the learning process (Carlisle et al., 2005). *Thus, this feature of Raptor applies to and satisfies criteria 4 and 8.*

The resulting programs can be executed visually within the environment and converted to Java code for novice, intermediate and object-oriented modes. Raptor uses a Java-like syntax (Giordano & Carlisle, 2006). *These features of Raptor apply to and satisfy criterion 2.*

Further, the tool also provides for error message diagnosis recovery, enabling recognition of appropriate error messages that clearly provide information about the error (Brown, n.d.(a)). *These features of Raptor apply to and satisfy criterion 7.*

Giordano and Carlisle (2006:117) emphasised that the advantages of Raptor include "standard symbol template visible to user; constrained symbol placement; input and output handled automatically; real-time syntax checking; flowcharts actually run; one may run a flowchart to completion or step through each symbol discretely; real-time variable inspection at run-time". Moreover, Carlisle et al. (2005) indicated that Raptor is a superior choice, as its code can be developed incrementally instead of enforcing a top-down decomposition approach. Thus, novices are able to create more enhanced programs. Raptor also caters for one- and two-dimensional arrays, files, strings and a more sophisticated graphics library allowing user interaction (Carlisle et al., 2005). The Raptor tool offers flexibility in its development for specific academic purposes, as instructors can customise the environment and facilitate more stimulating implementations by adding to the built-in procedures (Carlisle et al., 2005). *These features of Raptor apply to and satisfy criterion 9.*

The Raptor e-learning system is a *web-based tool* for teaching novice programmers. Truong et al. (2005) highlighted that web-based systems eliminate programming environment difficulties usually experienced by novice programmers. Thus, web-based systems eliminate issues such as installing a compiler or learning how to use a program editor, some of the problems encountered by novice programmers (Truong et al., 2005). Most importantly, the strength of the selected e-tool is that it provides an anytime, anywhere, easy access learning environment for novice students (Truong et al., 2005). Raptor is a free, open-source tool that entirely supports introducing object-oriented programming; it includes the complex features of polymorphism and inheritance (Carlisle, 2009). *These features of Raptor apply to and satisfy criteria 5 and 6.*

The decision to adopt Raptor was also based on several studies that have shown the significance of using Raptor in education. The study by Carlisle et al. (2005) confirmed that students preferred and were more successful at designing algorithms visually when compared to using a traditional language or the writing of flowcharts. Giordano and Carlisle (2006) showed the effectiveness of the Raptor visual programming environment, as the use of flowcharts when developing algorithms

actively engaged students, thus having a positive impact on student learning. *This feature of Raptor applies to and satisfies criterion 8.*

Further, Giordano and Carlisle (2006) noted that in comparison to the Cots tool, Raptor was the preferred tool and was easier for the development of Java applications. Giordano and Carlisle (2006) further indicated that it is perfectly suitable for teaching novices how to design algorithms.

Based on the advantages discussed, it would be suitable to use Raptor.

4.5 PROGRAM VISUALISATION TOOLS EVALUATED

The tools investigated for supporting the instructional process of OOP so as to ease novice programmers' difficulties, as explained in Chapter 2, are evaluated below.

AGUIA/J is a pedagogical tool for interactive experimentation and visualisation of object-oriented Java programs. The strength of AGUIA/J is that it contributes significantly to enhancing motivation and lowering dropout rates (Santos, 2011).

ALICE is a 3D programming animation environment, which includes 3D characters, objects and animations in which objects, classes, methods, events and their behaviours are visualised (Georgantaki & Retalis, 2007; Giordano & Carlisle, 2006). The strength of Alice is in its visual feedback environment, enabling learners to understand essential concepts as well as be actively involved in the learning process (Esteves et al., 2009).

According to Dillion et al. (2012), Alice is cross-platform compatible and has a *positive effect* with respect to performance, retention rate and attitudes. However, Alice is too elementary to be used in a tertiary institution for teaching, as it does not offer a smooth transition to a real programming environment and thus may result in misunderstandings (Stephen et al., 2012). Therefore, even though Alice has been successful in teaching basic programming concepts, learners are required to ultimately move to a more complex programming language (Li & Watson, 2011). Further, with highly assistive visual environments, drag and drop functionality prevents novice students from underpinning programming concepts, which include syntax checking, compilation and file systems (Dillon et al., 2012). These

environments are characterised by non-flexible programming with a lower learning curve proposed for early strategies of learning to program (Dillon et al., 2012). Alice rejects text-based programming even though it may require some syntactic understanding (Giordano & Carlisle, 2006).

Stephen et al. (2012) highlighted that Alice's focus is on introducing OOP concepts using the syntax of Java, C++ and C#. Alice supports teaching introductory programming concepts using an object-based approach (Baldwin, 2007). Additionally, even though it supports encapsulation moderately well, it does not directly support inheritance, and there is no support for polymorphism (Baldwin, 2007).

ALVIS Live! is a pedagogical tool and provides immediate feedback through an algorithm editing and visualisation model to teach novices to program.

According to the study of Hundhausen and Brown (2005), the ALVIS Live! environment results indicated that its “live” editing and visualisation model enabled novices to program significantly faster and accurately as well as resulted in a positive transfer to textual programming interface. This is beneficial for computer science students, as they will eventually have to program in text-based environments. Further constructing *personalised* visual representations that depict solutions to algorithm design problems resulted in increased engagement and enjoyment by students (Hundhausen & Brown, 2005). However, results were inconclusive with respect to the benefit in students understanding algorithms, and the authors noted that students still found it difficult to develop correct programs. They noted, in particular, difficulty with writing array iterative algorithms and constructs, correct loop construction, as well as referencing array elements correctly within those loops through the use of array indexes (Hundhausen et al., 2006; Hundhausen & Brown, 2005).

BlueJ tool is one of the most extensively utilised educational programming environments (www.bluej.org) (Ben-Ari et al., 2002). Among the important features of BlueJ are its simplicity and static visualisation of the class structure as a UML diagram. BlueJ graphically presents OO UML-like diagrams of software project structure; thus, it is an ideal tool to present OOP fundamental concepts such as data abstraction and encapsulation, inheritance and polymorphism, message passing

including others, which are typically challenging for students to comprehend (Stephen et al., 2012).

BlueJ uses an “objects-first” approach to teaching Java. This is grounded on the basis that students study OOP from the beginning in order to avoid a change of programming paradigm (Ben-Ari et al., 2002). This serves as a disadvantage as does not necessarily address other critical aspects of software development involving how to analyse and design a problem (Bednarik et al., 2006). A further disadvantage of this approach is students are required to concurrently master general and OOP-specific concepts as well as paradigm-independent aspects of programming (Ben-Ari et al., 2002). BlueJ does not provide any dynamic visualisation of execution of a program (Moreno et al., 2004). Further, misconceptions can be attributed to the use of BlueJ, as the tool emphasises visualisation of OO concepts and indirectly assumes knowledge of general and basic programming concepts (Ben-Ari et al., 2002). Other disadvantages of BlueJ include that it does not offer true visual semantics; its execution is not completely interactive, nor can it be used directly to find logical errors (Gestwicki, 2004).

Greenfoot is an educational programming environment utilising an engaging and stimulating approach to programming for novices in order to increase retention and to learn the fundamentals of computer programming using the Java programming language (Gallant & Mahmoud, 2008). This educational IDE enables simple interactive graphical applications to be written and through instant graphical feedback, which has proven to be highly motivational. According to Georgantaki and Retalis (2007), the environment is constructed with visual interactions between the world and the objects. Programmers can concentrate fully on programming object behaviour, as the system encapsulates the actual graphics code (Kölling, 2008; Henriksen, 2006). Greenfoot caters for easy entry to the full programming language of Java (Henriksen, 2006). However, Greenfoot only supports the objects-first teaching approach as with BlueJ.

Jeroo is a pedagogical tool to assist novices in offering a lighter introduction to object-oriented programming (Dorn & Sanders, 2003). Jeroo assists novices in mastering fundamental concepts, supporting confidence and interest, and engages students (Sanders & Dorn, 2003). Jeroo’s syntax enables an easy transition to Java,

C++, or C# (Dorn & Sanders, 2003; Stephen et al., 2012). However, Jeroo tool entails a deeper understanding for its efficient use, as it was developed metaphorically from the behaviour of Jeroo, which is a rare Kangaroo-like animal found in the Pacific Islands. This metaphor is limited to users around the Pacific Islands who are familiar with the behaviour of a Jeroo (Stephen et al., 2012).

Jeliot is an interactive open-source program visualisation tool aimed at assisting novice students to learn procedural programming and OOP (Moreno et al., 2004). Jeliot supports objects-first or fundamentals-first approaches in the teaching of introductory programming (Myller & Nuutinen, 2006). However, Jeliot tool is limited only to Java programming OOP language and does not distinguish certain standard Java keywords and functions. Thus, it could result in confusion among weaker students, consequently making it difficult for students to switch to using an ideal Java Integrated Development Environment (IDE) (Stephen et al., 2012). Furthermore, even though Jeliot can visualise the control flow, it does not visualise the roles of variables (Bednarik et al., 2006).

Java Interactive Visualisation Environment (**JIVE**) software tool is an innovative method to runtime graphical notations of OOP (Gestwicki, 2004). Gestwicki (2004) utilised the JIVE prototype and its related visual representations in various courses, ranging from undergraduate to more advanced level. JIVE includes the following three features: visual semantics, interactive execution and effective drawing (Gestwicki, 2004).

Javavis uses a Java Debugging Interface to visualise the state of the program (Moreno et al., 2004). The strength of Javavis lies in its ability to visualise the state of the program and its changes during execution, enabling it to obtain information about the runtime behaviour of the programs (Moreno et al., 2004). This is shown using animated UML-like object and sequence diagrams (Bednarik et al., 2006). However, Javavis is not suitable for novices; it should rather be used in more advanced programming courses, as the visualisations require users to understand UML and the fundamentals of programming (Moreno et al., 2004).

SFC is a structured flowchart editor (Carlisle, 2009). Its emphases are on structured flowchart style development and creating generic or C++ pseudo code (Giordano & Carlisle, 2006). Limitations of SFC are that students can only manipulate flowcharts and symbols, not pseudo code (Giordano & Carlisle, 2006). Further, SFC uses an imperative subset of the C++ code it generates (Carlisle, 2009).

Pedagogic IDE **jGRASP** is beneficial for teaching concepts of object-oriented programming; however, it does not offer true visual semantics or fully interactive execution (Gestwicki, 2004). Further, it uses static visualisations; thus, it is not very interactive (Stephen et al., 2012). JGrasp is not suitable for novice programmers, as it requires programming knowledge, and there is no provision for code highlighting (Stephen et al., 2012).

VILLE is a dynamic program visualisation tool with the key purpose of supporting the learning process of novice programmers. The principal purpose of VILLE is to support a higher level of abstraction focusing on a programming language independency paradigm. Even though VILLE supports all the programming concepts in introductory programming courses, the support for more advanced concepts is limited. Despite, this limitation enables VILLE to define new syntaxes with corresponding features to existing languages (Rajala et al., 2008).

4.6 TOOL SELECTED: ABOUT RAPTOR

The preceding chapters indicated that Raptor runs in the .NET Framework and was developed using Ada, C# and C++ (Carlisle et al., 2005). Raptor is a visual programming environment built with flowcharts (Brown, n.d.(a)). Students can use Raptor to design algorithms by combining basic graphical symbols (Carlisle, Wilson, Humphries & Hadfield, 2004). The syntax used within a Raptor flowchart symbol is designed to be flexible, enforcing syntax checking on each flowchart symbol as it is edited (Carlisle et al., 2004). Thus, students are prevented from entering syntactically incorrect flowcharts (Carlisle et al., 2004). The Raptor flowcharts are enforced to be structured (Carlisle et al., 2005).

Within the Raptor environment, the speed of execution is adjustable; algorithms can either be run step by step or in continuous play mode (Carlisle et al., 2005). As such, programs can be designed and execution traced visually, enabling *visual problem-solving* (Carlisle et al., 2004). Therefore, Raptor's visual development environment enables users to follow the flow of instruction execution, of Raptor programs, one symbol at a time. Algorithms can be executed and compiled within the environment, thus visual debugging of a representation of the algorithm rather than a textual representation (Carlisle, 2009). Further, the state of all variables is displayed in a watch window. Thus, there is no need to use multiple tools (Carlisle, 2009).

Raptor begins by opening UML Designer, which enables the creation of classes, interface, and enumeration types (Carlisle, 2009). A blank workspace with a start and end symbol is visible when opening the Raptor tool (Brown, n.d.(a)) (as displayed in Figure 4.1). Among these symbols are assignment, procedure calls, input, output, selection, loops and return used for OOP explained in further detail in Table 4.1 below (Brown, n.d.(a)).

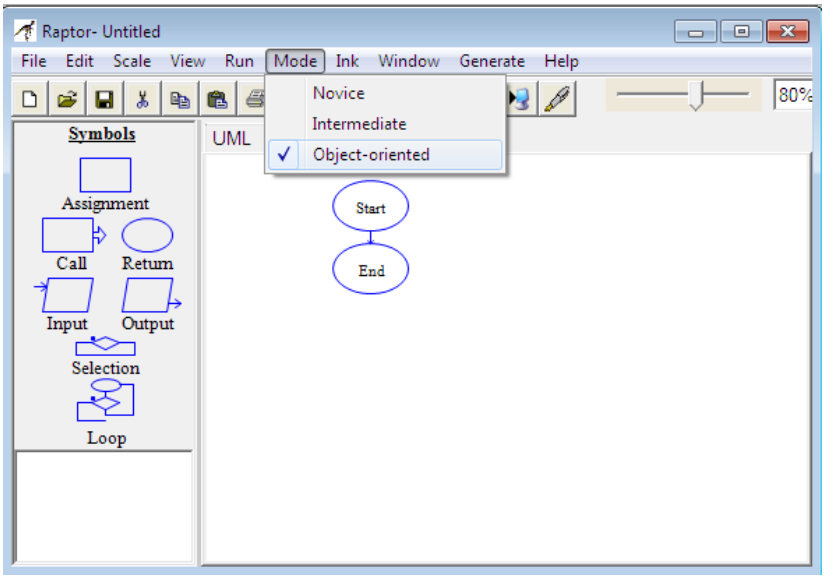


Figure 4.1: Raptor UML Designer

The Raptor environment provides for Java access modifiers of public, private protected or default (Carlisle, 2009). Further, UML allows specification of relationships – a possible relationship includes inheritance (Carlisle, 2009). An example of a hierarchy relationship is created in Figure 4.2. Raptor enables methods and attributes to be added to a class once created. This can be done through the class editor, which provides a method for directly editing the Java syntax of the method, attribute or constructor (Carlisle, 2009). An example of a class editor and method editor are displayed in Figures 4.3 and 4.4. Additionally, GUI tools are also provided to assist and ensure that the correct syntax is applied. Figure 4.5 displays a main program demonstrating the base class Vehicle.

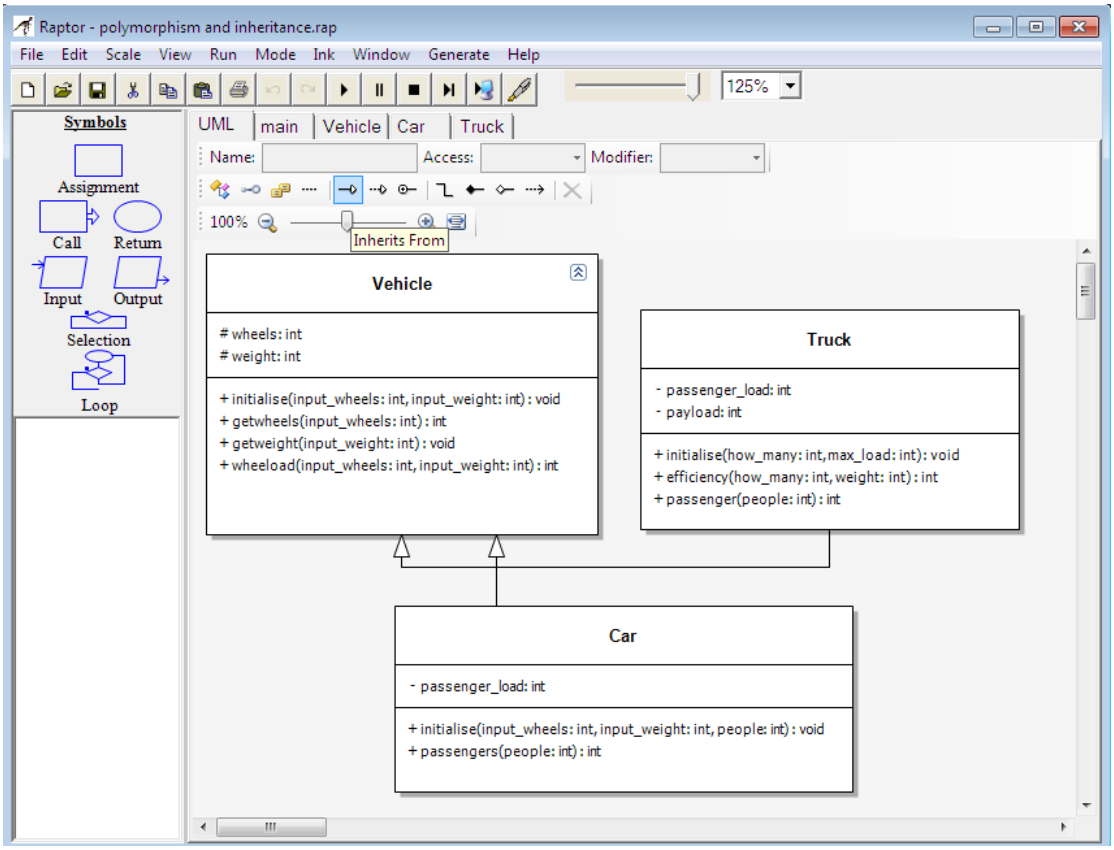


Figure 4.2: Inheritance relationship association between a base class Vehicle and derived classes Car and Truck

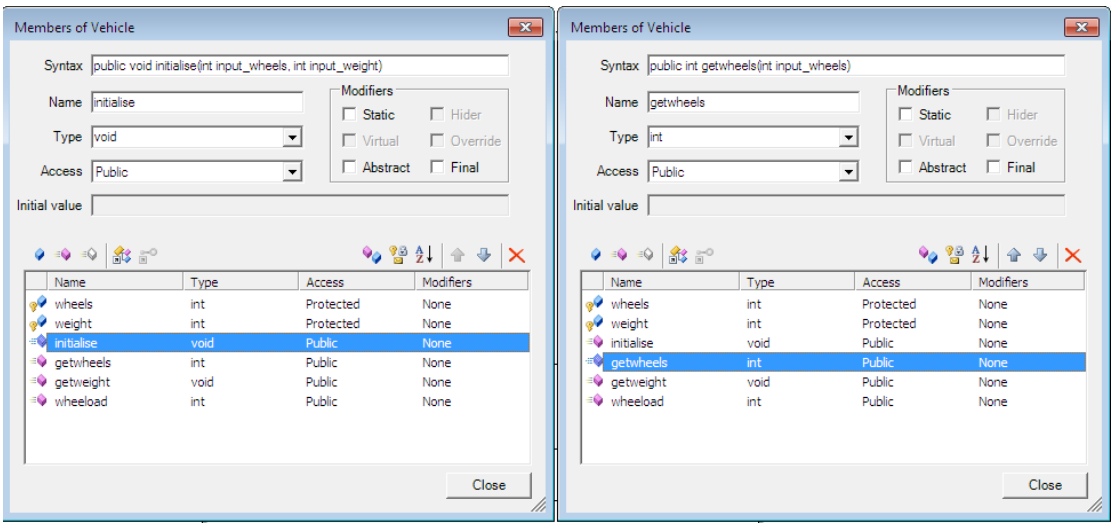


Figure 4.3: Raptor Class Editor – Vehicle Class Method Syntax

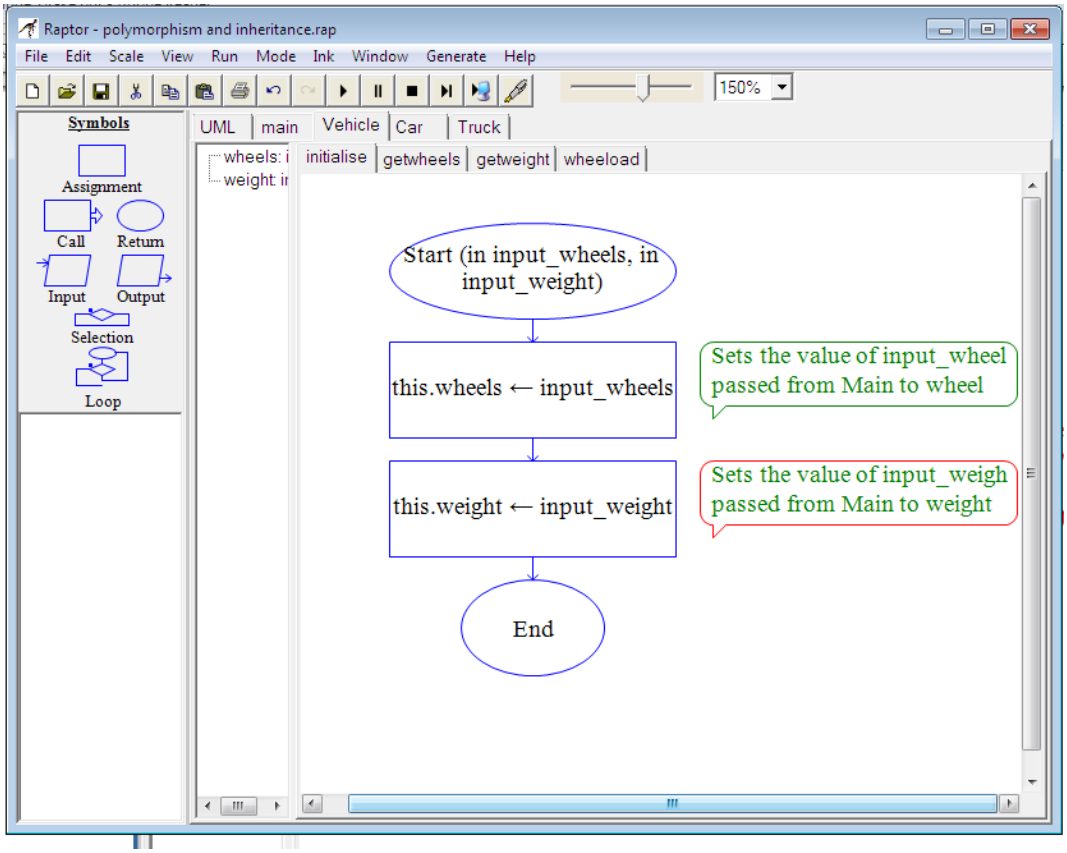


Figure 4.4: Method Editor – Vehicle Class initialise tab to code initialise method

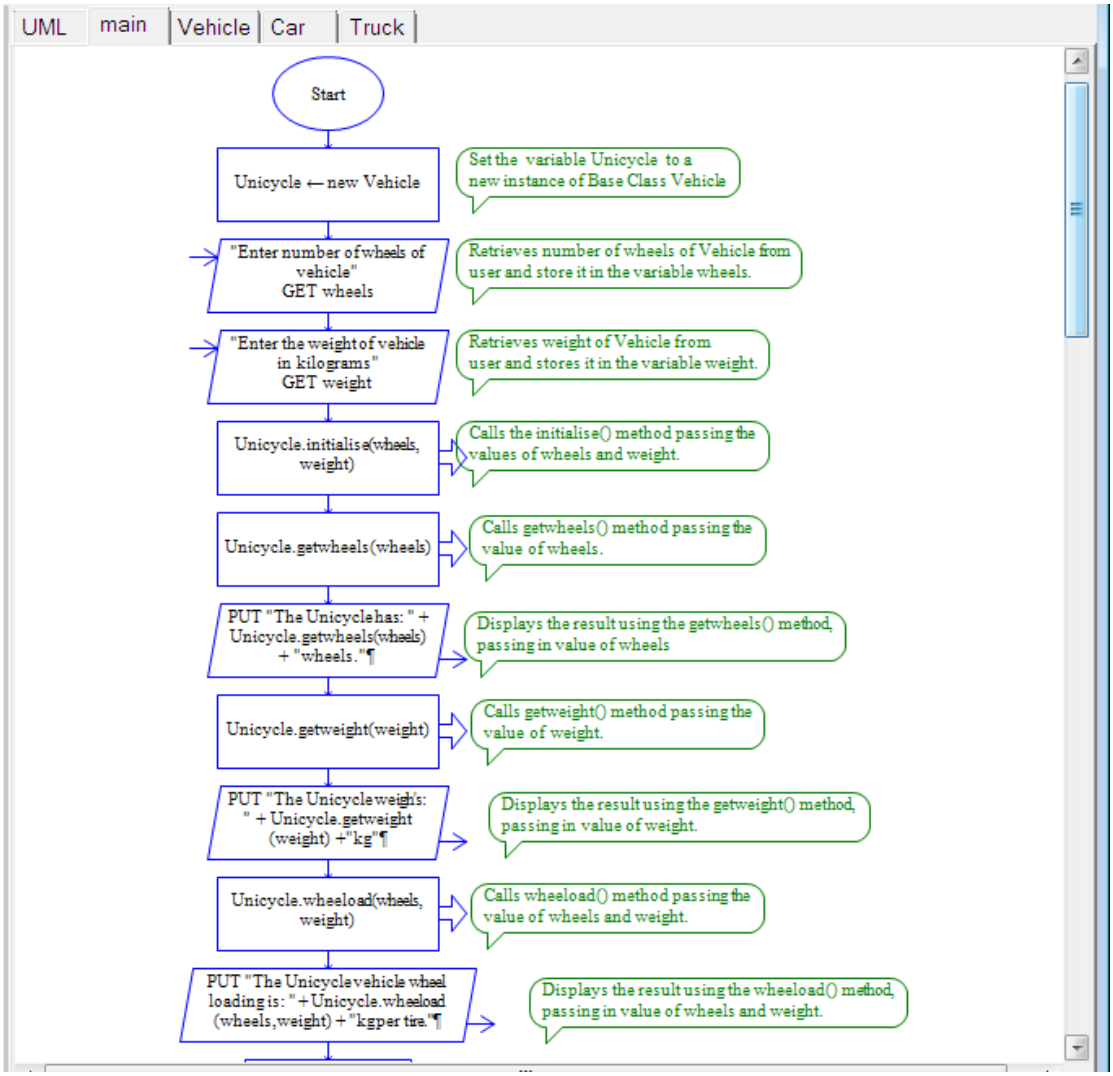
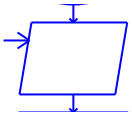
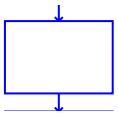
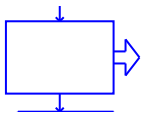
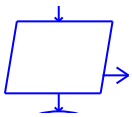


Figure 4.5: Raptor in action – Main program demonstrates base class Vehicle

Raptor enables the transition to Java, through a Java code generator translating the UML Diagram and all of the methods (Carlisle, 2009). Thus, Raptor allows students to create their designs and some of the implementation, and then complete the code in a Java development environment (Carlisle, 2009). However, currently there is no facility for “round tripping”; this implies modifications to the Java code cannot be imported back into the Raptor design (Carlisle, 2009). This facility had no impact on our choosing Raptor, as we required a visualisation tool that used java syntax.

Table 4.1: Raptor symbols

Purpose **Symbol** **Name**	**Description**
Basic symbols	
INPUT  Input statement	Allows user to enter data into a program during program execution and is stored as a *variable*. When defining an input statement, a prompt (text/expression) and variable must be specified. At runtime/execution, an input box will be displayed (Brown, n.d.(a)).
PROCESSING  Assignment statement	Enables manipulation of a variable using mathematical calculations. For definition of assignment statement, the variable and computation/expression performed must be specified. Operations in the expressions are performed based on a predefined order of precedence (Brown, n.d.(a)).
PROCESSING  Procedure call	Enables execution of a collection of instructions defined in the named procedure. Procedure arguments can be changed by the procedure's instructions. When defining procedures, the procedure's name and its arguments need to be specified. When calling a procedure, the arguments need to be matched in data type and order of the defined procedure. Raptor decreases the number of procedure names that one has to recall by displaying procedure names and arguments required as one enters the procedure required. Raptor defines many built-in procedures; these include generating random numbers, performing trigonometric computations and drawing graphics (Brown, n.d.(a)).
OUTPUT  Output statement	Enables one to display or save to a file the value of a variable to Master Console when a program is executed. Output can be specified as text or as an expression (Brown, n.d.(a)).
Programming Control Structures determine the order in which program statements flow and are executed	
1. Sequential	
2. Selection - Selection	Includes single selection or multiple selection control statements (Cascading Selection statements) of current code state of program

control statement (SCS)	data. SCS requires having to make a decision involving alternative paths to consider to the next statement (Brown, n.d.(c)). Decisions are based on the use of relational operators and logical operators to get a Boolean value for the decision. Selections need to be properly nested. Selection is modelled after Java if-else statement (Carlisle, 2009).
3. LOOP - Loop Control statement	Repeats statement/s until a condition becomes true. A loop structure must ensure that it has a single exit point and must be properly nested. It is modelled after Java while loop. The exit condition can have a pretest, mid-test or post-test at any point inside the loop body simply by adding flowchart symbols before and/or after the loop test (Brown, n.d.(c)). Use of loops includes: • Validate user input. • Counter-controlled loop –executes a block of code a specified number of times. • Input loops – enter a series of values to a process, of which there are two techniques, either: o the user enters a "special" value that signifies that the user is finished entering data; or o implement a counter-controlled loop where the user specifies a value in advance. • "Running total loops" or "running sum" calculates the sum of a series of data values. • Counting Loops is used for counting the number of times an event occurs.
Commenting	Used to explain complex program code, enhancing understanding of the way the program works. There are three types of comments commonly used; these are program header, section descriptions and logic description (Brown, n.d.(a)).

4.7 CONCLUSION

This study was conducted with the purpose of evaluating various tools in order to select the most appropriate tool based on the criteria outlined. The primary goal of the selected tool is to ease OOP difficulties experienced by novices through improving student problem skills and minimising syntactic complexity. The key feature is to ensure that the tool is well suited to distance education novice

programmers. A combination of the following features makes Raptor an ideal choice of educational programming environment for the purpose of this study:

a. Designed specifically to help students visualise classes and methods and limit syntactic complexity in writing correct program instructions.
b. Raptor's visual development environment enables users to follow the flow of instruction execution of Raptor programs one symbol at a time.
c. Raptor enables visual representation of the algorithm.
d. Easy-to-use program development environment.
e. Recognition of appropriate error messages that clearly give information about the error.
f. The resulting programs can be executed visually within the environment and converted to Java code for novice, intermediate and object-oriented modes.
g. Raptor is an open-source tool, which could be used in teaching object-oriented programming. Further, Raptor completely supports object-oriented programming, including encapsulation, inheritance and polymorphism.
h. Raptor reduces the complexity of writing programs, hence allowing students to focus on the problem to be solved and engaging them more actively in the learning process (Carlisle et al., 2004).
i. The Raptor e-learning system is a *web-based tool* for teaching novice programmers; as such, it eliminates the programming environment installation difficulties usually experienced by novice programmers. Raptor enables students to execute their algorithms within the environment; thus, there is no need for multiple tools for compiling and executing programs.
j. The Raptor tool offers flexibility in its development for specific academic purposes.

The procedure of how the survey data is presented and interpreted is discussed in Chapter 5 of this research document. The research design and methodology will also be discussed in that chapter. Aspects related to mixed methods will be outlined and will include a framework on analysis.

CHAPTER 5
RESEARCH DESIGN AND METHODOLOGY

5.1 INTRODUCTION

The previous chapter discussed the tools platform of this study. The contribution of this dissertation is on an educational methodology based on the recognition of supplementing distance education prescribed works with a technology-mediated tool. The purpose of this tool is to reinforce and teach fundamental concepts of OOP to address barriers that novices face when learning to program.

The main components of the research design are organised into three successive intersecting activities. The initial phase involves implementation and finding various tools to ease novice programmers' understanding of fundamental OOP concepts as discussed in Chapter 2 (tool support relating to OOP pedagogy to address difficulties in OOP). Thereafter, a tool is selected based on the criteria as outlined in Chapter 4. The final phase of the research entails evaluating the subjective view of students with regard to the selected visual algorithmic e-learning tool selected. An aspect of the final phase, which discusses the design and the methodology for the evaluation, is covered in this chapter.

The research utilised the Raptor web-based, e-learning platform with compatible access to various useful digital content found on websites and delivered via UNISA's Learning Management System. As a result, this would enable students to augment their learning resources, to bridge existing knowledge gaps or to get more suitable explanations. Students registered for the semester module Interactive Programming (ICT2612) in the undergraduate Diploma in Information Technology as well as students studying Introduction to Web Development (ICT1513) were required to complete an electronic survey based on their perceptions and experience of the web-based, e-learning platform with respect to its usability, user experience and the learning capability of the tool and whether the tool contributed to overcoming difficulties in OOP.

According to Creswell and Clark (2011), methodology encompasses the methods – which are the techniques/procedures – used to collect, analyse and understand data. The mixed method approach was used in this study, where use was made of qualitative as well as quantitative methods to collect and analyse data. Creswell and

Clark (2011) provided a definition of mixed methods research based on various perspectives, emphasising the fundamental aspects associated with planning and implementing a mixed methods study. The core features of mixed methods research are listed below.

a) Quantitative and qualitative data is collected and analysed convincingly and meticulously based on research questions.
b) Quantitative and qualitative data can be integrated, by merging the data concurrently, sequentially having one based on the other or embedding data.
c) The research can give precedence emphasising either quantitative or qualitative data, or to both.
d) The researcher can use these processes only within a study, or processes can be implemented in various stages of a research.
e) The philosophical and theoretical foundation of this work fits into mixed method research.
f) Explicit research design plans that guide the research.

Thus, the mixed method methodology can assist in addressing research questions more completely than if only one approach was used. With the acquired knowledge, it is possible to do a subsequent qualitative and quantitative investigation to address research sub-question 3 below.

What are students' perceptions of the tool?

The third research sub-question investigates the impact on student's perception of the process of integrating the *selected* technology-enhanced tool approach in distance education. This question will determine student perceptions of the impact of a visual e-learning tool support environment in overcoming difficulties in learning OOP in distance education.

This chapter focuses on the philosophical assumptions, research design and methodology approach undertaken by the research study so that valid data can be collected and analysed to create meaning.

5.2 PHILOSOPHICAL ASSUMPTIONS: PRAGMATISM

Paradigm is essentially a worldview, which is used to describe assumptions. The pragmatism worldview is coupled with mixed methods, which is utilised in this study. The focus is on the research questions and use of multiple methods of data collection to inform the problems under study (Creswell & Clark, 2011).

According to Creswell and Clark (2011), the pragmatist worldview emphasises:

- Ontology, which refers to the nature of reality when conducting research. In relation to pragmatist views, there are single and multiple realities. For example, a theory exists to explain a phenomenon and assess individual input respectively, thus providing multiple perspectives.
- Epistemology, referring to how people gain knowledge about the relationship between research and a researcher. In relation to pragmatist views, the focus is practically on addressing the research question.
- Axiology, which refers to the role values play in research. In relation to pragmatist views, various perspectives are considered.
- Methodology, referring to research process. In relation to pragmatist views, quantitative and qualitative data is mixed; thus, the approach combines both inductive and deductive reasoning as we are collecting both quantitative and qualitative data and mixing them.
- Rhetoric, which refers to the language of research. In relation to pragmatist views, both formal and informal writing styles are used.

Therefore, the basic characteristics of pragmatism are consequence of actions, problem centred, pluralistic and real-world practice-oriented (Creswell & Clark, 2011).

Pragmatism is suitable for this study, as it provides a guiding framework that brings together both quantitative and qualitative methods, focusing on gaining a better understanding in addressing the research questions in this study. This research will utilise an empirical research design and primary data (new data collected) using an electronic survey to determine learner perceptions and their experience of the Raptor web-based, e-learning platform with respect to its usability, user experience and self-learning capability of the tool. Thus, this provided multiple perspectives relating to the capacity of the tool to ease understanding of OOP concepts among novices.

5.3 RESEARCH DESIGN

Creswell and Clark (2011:53) explained that, research design entails "*procedures* for collecting, analysing, interpreting and reporting data in the research" process. Research design embodies distinctive models, names and procedures for undertaking research. Further, it assists in guiding methods, decisions and processes that researchers undertake in their research study. The research design appropriately addresses the problem and the research questions in a study (Creswell & Clark, 2011).

The approach used in this mixed methods study to assist in the design process includes fixed mixed methods design. This is the case because quantitative and qualitative approaches are determined and designed at the beginning of the research process (Creswell & Clark, 2011).

A typology-based approach to mixed method design highlights the taxonomy of practical mixed method designs adapted to the study's purpose and research questions. Among the reasons Creswell (2009) gives for using a typology-based approach to mixed methods design is that it provides a guiding framework to assist in design selections, thus enabling this researcher to use a concrete method in answering the research questions. Further, it provides the researcher with a structure and reasoning to conduct the implementation of the research methods, hence ensuring the resulting design is thorough, convincing and of value (Creswell & Clark, 2011).

The basic mixed method design typology that best fits this research is convergent parallel design typology. This design adequately matches the research problem, purpose and questions that guide the process. Convergent parallel designs are characteristic of concurrent data collection and analysis of quantitative and qualitative data in the same stage of the research procedure followed by the merging of the two data sets into the final interpretation phase (Creswell & Clark, 2011).

Key decisions involved in mixed methods design typology are as follows (Creswell & Clark, 2011):

a. *Level of interaction between both strands:* This study utilises an independent level of interaction in contrast to an interactive level of

interaction. This implies that quantitative and qualitative strands are distinct in that the research questions, data connection and data analysis are separate; thus, an independent level of interaction is implemented.

b. *Priority of strands:* Quantitative and qualitative methods are both equally important to address the research problem; therefore, there is equal priority of strands in the current study.
c. *Timing of strands:* This research utilises concurrent timing in that both strands are implemented during the same phase of research procedure.
d. *Procedures for mixing of strands:* The primary point of interface at which mixing of quantitative and qualitative strands occurs is in the interpretation final phase of the research process after collection and analysis of both sets of data independently. Here conclusions are derived based on a combination of what was learnt from the quantitative and qualitative strands.
e. *Purpose/rationale for the design:* The rationale for this type of design is that it enables the researcher to gather complementary data, enabling enhanced and a more comprehensive understanding of students' perceptions of the tool and whether they felt it assisted in overcoming difficulties in OOP. Further, as data is collected during the same phase, this makes this design typology very efficient.

Qualitative research, according to Creswell (2009), aims to gather a comprehensive understanding of human behaviour. Qualitative data provides open-ended questions in order to elaborate on perceptions on issues and circumstances affecting them, enabling comprehensive understanding and additional insight into a problem (Creswell & Clark, 2011). It is not as focused as in quantitative research where one uses statistics to analyse the data. Qualitative research was well suited to this study, as it gave this researcher a better understanding of the subjective views of the students.

Apart from the qualitative method adopted for this study, the quantitative approach will constitute the essential method for data collection and analysis. Creswell (2009) highlights the process of measurement as essential and fundamental in quantitative research, as it provides an association between empirical observation and mathematical expression of quantitative relationships. The interpretation of quantitative data provides a solid basis for description and analysis based on measured quantities rather than the subjective view of students; hence,

interpretations can be checked by others for authenticity. According to Creswell and Clark (2011), quantitative data enables a broader understanding of a problem that can be generalised to a larger group. Moreover, tables and charts provide a concise and effective way of consolidating quantitative data and presenting the findings to others (Creswell, 2009). These advantages largely influence why this research uses quantitative analysis.

Therefore, to get a more complete understanding of a research problem, both data should be provided, as the limitation of one can focus on the strength of the other. It is a more natural tendency of individuals to solve problems using both inductive and deductive reasoning in observing and recording behaviour. Both quantitative and qualitative data enlighten and enhance each other (Creswell & Clark, 2011).

This research utilised the data validation variant of convergent parallel design where both qualitative questions with quantitative survey measures were included in the study. The qualitative data is used to validate the quantitative results from the survey item. There is no complete context-based qualitative data set, as qualitative questions are added on to a quantitative instrument. However, they provide emergent themes and interesting quotes enabling the validation and enhancing of quantitative survey findings (Creswell, 2009).

The flowchart of procedures in implementing convergent design is discussed below and depicted in Figure 5.1. The quantitative process is shown on the left and the qualitative process on the right (Creswell & Clark, 2011).

The steps to be followed in implementing convergent design are as follows:

Step 1: Concurrent, separate data collection of quantitative and qualitative data that is independent of each other and of equal importance.
Step 2: Analysis – separate and independent analysis using quantitative and qualitative analytical procedure.
Step 3: Point of interface to merge results – compare/transform to relate data.
Step 4: Relate and combine two databases for better understanding of research problem.

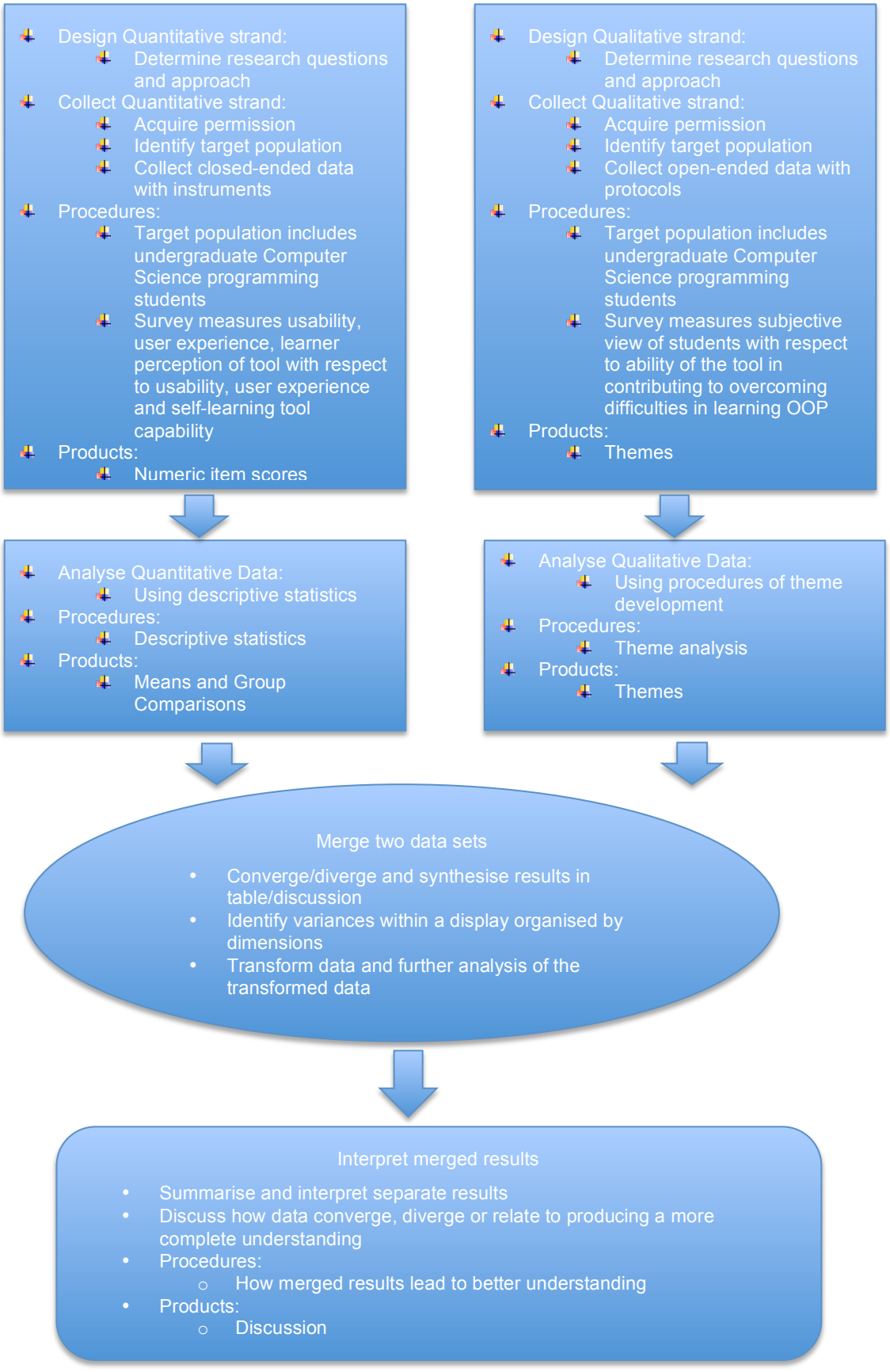


Figure 5.1: The flowchart of procedures in implementing convergent design

The challenges and limitations of convergent design typology of this study included:

- Challenges in merging two data sets and determining if the results are meaningful. Thus, it was necessary to ensure that the quantitative and qualitative data addresses the same concepts of the tool in enhancing understanding of OOP concepts to lower learning programming barriers experienced by novices.
- If results do not converge, the typology could provide further insight into the research, thus requiring additional data collection and re-analysis.

5.4 METHODOLOGICAL APPROACH

The methodology covers the following subsections:

- target application and sampling strategy
- data instruments
- validation of data instruments
- administration of tools for data collection
- data collection
- data analysis
- data presentation and interpretation procedures
- ethical considerations

5.4.1 TARGET APPLICATION AND SAMPLING STRATEGY

The sampling strategy utilised in this study is a non-probabilistic sample, which includes individuals chosen based on availability, recognising that this target population is not representative of the population of all students enrolled for these modules at UNISA. Undergraduate CS students' characteristic of being able to apply OOP and design skills was required for participation in the study. The target population constituted of students registered for the semester module Interactive Programming (ICT2612) as well students studying Web Development (ICT1513) in the undergraduate Diploma in Information Technology in 2014 and Interactive Programming (ICT2612) in 2015 at UNISA

The module ICT2612 is one of the second-level modules in Diploma in Information Technology and focuses on object-oriented programming principles. The purpose of the course is to equip students so that they can be introduced to designing and building mobile application using the Android open-source platform. In this course, students will utilise Java programming language and will work within Eclipse IDE (UNISA, 2012).

The module ICT1513 is a semester module presented online in the Diploma in Information Technology curriculum. The focus of this module is to enable a novice web developer to evaluate, build and maintain web pages. The purpose of this module includes presenting critical arguments around information and communications technologies (ICTs) for development, considering ethical dimensions within an information society in relating to copyright and intellectual property rights, privacy, conduct and expectations. Moreover, novice students will be able to express their ability to maintain efficient, organised and secure electronic working environments through management of digital files, systems and application software and abilities related to engaging with textual and numerical data to present information in various formats (UNISA, 2012).

Table 5.1: Sample of population, with students and number of responses

Course	**Number students contacted**	**Responses returned**
Interactive Programming (ICT2612) – semester 2 of 2014	250 students	18
Interactive Programming (ICT2612) – semester 1 2015	519 students	19
Introduction to Web Design – semester 2 of 2014	216 students	3

Table 5.1 indicates the sample of population, number of students contacted/registered for the modules for that year of study and number of responses returned. The research population consisted of 985 students registered over the two years (2014 and 2015). A total of 40 (approximately 4% of the population) responded. Due to a poor response rate contributing to a small sample size, this resulted in low statistical power. Consequently, the scope for generalisation purposes may be limited, as a small sample was used.

5.4.2 DATA INSTRUMENTS

For the purpose of this study, structured, electronic questionnaires as a source of data collection were utilised. A survey is a research method utilising questionnaires and/or statistical surveys to collect data about people, their thoughts and behaviours (Creswell, 2009). A specially designed survey available at http://goo.gl/lWPSTv (attached in Appendix B) was distributed to the target population for completion. The questionnaire was meant to assist in determining student perceptions on the impact of an e-learning tool support environment to overcome difficulties in learning OOP in distance education.

The electronic questionnaires that were utilised served as an advantage, as respondents could be contacted over a long distance; it was cost-effective with respect to time, money and travelling; and confidentiality was maintained. Information was then gathered about the student's interaction with the system and feedback about the e-learning platform with respect to the student's perceived usability of the tool, user experience, learning perception of the tool regarding self-learning tool capability and lastly if students felt the tool contributed to overcoming difficulties in OOP.

In this study, emphasis was placed on closed-ended questions, since they are easier to analyse than open-ended questions and because they are subject to statistical treatment and analysis. With a closed-ended question, the respondents pick an answer from a given number of options (Olivier, 2009). A Likert-type scale will be used in this study. According to Creswell (2009), a Likert-type scale is a psychometric scale used predominantly in questionnaires where participants specify the extent to which they agree or disagree based on an ordinal scale for a sequence of statements, so the scale captures the strength of their experience. In this study, a 5-point Likert scale technique was used to formulate closed-ended questions ranging from strongly agree to strongly disagree. The qualitative questions were appropriate in addressing the subjective opinions of students. Open-ended questions asked the respondent to formulate their personal opinions (Olivier, 2009).

The survey consisted of three sections. The first section is the background, where respondents were asked about their biographic information. The second section was designed in order to evaluate both the experience with and the attitude towards Raptor. In the third and last section, the survey asked the respondents to express

their opinion about whether the tool contributed to overcoming difficulties experienced in OOP.

The survey table in Appendix B indicates groups, subgroups, survey questions, as well as further elaboration on quantitative and qualitative questions addressed in the survey. Groups and subgroups were adapted from the research of various authors (Bouvier, Chen, Lewandowski, McCartney, Sanders, & VanDeGrift, 2012; Jourjon, Kanhere & Yao, 2011; Ssemugabi & De Villiers, 2007).

The quantitative questions examined the extent to which novice computing students evaluate the following key variables:

- The user interface factors that include perceived usability of the Raptor tool with respect to its effectiveness, efficiency, learnability, and error-recognition-diagnosis-recovery cycle.
- The user experience with respect to affect, visualisation appearance, and learner motivation and interactivity.
- The learning perception with respect to learner control and learning capability of the web-based tool that was utilised.

The qualitative questions examined the students' subjective evaluation of Raptor regarding the perception of whether the tool environment contributed to overcoming difficulties in OOP as well as in determining any previous object-oriented programming experience details.

5.4.3 VALIDATION OF DATA INSTRUMENTS

Before a questionnaire can be used for data collection, the researcher has to make certain that it is error-free by pretesting. The pretest was conducted by a lecturer and module leader – Ronell van der Merwe – who reviewed the items of the questionnaire for clarity, layout, readability and understandability to be able to identify errors in the survey so that it could be adapted if any problems were encountered. It was also ensured that the instrument has an easy layout and that it is not too long in order to make it easy for novices to respond to it. No pilot study was conducted, as a low response rate was anticipated and this researcher did not want to lose out on possible respondents because the pilot study would impact negatively on the research in terms of possible respondents.

5.4.4 ADMINISTRATION OF TOOLS FOR DATA COLLECTION

This researcher wished to support the experience of OOP students by presenting concepts in a manner which makes them more intuitive and easier to understand through a specifically designed tutorial that is attached in Appendix I. The approach of this researcher was to reinforce and enhance the learning of programming concepts through the use of the Raptor platform. The Raptor tutorial task of polymorphism and inheritance allows students to graphically experiment with these and related programming concepts. This researcher believes that the use of the visualisation tool Raptor with important graphical visualisation can provide a profound effect towards effective learning in easing the difficulty of OOP among novice programmers.

The Raptor Tutorial Work Plan in Appendix H was distributed to students via email and through the UNISA Learning Management System (LMS) during semester 2 of 2014 and semester 1 of 2015. The aim in this study was to express programming concepts to novice undergraduate students through a succession of clearly designed steps in tutorials as described in appendix H so that students can understand and reinforce programming concepts by completing the tasks.

5.4.5 DATA COLLECTION

The design in this study involves prioritising quantitative and qualitative information equally. Independent quantitative and qualitative data sets on the same questionnaire (form) from the same source were collected. Parallel questions for both quantitative and qualitative data collection were created. This implies the same set of individuals participating and that the same concepts are addressed in both data collection sets. Thus, the database can be merged so that there exist both quantitative and qualitative questions addressing the novice's perception of the tool in order to overcome difficulties in OOP.

This researcher developed their own instrument and an online questionnaire developed using Google forms. Students were provided with a URL (http://goo.gl/lWPSTv) in order to link to the survey. This researcher, as the originator of the questionnaire, was able to receive the feedback from students in a spreadsheet format. Both quantitative and qualitative data was collected from computer-based methods of Google forms in an Excel spreadsheet. These were

carefully recorded and placed in secure electronic files available in the results section.

5.4.6 DATA ANALYSIS

Creswell (2009) explained that data analysis is the process of examining, cleaning, transforming and modelling data. The objective of analysing data is to highlight valuable information, derive conclusions and support decision-making. Specifically, the study provided answers to research sub-question 3. In analysing data, this study employs both quantitative and qualitative methods (mixed methods), but the emphasis was placed on the former.

Descriptive statistics are described, as the main features of analysis of data in quantitative methods (Creswell, 2009). The aim of descriptive statistics is to quantify a summarised data set presented with more formal analysis (Creswell, 2009). This study used descriptive research in the analysis process to address quantitative questions.

On the other hand, qualitative responses based on subjective views of the students were categorised, and themes were generated from the open-ended comments so that key aspects could be identified. With that said, inferences were made in this study. Inference, according to Creswell (2009), is to draw conclusions by deductive reasoning from facts. The qualitative analysis of the research provided potential benefit of the analysis of the impact e-learning platform tools – as seen by students – have in addressing difficulties of OOP.

For the purpose of this study, the value of the mixed method responses from the subjective view of open-ended questions were mainly used to triangulate with the questionnaire closed-ended questions in order to ascertain the accuracy and validity of the responses and to allow for a more complete, comprehensive analysis of the research situation. It enabled this researcher to get the whole picture (improve integrity) of the potential benefit of the impact of an e-learning platform as seen by students in terms of usability and the self-learning capability of tools to overcome difficulties in OOP.

5.4.7 DATA PRESENTATION AND INTERPRETATION PROCEDURES

To present the data collected, this researcher made use of biographical data display tables and figures and provided interpretation after each table and figure. Further discussion is arranged according to categorised survey questions, and interpretations follow the table or figure.

5.5 ETHICAL CONSIDERATIONS

According to Creswell and Clark (2011), ethical issues entail consent to provide data, controlling sensitive information, and revealing the purpose of research for both quantitative and qualitative data. In relation to the UNISA expectations, policies and procedures have been followed by having a legitimate supervisor and adherence to supervisor-supervisee ethics. Standardised procedures for collecting data on instruments permissions and review of UNISA ethics board approvals are attached in Appendix (A) in order to ensure rights of individuals participating in the research are protected and to assess any risk or harm that could be involved. Letters of consent to conduct the research study were obtained from the Executive Director of Research Department (Appendix A).

An informed consent form (Appendix G) was sent with a Raptor Tutorial Work Plan (Appendix H), where students were made aware of the purpose of their participation in the survey, that their participation was voluntary and were assured of anonymity and confidentiality of their information. A general research route was followed and ethics observed.

5.6 CONCLUSION

Chapter 5 discusses the mixed method approach utilised in the study with qualitative and quantitative characteristics. The chapter describes how the population and sample were picked, the use of a systematic sampling technique, and the tools for data collection and how they were administered and conducted. Furthermore, it describes how validity of data collection was ensured and how data was presented and analysed in terms of survey questions. The chapter concludes with ethical considerations undertaken in the study.

CHAPTER 6
RESULTS AND DISCUSSION

6.1 INTRODUCTION

Technological influences in recent years have been characterised by an increase in the accessibility and usability of information, educational and communication technology tools. This study's aim was to evaluate students' perceptions of the Raptor e-learning tools with respect to the perceived effectiveness of the usability of the tool, the quality of the user experience and the perceptions of the quality of self-learning capability of the tool in enhancing novices' learning experience. Moreover, students were evaluated on the basis of whether the tool assisted in overcoming difficulties experienced in OOP.

To assess the tool support, an online survey was developed. Students were required to complete tutorial tasks at their own pace asynchronously as discussed fully in the previous chapters. At the completion of the tutorial, the survey was administered to novices. The survey contained three sections (Appendix B). The first background section asked the respondents about their biographic and course information. The second section was designed to evaluate the user interface, user experience and learning perception of the Raptor tool (interactive visualisation systems) based on closed-ended questions. In the third and last section, the survey asked the respondents to express their opinion about their experience with and the attitude towards the Raptor tool.

The preceding chapter discussed the research design and methodology employed in this study. This chapter presents the data collected and the analysis. The processes include quantitative and qualitative frameworks within them. Quantitative and qualitative data was gathered from the concurrent questionnaires, which were administered to the computing students. The research question findings are discussed in terms of the following: quantitative results, which involve the statistical analysis for the particular questions, and qualitative results of the open-ended questions. Thereafter, final conclusions were drawn.

The survey table in Appendix B indicates groups, subgroups, survey questions, and further elaboration on the questions addressed in the survey. Groups and subgroups

were adapted from the research of Bouvier et al. (2012), Jourjon et al. (2011), and Ssemugabi & De Villiers (2007).

This chapter is divided into four major sections: introduction, biographical information, quantitative research questions, qualitative research questions and the chapter conclusion.

6.2 BIOGRAPHICAL INFORMATION

ICT students registered for the undergraduate semester module of Interactive Programming (ICT2612) and Introduction to Web Design (ICT1513) students at UNISA were selected for inclusion in the study.

6.2.1 AGE OF RESPONDENTS

Figure 6.1 indicates the age group of students who participated in the research study.

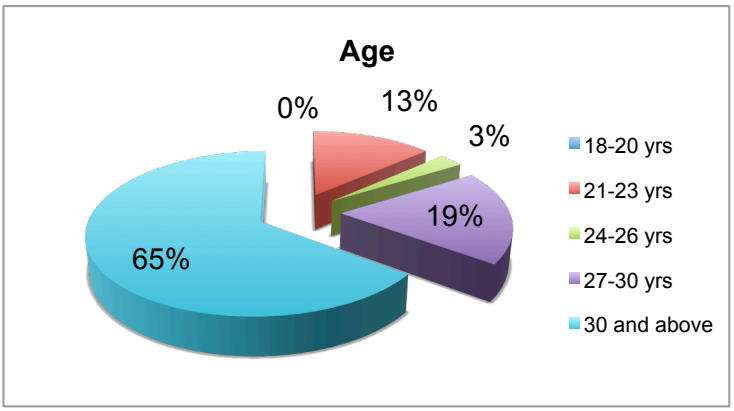


Figure 6.1: Age of respondents

From a total of 40 students that participated, the majority (65%) of the respondents were in the age group of above 30 years. Moreover, as shown in Figure 6.1, there were no respondents below the age of 20 years.

6.2.2 GENDER OF RESPONDENTS

Figure 6.2 illustrates the percentage of male and female respondents.

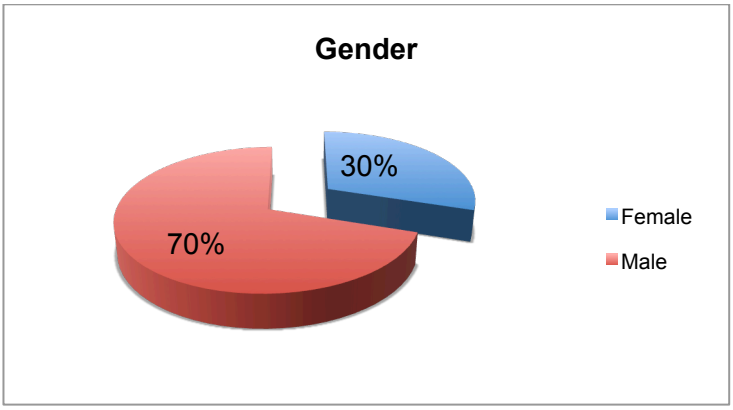


Figure 6.2: Genders of respondents

There were both male and female students who responded to the questionnaires. There were more male respondents than female respondents. Only completed questionnaires were used for the purpose of analysis.

6.2.3 COURSE DISTRIBUTION

Figure 6.3 shows the course distribution of students.

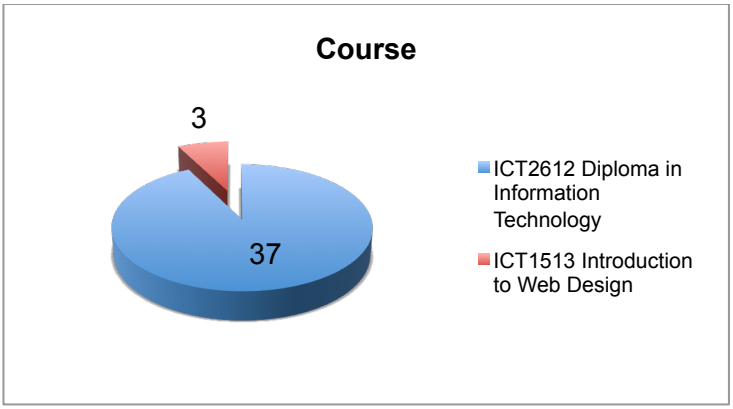


Figure 6.3: Course distributions

Table 6.1: Course distributions

Course code:	Course name:	Participants
ICT2612	Diploma in Information Technology	37
ICT1513	Introduction to Web Design	3

The data collected and presented in Table 6.1 indicates that from a total of 40 respondents, 37 – making up the majority – were registered for course code ICT2612 Diploma in Information Technology. The reason for this number is that the participants were selected for inclusion in the data during 2014 and 2015.

6.2.4 YEAR OF STUDY DISTRIBUTION

Figure 6.4 depicts the distribution of students with respect to the year of study.

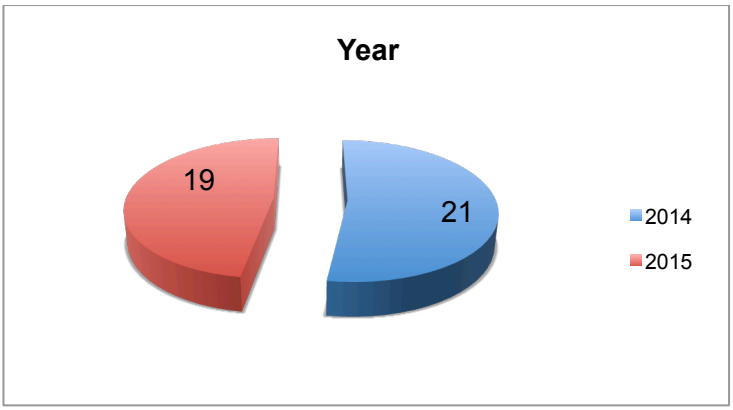


Figure 6.4: Year of study distributions

The data collected and presented in Figure 6.4 indicates that from a total of 40 respondents, 21 responded in 2014, and 19 responded in 2015. In 2014, respondents were registered for both courses, ICT2612 and ICT1513, whereas in 2015, only respondents who registered for ICT2612 were selected for inclusion in the study. Due to the poor response rate in 2014, possibly attributed to involvement in activities and preparation for exams, the survey was administered again at the beginning of the term during the 2015 semester.

6.2.5 OCCUPATION DISTRIBUTION

Table 6.2: Occupation

Occupation	Participants	Percentage
Information Technology	18	45%
Other	12	30%
Unemployed	4	10%
Student	6	15%

The data collected and presented in Table 6.2 indicates that from the total of 40 respondents, predominantly 18 (45%) of the respondents were employed in information technology. Furthermore, the least number of respondents (four) were unemployed, which equates to 10%.

6.3 DISCUSSION OF RESEARCH SUB-QUESTION 3

In this section, research sub-question 3 as outlined below will be investigated.
What are students' perceptions of the tool?

The third research sub-question investigates the students' perceptions of the process of integrating the selected technology-enhanced tool approach in distance education. This question will determine student perceptions regarding the impact of a visual e-learning tool support environment in overcoming difficulties in learning OOP in distance education.

The third sub-question addresses two aspects:

a) Will the selected visually engaging e-learning tool approach that illustrates solutions to algorithm design problems assist novices by motivating them through creating a more positive, enhanced user learning experience with technology? Further, a determination was made of novices' perceptions of Raptor with respect to its usability, engaging user experience and learning capability of the tool to support efficient and effective learning in understanding fundamental OOP concepts. These aspects of the research question are discussed in terms of the quantitative results, which include the statistical analysis for the particular questions in section 6.4.
b) To what extent does the student believe the tool environment has contributed to overcoming difficulties in learning OOP? These aspects of the research question are discussed in terms of the qualitative open-ended questions results in section 6.5.

6.4 QUANTITATIVE RESULTS

In this section, the quantitative survey results and statistical analysis of the closed-ended survey questions based on the first aspect in section 6.3 are discussed. Quantitative data was interpreted using simple descriptive statistics to consolidate and explain data. Data has been captured and portrayed in tables and, graphically, using chart diagrams. Frequencies were worked out to represent data.

The survey consisted of 30 quantitative statements. In this study, the 5-point Likert scale technique to formulate closed-ended questions ranging from strongly agree to strongly disagree (1 – Strongly Agree, 2 – Agree, 3 – Neutral, 4 – Disagree, 5 – Strongly Disagree) was used.

The quantitative statements examined the extent to which novice computing students evaluate the following three categories (key variables):

- The user interface factors that include perceived usability of the Raptor tool with respect to its effectiveness, efficiency, learnability, and error-recognition-diagnosis-recovery cycle.
- The user experience with respect to affect, visualisation appearance, and learner motivation and interactivity.
- The learning perception with respect to learner control and learning capability of the web-based tool that was utilised as an effective e-learning tool to ease the difficulty of OOP.

Table 6.3 displays the quantitative statements evaluation categories.

Table 6.3: Quantitative statement evaluation categories

Category	Subcategory	Description
"User Interface factor": - Category 1: Perceived Usability of tool	Effectiveness	Involves general goal capturing. Interface allows users to do what the task entails (Bouvier et al., 2012).
	Efficiency	This is defined as resources used in relation to the correctness and comprehensiveness with which users achieve objectives (ISO 1998).
	Learnability	User ability to use the tool interface correctly in order to accomplish a task (Bouvier et al., 2012).
	Error-recognition-diagnosis-recovery cycle and support information	Interface recovers from errors and undesirable conditions (Bouvier et al., 2012)
"User Interface factor": - Category 2: User Experience	Affect	User's perception of the interface quality (Bouvier et al., 2012).
	Visual appearance	Relates to general user experience (Bouvier et al., 2012).
	Learner motivation and interactivity	Refers to content and interactive features (Bouvier et al., 2012).
Category 3: "Learning Perception"	Learner control	Learner control with respect to time, place, content and sequence of learning (Ssemugabi & De Villiers, 2007).
	Learning capability of tool	As an effective teaching capability of tool refers to both concept understanding and self-learning tool capability (Jourjon et al., 2011).

6.4.1 CATEGORY 1: PERCEIVED USABILITY OF TOOL

Category 1 refers to the perceived usability of the tool part of the user interface factor. This study investigated the following subcategories: effectiveness, efficiency, learnability and error recognition of the perceived usability of the tool as illustrated in Figure 6.5. Each of the subcategories are discussed with respect to the survey questions.

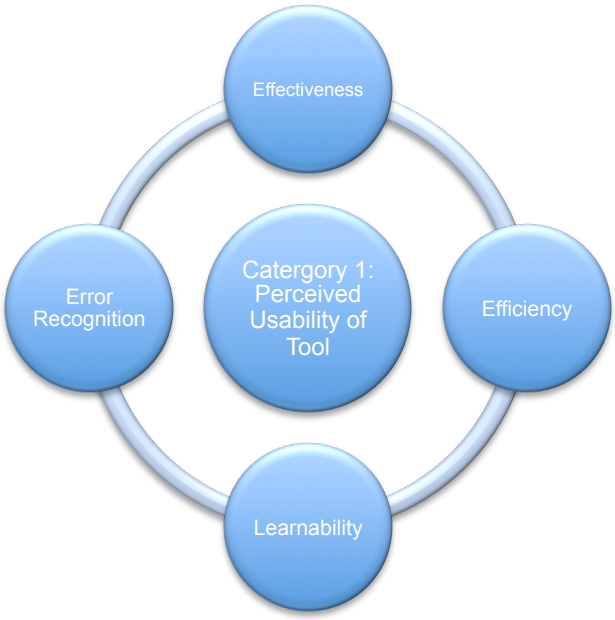


Figure 6.5: Category 1 subcategories

6.4.1.1 Effectiveness

According to the International Organization for Standardization (ISO), effectiveness is defined as the correctness and comprehensiveness with which users accomplish specified objectives (ISO, 1998). Questions 1, 2 and 3 dealt directly with effectiveness. Figure 6.6 indicates that students perceived Raptor to be positive with respect to its effectiveness.

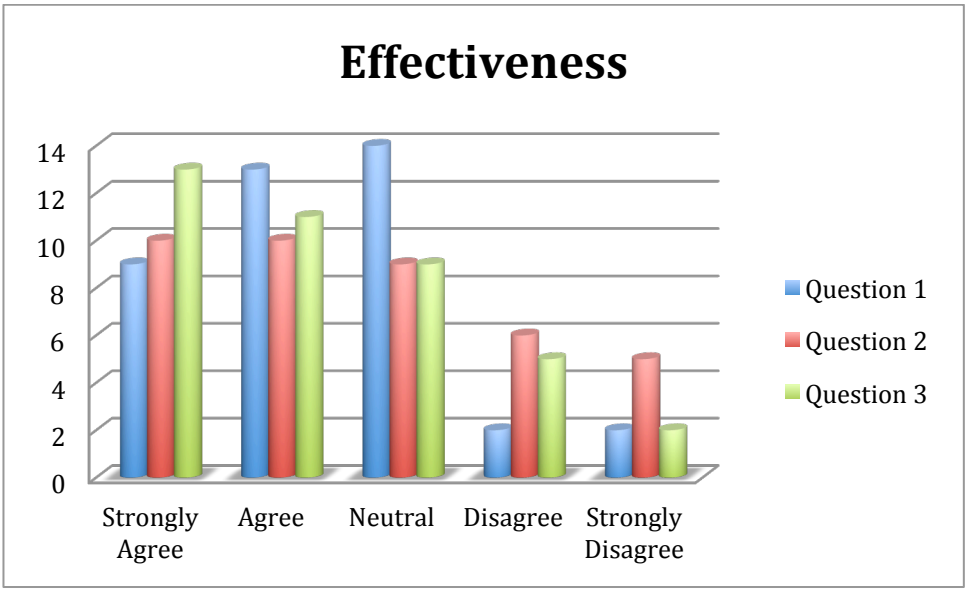


Figure 6.6: Effectiveness distribution

Statement 1: Using the Raptor tool in my studying would enable me to accomplish program tasks more quickly and efficiently.

Figure 6.6 illustrates that students' perceptions of the effectiveness of the Raptor tool in DE learning process were perceived as moderately positive. The students predominantly indicated 'neutral' (14) responses followed by 'agree' (13) and 'strongly agree' (9) regarding Raptor's ability to assist in increasing productivity and effectiveness in their studying. Further, only two of the students responded 'disagree' and another two responded 'strongly disagree' to this statement.

Statement 2: I think the tool would be useful for novices and more advanced users.

Figure 6.6 highlights that students' responses were positive with respect to the flexibility of the tool in its ability to accommodate different user levels, from novice to more advanced user level students. The students mostly indicated 'strongly agree' (10) and 'agree' (10) responses. Additionally, it was noted that six indicated 'disagree' and five indicated 'strongly disagree' to this statement.

Statement 3: The tool has a simple and easy-to-use navigation structure.

With respect to the simple and easy-to-use navigation structure of the tool, students' responses were positive, as indicated in Figure 6.6. The students predominantly indicated 'strongly agree' responses (13) followed by 'agree' responses (11). This statement implied that the tool required minimal user control actions to accomplish the tasks. Moreover, it was flexible to interact with, enabling shortcuts to accelerate completing the task.

6.4.1.2 Efficiency

According to ISO (1998), efficiency is described as resources used in relation to the correctness and comprehensiveness with which users achieve objectives. Statements 4, 5, 6 and 7 dealt directly with effectiveness. Figure 6.7 shows that students perceived Raptor to be positive with respect to its efficiency.

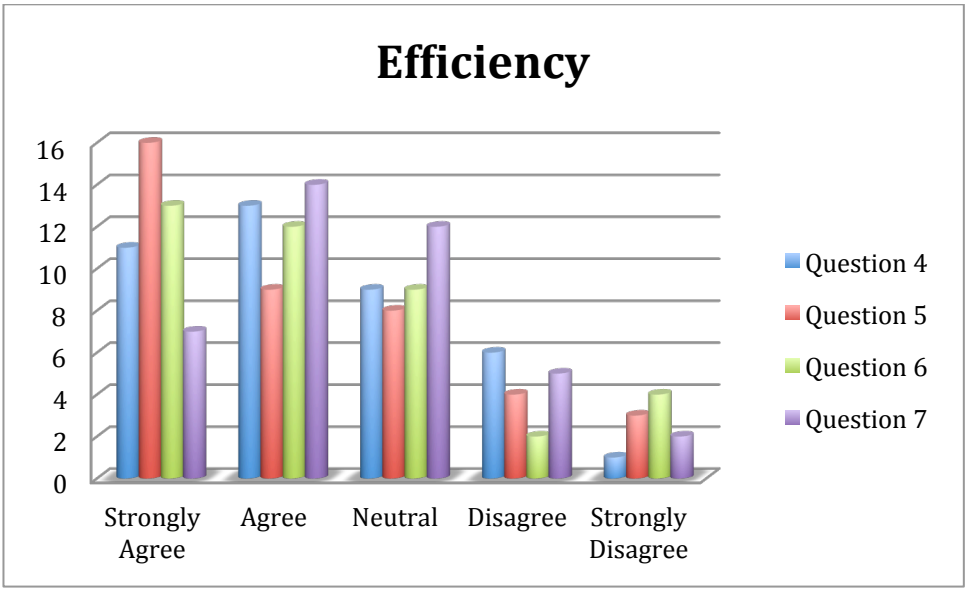


Figure 6.7: Efficiency distribution

Statement 4: Using the Raptor tool in my studying would enable me to accomplish program tasks more easily and effectively.

Students felt confident towards using the Raptor tool in their studying, which enabled them to accomplish program tasks more efficiently and effectively. Figure 6.7 reflects predominantly ‘agree’ (13) followed by ‘strongly agree’ (11) responses.

Statement 5: It was easy to use the tool to create flowcharts.

Figure 6.7 highlights the ease of creating flowcharts with the Raptor tool. Students’ responses ranged positively from ‘strongly agree’ (16) followed by ‘agree’ (9) and ‘neutral’ (8). In addition, four of the students ‘disagree’ and three ‘strongly disagree’ in response to this statement.

Statement 6: It was easy for me to run Raptor program flowcharts once created.

Students’ responses varied regarding the ease of running Raptor program flowcharts once created. Figure 6.7 illustrates the greatest number of students indicated ‘strongly agree’ (13) followed by ‘agree’ (12) and ‘neutral’ (9) responses.

Statement 7: It would be easy for me to become skilful at learning to operate the Raptor tool to complete program tasks effortlessly and confidently.

Figure 6.7 indicates that it was easy to operate and to complete program tasks effortlessly and confidently with the Raptor tool. Students’ responses ranged positively from ‘agree’ (14) followed by ‘neutral’ (12) responses. Furthermore, only five students ‘disagree’ and only two ‘strongly disagree’ with this statement.

6.4.1.3 Learnability

Learnability refers to the user's ability to use the tool interface correctly in order to accomplish a task (Bouvier et al., 2012). Statements 8, 9 and 10 dealt directly with learnability. Figure 6.8 shows that students perceived Raptor to be moderately positive with respect to learnability.

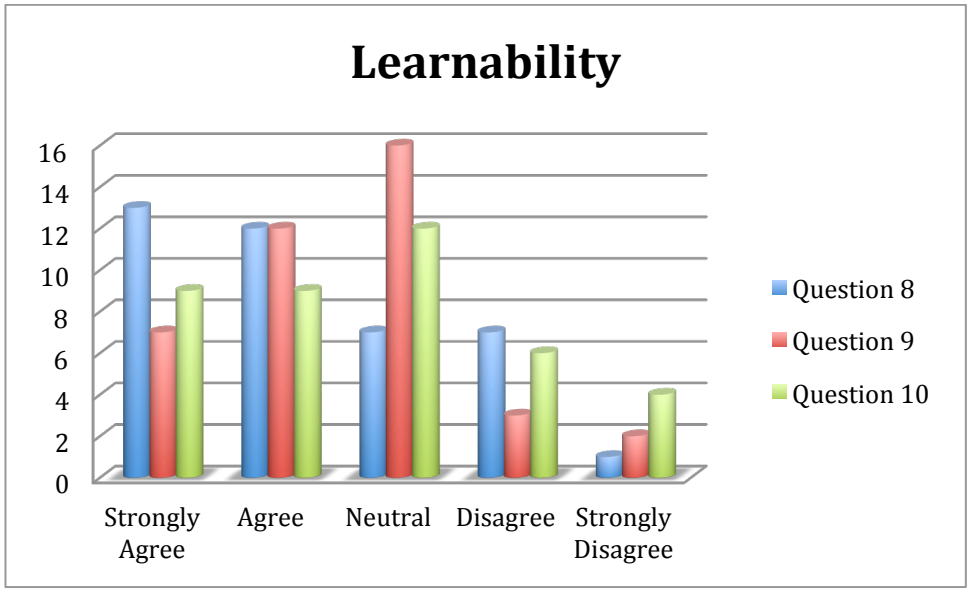


Figure 6.8: Learnability distribution

Statement 8: The symbols (assignment, call return, input, output, selection and loop) used in Raptor tool are easily understood and meaningful within the perspective of learning object-oriented programming tasks.

Students' views were positive with regard to Raptor's consistency in its representation of symbols and actions. Figure 6.8 highlights that the greatest number of students 'strongly agree' (13) followed by those who 'agree' (12). This statement indicates that the symbols (assignment, call return, input, output, selection and loop) used in the Raptor tool are easily understood and meaningful within the perspective of learning OOP tasks. Further, metaphors and language used in the tool correspond to real-world daily environments, thus were understandable and meaningful to the students.

Statement 9: There was no need to recall/remember information and instructions, as it was retrievable and visible whenever appropriate.

Figure 6.8 illustrates that students' responses were moderately positively, with predominantly 'neutral' (16) followed by 'agree' (12) responses. This indicates that information and instructions were retrievable and visible whenever appropriate. Also, only three of the students 'disagree' and two 'strongly disagree' with this statement.

Statement 10: I found Raptor to be an effective design tool to enhance algorithmic thinking of programming tasks without the need to consider the syntax of the programming language.

Raptor was perceived to be an effective design tool to enhance algorithmic thinking of programming tasks without the need to consider the syntax of the programming language. Figure 6.8 shows that students' responses varied with predominantly 'neutral' (12) responses followed by nine 'strongly agree' and another nine 'agree' responses with respect to Raptor effectiveness as a design tool to improve problem-solving. However, it was noted that six of the students 'disagree', while four of them 'strongly disagree' with this statement.

6.4.1.4 Error-recognition-diagnosis-recovery cycle and support information

This subcategory refers to the interface's ability to recover from errors and undesirable conditions (Bouvier et al., 2012). Figure 6.9 highlights responses to statements 11, 12 and 13 that dealt directly with the error-recognition-diagnosis-recovery cycle. The figure indicates that students perceived Raptor to be positive with regard to its ability to recover from errors and undesirable conditions.

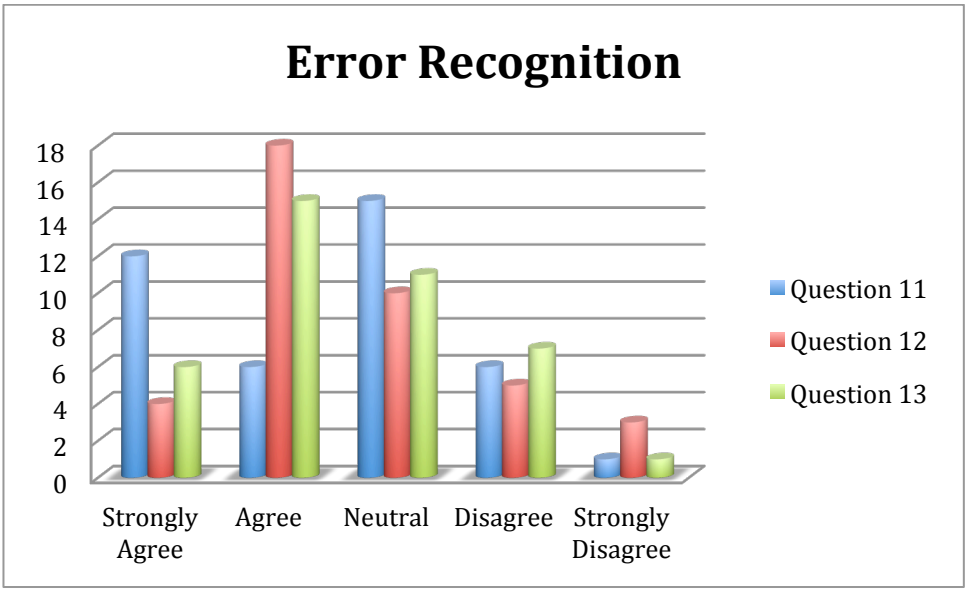


Figure 6.9: Error recognition distributions

Statement 11: Once Raptor flowchart was created, it was easy for me to generate the corresponding Java code.

Students' responses varied with respect to the ease of generating corresponding Java code once a Raptor flowchart was created. The results in Figure 6.9 reveal that the greatest number of students indicated 'neutral' (15) responses followed by 'strongly agree' (12) responses. Furthermore, six students indicated that they 'agree' and another six indicated that they 'disagree' with this statement.

Statement 12: Raptor displays appropriate error messages that clearly inform me about the error in plain language.

Figure 6.9 depicts mostly positive 'agree' (18) responses in Raptor's ability to display appropriate and clear error messages.

Statement 13: I am able to recover quickly and easily from errors given through precise, simple, efficient and effective error messages.

Figure 6.9 illustrates mostly positive 'agree' (15) responses in Raptor's ability to recover quickly and easily from errors given through clear and precise instructions. Moreover, it was noted that there were 11 'neutral' and seven 'disagree' responses to this statement.

6.4.2 CATEGORY 2: USER EXPERIENCE

Category 2 refers to user experience, which is part of the user interface factor. This study investigated the following subcategories: affect, visualisation, and motivation and interactivity of the perception of user experience, as illustrated in Figure 6.10. Each of the subcategories are discussed with regard to the survey statements.

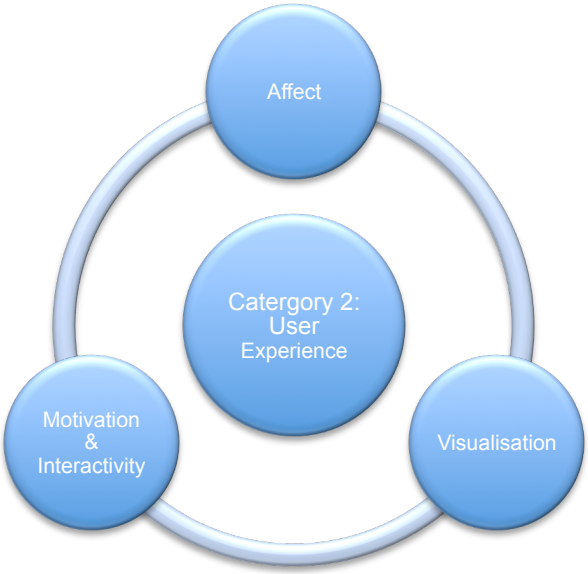


Figure 6.10: Category 2 subcategories

6.4.2.1 Affect

Affect refers to the subjective quality of how interacting with the interface feels to users (Bouvier et al., 2012). Statements 14 and 15 dealt directly with affect. Figure 6.11 indicates that students perceived Raptor to be positive with respect to the affect of the tool.

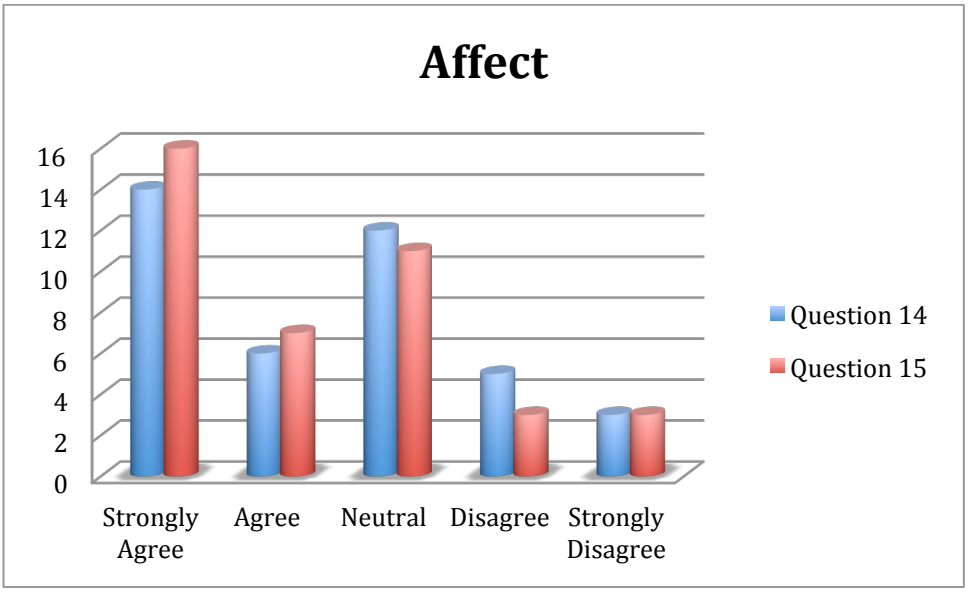


Figure 6.11: Affect distributions

Statement 14: The information (such as online help, on-screen messages, and other documentation) provided with this tool was useful, clear and easy to understand for me to effectively create programs and flowcharts.

Figure 6.11 highlights that students felt the Help system in Raptor and the support information provided proved to be useful and clearly defined in helping students to effectively create programs and flowcharts. The results reveal that typically students responses were 'strongly agree' (14) followed by 12 students who are 'neutral'. However, more positive responses were anticipated concerning the effectiveness and usefulness of the help provided by the Raptor system as well as the support information provided. This statement indicates information (such as online help, on-screen messages, and other documentation) provided with this tool was fairly useful, appropriate and presented in a structured, well-organised manner in order to effectively create programs and flowcharts. Furthermore, it was noted that only six students 'agree', five 'disagree' and three 'strongly disagree' with this statement.

Statement 15: I would recommend the tool to others.

Figure 6.11 demonstrates that students' responses were positive, with predominantly 'strongly agree' (16) responses, with recommending the tool to both occasional and regular users.

6.4.2.2 Visualisation appearance

This subcategory relates to general user experience (Bouvier et al., 2012). Statement 16 dealt directly with visualisation appearance. Figure 6.12 indicates that students perceive Raptor to be moderately positive in respect of the general user experience.

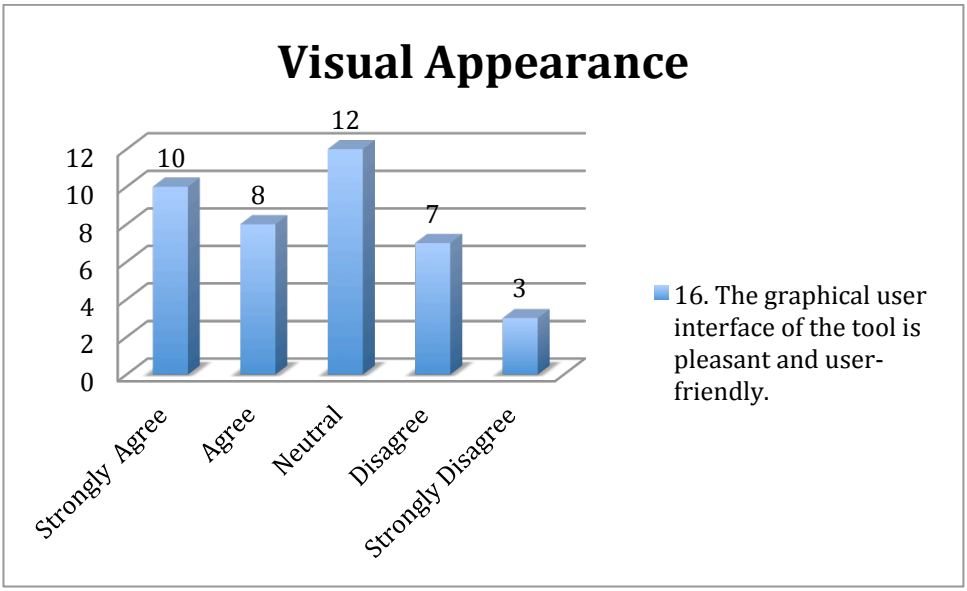


Figure 6.12: Visual appearance – Statement 16 distributions

Statement 16: The graphical user interface of the tool is pleasant and user-friendly.

Figure 6.12 illustrates moderately positive responses regarding visual appearance with respect to the general look and feel of the Raptor tool. The results indicate that there were predominately (12) 'neutral' responses followed by 'strongly agree' (10) responses. Thus, students were satisfied with the GUI of the tool that was pleasant and user-friendly. Furthermore, it was noted that eight of the students 'agree', seven 'disagree', while three 'strongly disagree' with this statement.

6.4.2.3 Learner motivation and interactivity

This subcategory refers to content and interactive features of the Raptor tool. Statements 17, 18 and 19 dealt directly with learner motivation and interactivity. Figure 6.13 highlights that students perceived Raptor to be mildly positive with respect to the learner motivation and interactive features.

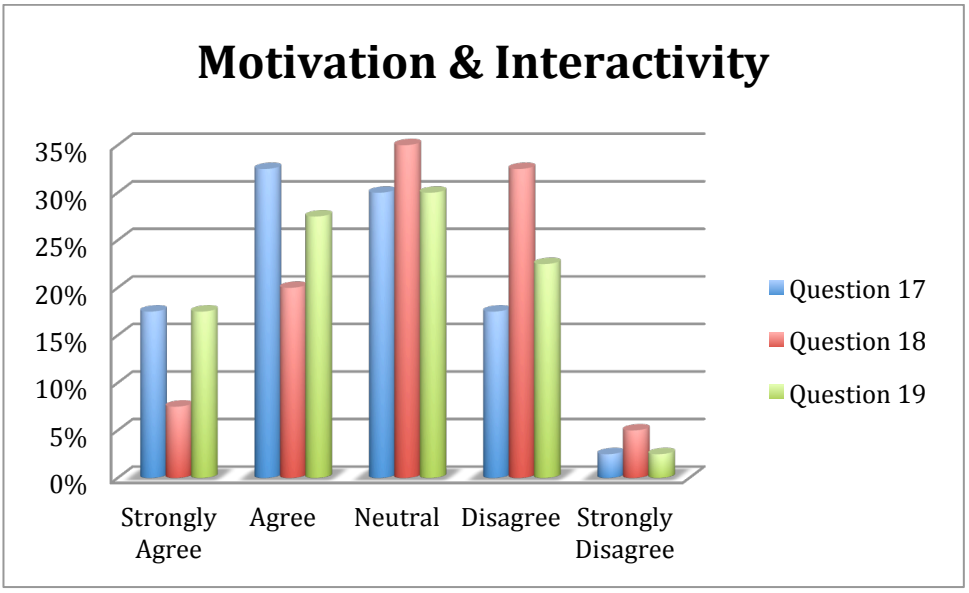


Figure 6.13: Motivation and interactivity distributions

Statement 17: It was a challenging, engaging and stimulating experience learning to program with Raptor.

Figure 6.13 demonstrates students' responses to their learning experience with Raptor. The data collected indicated positive responses in Raptor's ability to create a challenging, engaging and stimulating programming experience, with responses from mainly 'agree' (13) followed by 'neutral' (12). The results suggest that students felt interaction with the tool was clear and understandable through constructive, appropriate and timely feedback informing about the visibility of the system. Results further indicate that illustrations provided and interactions were used at the right level for effectiveness in studying through a DE learning process.

Statement 18: Utilising the Raptor tool increased my course workload.

Figure 6.13 illustrates that students' views varied with respect to utilising the Raptor tool, regarding in an increase in course workload. The results reveal that typically responses were 'neutral' (14) followed by 'disagree' (13). It would be worthwhile to consider, possibly in future, reducing tasks involved in the tutorial in order to cater for reducing the workload or perhaps incorporating it into course content.

Statement 19: The duration of experimenting and learning with the Raptor tool was at the right pace.

Students' subjective views indicate that the duration of experimenting and learning with the Raptor tool was at the right pace. Figure 6.13 shows that the greatest number the students indicated 'neutral (12) responses followed by 'agree' (11) responses. Furthermore, it was noted that only seven students 'strongly agree', nine 'disagree', while one indicated 'strongly disagree' with this statement. In future experiments, it may be advisable to possibly consider extending the time the students have to experiment with the tool.

6.4.3 CATEGORY 3: LEARNING PERCEPTION

Category 3 refers to learning perception. This study investigated the following subcategories: learner control and learning capability of the Raptor tool, as illustrated in Figure 6.14. Each of the subcategories will be discussed with respect to the survey statements.

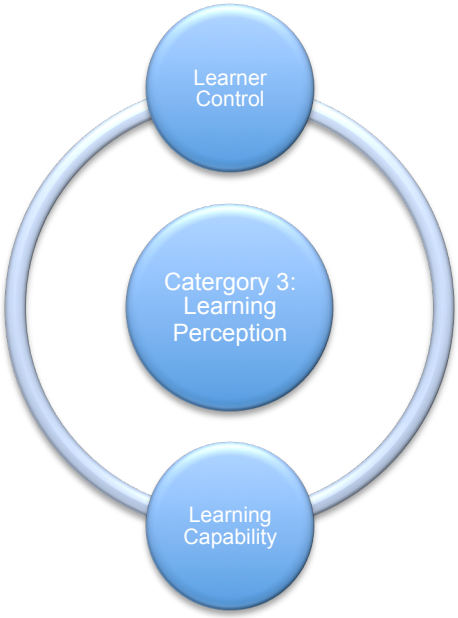


Figure 6.14: Category 3 subcategories

6.4.3.1 Learner control

This subcategory refers to learner control concerning time, place, content and sequence of learning (Ssemugabi & De Villiers, 2007). Statements 20 and 21 dealt directly with learner control. Figure 6.15 indicates that students perceived Raptor to be mildly positive with reference to the learner control.

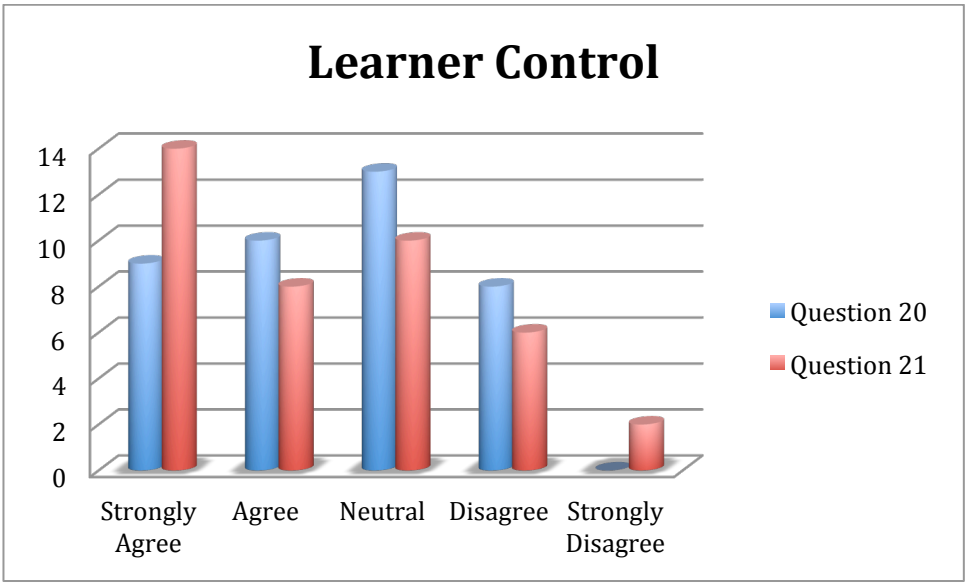


Figure 6.15: Learner control distributions

Statement 20: I am satisfied with the self-controlled learning with respect to time, place, content and sequence of learning.

Students felt Raptor supports various ways or learning styles and were satisfied with the self-controlled learning process of taking control of the learning process with respect to time, place, content and sequence of learning. Figure 6.15 highlights that students' responses were mostly 'neutral' (13) followed by 'agree' (10). The results reveal the tool's flexibility in its ability to be customised to personal learning approaches according to learner ability and styles. Additionally, it was noted that nine of the respondents 'strongly agree', eight 'disagree', whereas none of the respondents 'strongly disagree' with this statement.

Statement 21: I enjoyed programming with the Raptor tool.

Figure 6.15 illustrates positive responses to enjoying programming with the Raptor tool, with predominantly 'strongly agree' (14) responses. Results highlighted that the students tended to enjoy using Raptor, which is viewed as an important result because it shows that Raptor encouraged enhanced feelings about learning to program.

6.4.3.2 Learning capability of Raptor tool

This subcategory implies effectiveness of teaching capability of the tool, referring to both concept understanding and self-learning tool capability (Jourjon et al., 2011). Statements 22, 23, 24, 25, 26, 27, 28, 29 and 30 dealt directly with learning capability of the Raptor tool.

Statement 22: I performed well on the program exercises and examples in the object-oriented tutorials.

Students were satisfied with their performance relating to the programming exercises and examples in the object-oriented tutorials. Figure 6.16 shows that the students' responses were predominantly 'neutral' (14) followed by 'agree' (12).

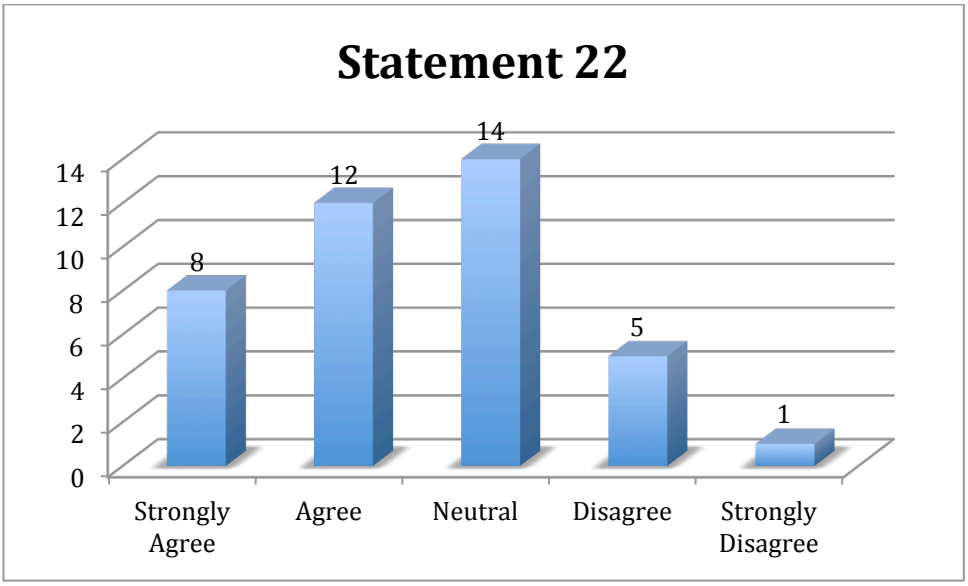


Figure 6.16: Statement 22 responses

Statement 23: There was sufficient feedback on the activities and knowledge construction guiding me as I created program flowcharts.

Students' responses in connection with sufficient feedback of activities and knowledge construction resulted in moderately positive responses. Figure 6.17 illustrates that students' responses were mostly 'neutral' (13) followed by 'agree' (11)

responses. This statement further indicates that students were satisfied with task guidance provided as they created program flowcharts.

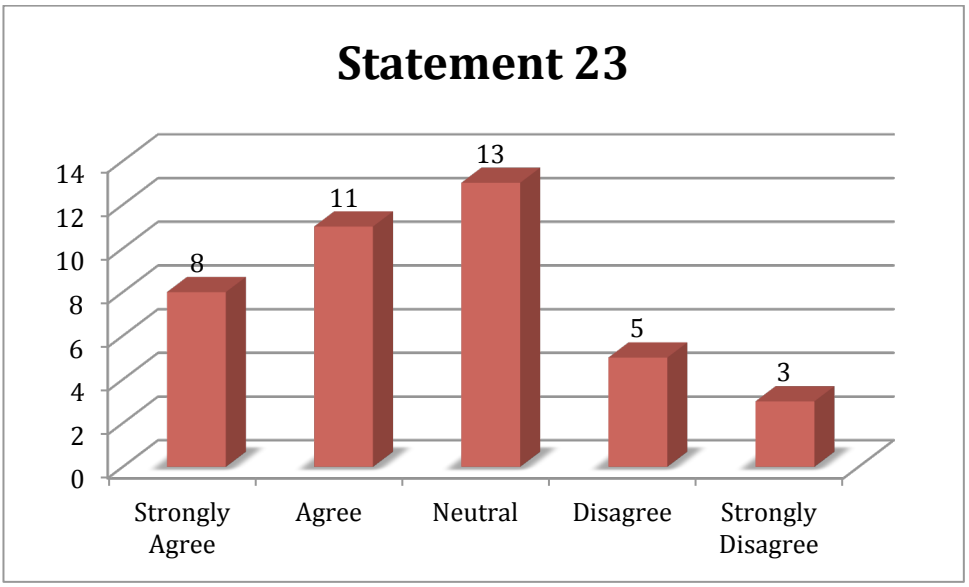


Figure 6.17: Statement 23 responses

Statement 24: The tool helped to enhance understanding and reinforced key object oriented concepts (inheritance and polymorphism).

Students were satisfied with the Raptor tool's ability in assisting to enhance understanding and reinforcing key object oriented concepts (inheritance and polymorphism). Figure 6.18 reveals that students' responses were mostly 'neutral' (15) followed by 'agree' (10) and 'strongly agree' (nine) responses.

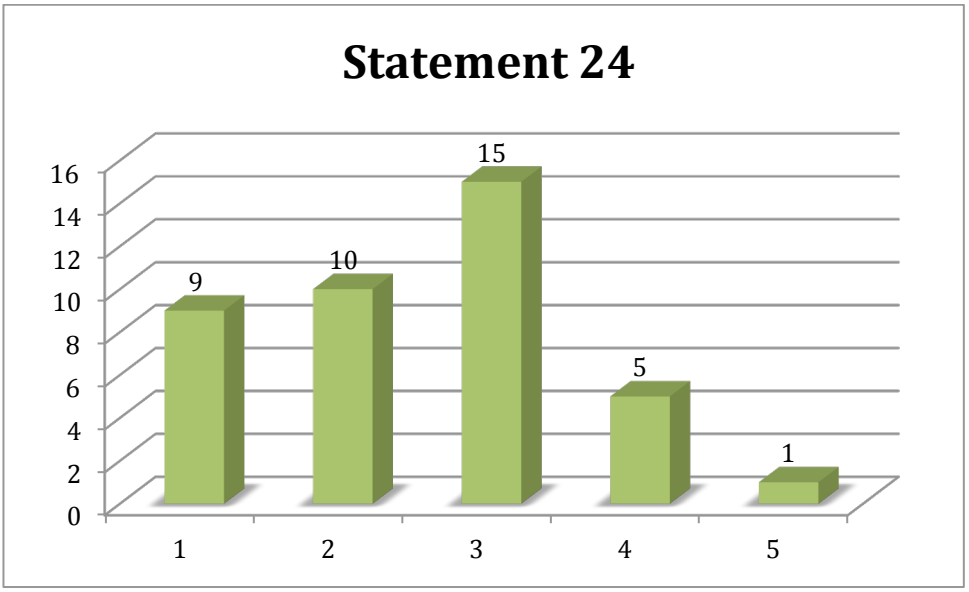


Figure 6.18: Statement 24 responses

Statement 25: I was able to effectively utilise the Raptor tool as a self-learning tool.

Students' perceptions of their ability to effectively utilise the Raptor tool as a self-learning tool and the effective teaching capability of the tool were perceived positively. Figure 6.19 highlights that students' responses were mostly 'agree' (15) followed by 'neutral' (10) responses. It was further noted that seven of the students 'strongly agree', another seven 'disagree', while only one indicated a 'strongly disagree' response.

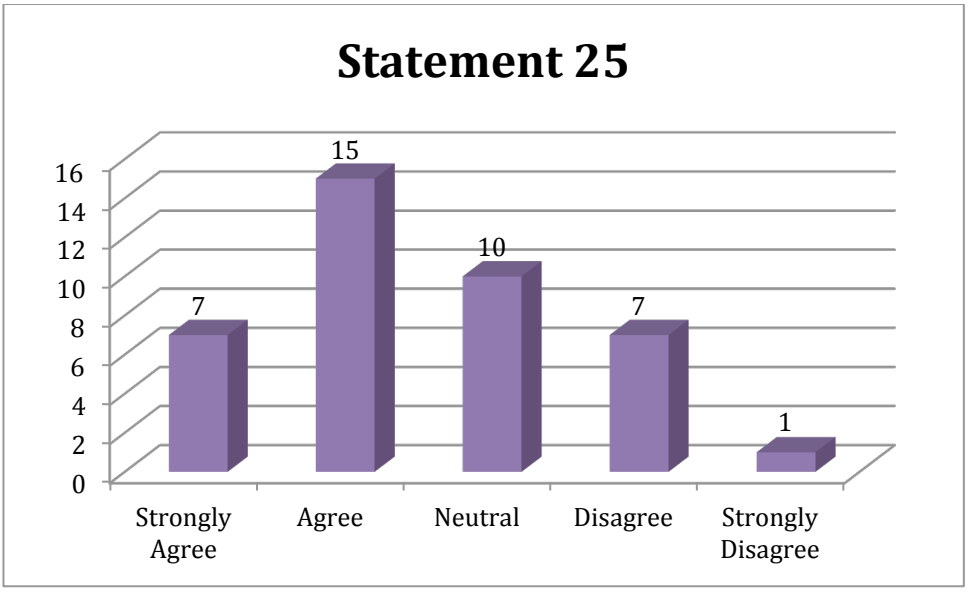


Figure 6.19: Statement 25 responses

Statement 26: As I was able to view contents of variables, it assisted me in testing and debugging programs.

Figure 6.20 depicts that students' perceptions of the variable watch features in Raptor were moderately beneficial. The students responses were 'neutral' (12) responses followed by 'strongly agree' (11) and 'agree' (10) responses. The variable watch feature enabled students to view contents of variables, which assisted in testing and debugging programs, improving problem-solving skills and improving knowledge understanding of how object-oriented programming operates. Moreover, it came to light that only three of the respondents 'disagree' and four 'strongly disagree' with this statement.

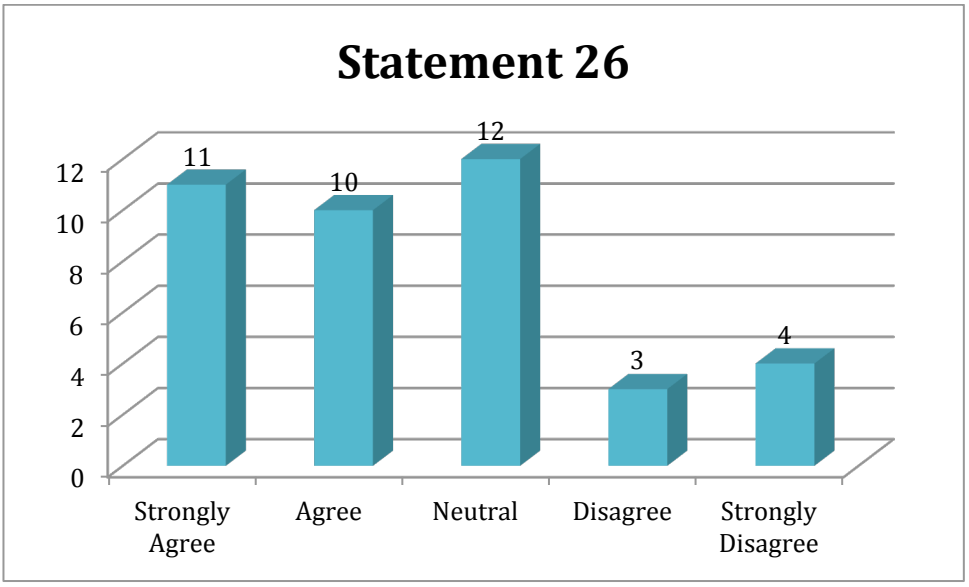


Figure 6.20: Statement 26 responses

Statement 27: Raptor tool made Java programming easier.

In Figure 6.21, it is illustrated that students' responses were mostly 'neutral' (14) followed by 'agree' (11) and 'strongly agree' (nine) responses. These results suggest that Raptor assisted in making Java programming easier. Furthermore, it was noteworthy that only four of the respondents 'disagree' and two 'strongly disagree' with this statement.

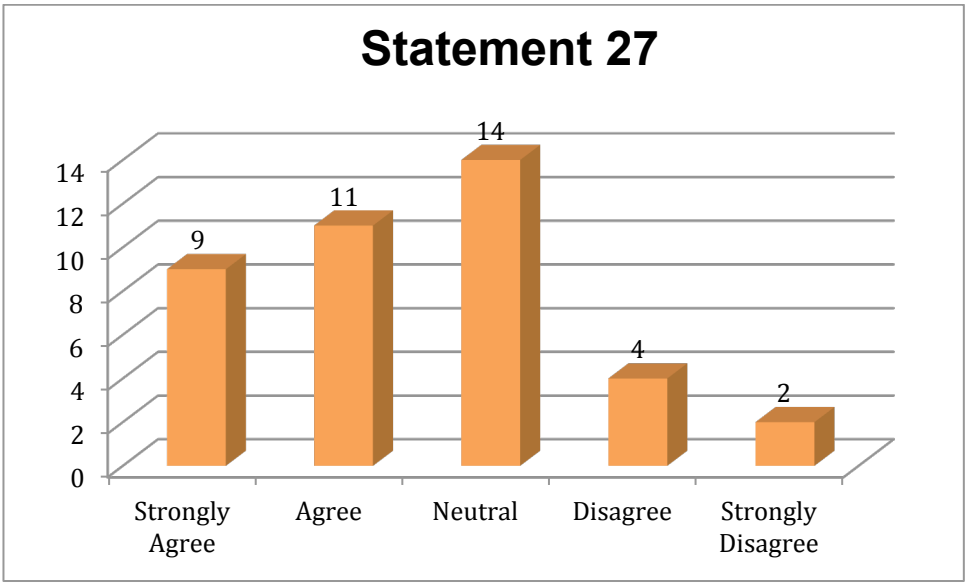


Figure 6.21: Statement 27 responses

Statement 28: Once the design in Raptor was implemented, generating the Java program would be easy for me.

Students' responses varied with respect to ease of generating the Java program once the design in Raptor was implemented. Figure 6.22 highlights that students' responses were mostly 'neutral' (15) followed by 'strongly agree' (11) and 'agree' (eight) responses. These results suggest that Raptor helps to reinforce a design methodology; thus, the flowcharting approach assists in easing the programming process.

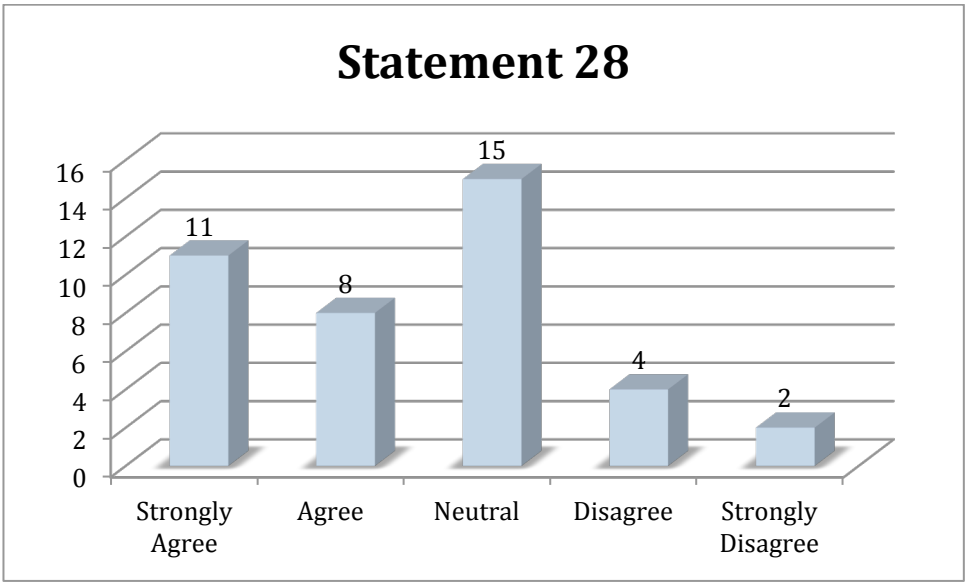


Figure 6.22: Statement 28 responses

Statement 29: I felt confident about taking what I learnt from this tool and applying it to my course.

Figure 6.23 shows that students' responses were mostly 'agree' (13) followed by 'neutral' (12) responses with regard to experimenting with the Raptor tool that supports correlation with course goals. This indicates that students felt confident about taking what they learnt from this tool and applying it to the course. Also, an observation was made that there were only eight 'strongly agree' and seven 'disagree' responses, but none of the respondents indicated 'strongly disagree' responses to this statement.

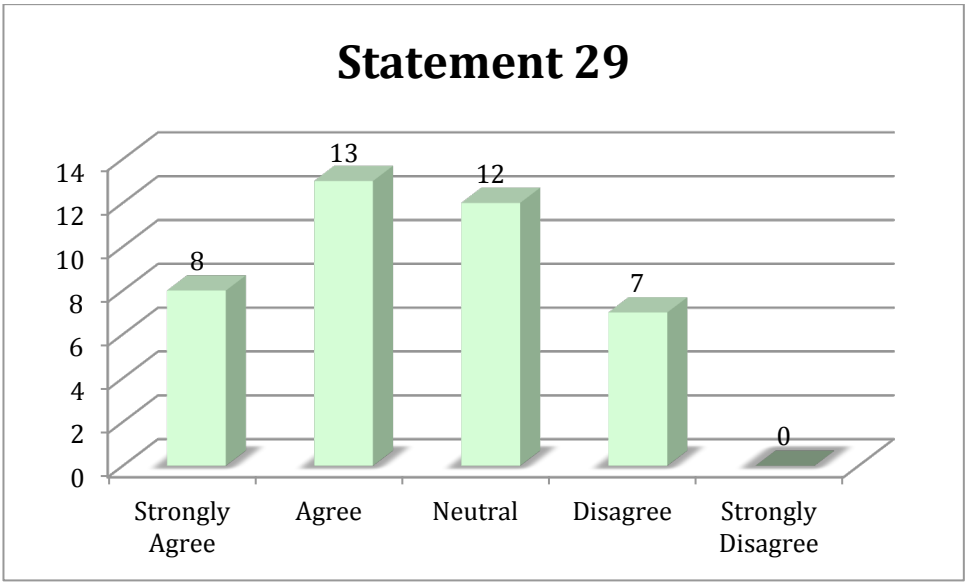


Figure 6.23: Statement 29 responses

Statement 30: Raptor assisted in enhancing my understanding of how computer programs operate.

Students' responses concerning Raptor's ability to assist in enhancing understanding of how computer programs operate were perceived to be moderately positive. Figure 6.24 demonstrates that students' responses were mostly 'neutral' (15) followed by 'strongly agree' (13) responses. This indicates that students perceived Raptor to be efficient in helping one to understand how computer programs operate; on that account, students perceived Raptor to be effective in helping to develop problem-solving skills and understanding the computer.

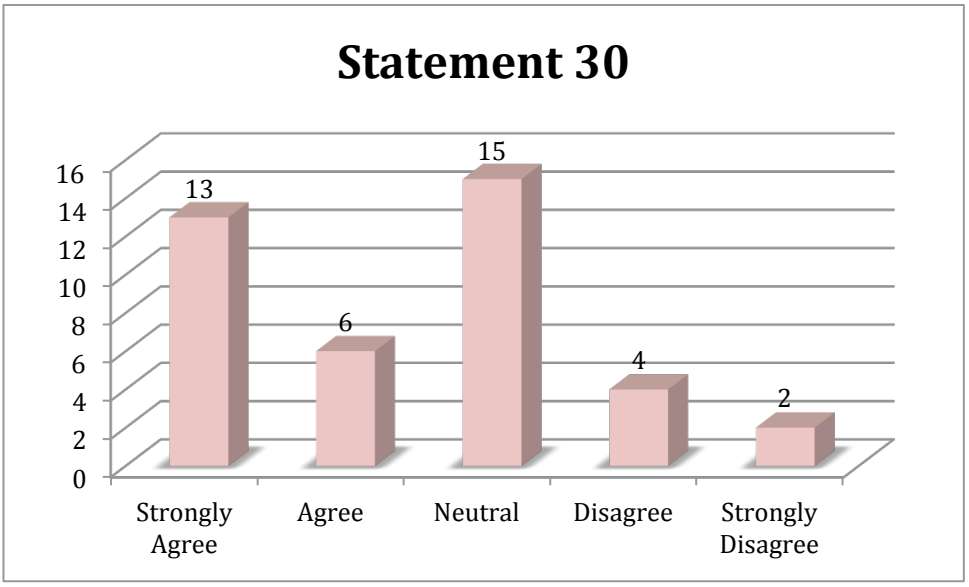


Figure 6.24: Statement 30 responses

6.5 QUALITATIVE RESULTS

In this section, the qualitative survey results and thematic analysis of the particular questions based on aspect 2 of research question 3 are discussed. The survey consisted of three open-ended questions, each of which are discussed and explained below. The qualitative questions examine the perception of novice computing students' evaluation of the effectiveness of the teaching capability of the web-based Raptor tool utilised in order to ease the difficulty of OOP. Further, students' experience level and evaluation of the tool were investigated. This section is important, as students' subjective evaluations will be used to confirm the quantitative findings already highlighted. The qualitative responses have been slightly adjusted to ensure readability.

6.5.1 QUALITATIVE QUESTION 1 – SURVEY QUESTION 31

To what extent do you consider the tool environment contributed to overcoming difficulties in object-oriented programming?

The first qualitative question corresponds to question 31 on the survey. The question examines the students' subjective evaluation of whether the Raptor tool environment contributed to overcoming difficulties in OOP. Qualitative responses were interpreted and consolidated to present and explain data. These have been captured, grouped and described into a small set of themes, which are discussed below. Appendix C indicates respondents' responses for this theme and categorised data reported for question 31 of the survey. Table 6.4 summarises the qualitative results reported for question 31 of the survey. The themes are categorised, and the corresponding number of respondents are detailed in Table 6.4 for each theme.

Table 6.4: Qualitative question 1 themes corresponding to question 31 of survey

Theme	Number
Lack of time: Respondents indicated that probably more time was needed in order to familiarise themselves with using Raptor.	2
Pedagogical concept understanding: Raptor assisted in enhancing understanding of OOP and problem-solving.	21
Visualisation and usability aspects: Respondents commented on the ease of use, flexibility, general usability, visual appearance and graphical user interface of Raptor.	6
User levels and guidance provided: This entails the audience that would most benefit and whether sufficient guidance was provided when doing the tutorial.	5
Undetermined: These responses were not considered in the analysis of themes.	6

6.5.1.1 Lack of time

This theme indicates that respondents felt that probably more time was needed for becoming familiar with using Raptor.

Only two students out of a total of 40 felt that they needed more time to become familiar with using Raptor and to master the tool. One of the students also noted that the visual representation enabled understanding of the logical flow of the programs. Respondent 4 indicated: “Instead of only words, one gets to work with diagrams that show the interrelation of things. I think given more time, I can get to understand the program better.” For further studies, it would be appropriate to allocate more time to master the tool, especially for those with little or no OOP experience.

6.5.1.2 Pedagogical concept understanding

This theme indicates that the Raptor tool assisted in understanding OOP fundamental concepts and problem-solving abilities.

Students' felt confident, that Raptor was beneficial in easing the difficulty of OOP and further that it assisted in understanding and reinforcing the fundamental concepts of inheritance and polymorphism. They also highlighted that the visual representation enabled understanding of the logical flow of arguments, hence assisting in identifying logic errors and helping in problem-solving abilities. Respondent 3 stated the following: "It helped in understanding how inheritance and polymorphism worked as well as with aid in the visual representation of the logical flow of arguments." Furthermore, it was interesting to note that one of the respondents indicated that Raptor is an effective design tool to enhance algorithmic thinking of programming tasks. Students felt confident about taking what they learnt from the Raptor tool and applying it to their course. The tool supports correlation with course goals, as Raptor assisted in easing Java programming through its visual nature. In addition to the pedagogical benefit of the tool, respondents particularly noted that the GUI of the tool was pleasant and user-friendly. In addition, the tool was simple and easy to use. However, one of the students suggested improving the GUI so as to attract more users.

6.5.1.3 Visualisation and usability aspects

This theme mainly focused on students' perceptions relating to ease of use, flexibility of the tool, general usability, visual appearance and graphical user interface of Raptor.

Respondents emphasised that the visual nature of Raptor was beneficial, as it was simple and had an easy-to-use navigation structure. The GUI of the tool was pleasant and user-friendly, enabling the student to solve problems and follow the logical flow of programs easily and efficiently. Further, the GUI was simple and easy to use, thus resulting in enhanced understanding of OOP. Respondent 19 indicated: "Raw Java coding can be very scary just by looking at it, but the Raptor tool eliminates that code fear because immediately when you see tons of code, you begin to feel intimidated and to think that you cannot do it. But with the Raptor tool, programming becomes easier because you deal directly with shapes, playing around with them to achieve your intended results and create Java code after. The tool is very important because it teaches and trains your mind to understand the concept of programming because if you understand the concepts, then code implementation

becomes easier." Moreover, students indicated the ease of creating flowcharts, which resulted in creating an enhanced and stimulating programming experience.

6.5.1.4 User levels and guidance provided

This theme entails the audience that would most benefit and whether sufficient guidance was provided when doing the tutorial.

Some students (five) also indicated that the tool would be most appropriate and beneficial for novice programmers. Respondent 15 indicated: "The tool is perfect for people learning how to code. The visuals of the application help to contribute in helping to understand how an application executes." More direction was required to complete the tutorial using the Raptor tool, particularly for those with minimal experience in flowcharts and algorithms. In future, workshops could be administered to provide more guidance on how to use the tool and tutorial activities. Furthermore, students enjoyed programming with the Raptor tool. The visual nature of Raptor assists in enhancing understanding of how computer programs operate, and it was beneficial in easing the difficulty of OOP. Respondents also suggested that Raptor should be implemented in the course. One of the students suggested introducing theoretical concepts at the outset before implementing Raptor.

6.5.1.5 Undetermined

Some respondents provided no comments, possibly as they felt they were not experienced enough to comment.

6.5.2 QUALITATIVE QUESTION 2 – SURVEY QUESTION 32

Do you have any previous object-oriented programming experience? If yes, please give details.

The second qualitative question corresponds to question 32 on the survey. The question determines students' previous programming history details. Qualitative responses were interpreted and consolidated to present and explain data. These have been captured, described and grouped into a small set of themes, which are discussed below. Appendix D indicates respondents' responses for this theme and categorised data reported for question 32 of the survey. Table 6.5 summarises qualitative results reported for question 32 of the survey. The category and corresponding number of respondents are displayed per categorised theme.

Table 6.5: Qualitative question 2 themes corresponding to question 32 of survey

Category	Number
None	23
Some	4
Experienced in OOP	13

6.5.2.1 None

These respondents indicated that they have no previous programming experience.

The majority of the students that responded have little or no programming experience in OOP. These users perceived the Raptor tool to have added value, particularly as the tool was beneficial in easing the difficulty of OOP and problem-solving. Further, these students indicated that the tool assisted in enhancing understanding and reinforced key object oriented concepts through its visual nature. Furthermore, they suggested that the tool would be most appropriate for novices. This was confirmed, as the majority of students with no experience indicated they felt Raptor provided the greatest benefit for concept understanding in comparison to other user levels. Raptor assisted in understanding the fundamental concepts of inheritance and polymorphism. These students also felt positive with respect to the visualisation and usability aspects of Raptor, as discussed in question 31 and 33. Respondent 19 stated: "I believe the Raptor tool will help students to understand programming

without being intimidated by the code or name of the language, since it uses shapes, which everyone can work with. I would also recommend Raptor for novice students, and I believe when the tool is exclusively implemented, it will yield the results expected by the institution."

Respondents in this category indicated that it was easy to use the tool to create flowcharts that assist in creating a stimulating programming experience and an ability to solve problems efficiently. Moreover, students enjoyed programming with the Raptor tool and were motivated to continue using the tool. Respondent 8 indicated: "I believe in my own personal capacity it would make programming interesting and make problem-solving quickly, especially with flowcharts." Raptor assisted in easing Java programming, reinforcing fundamental concepts within course goals. Thus, the tool supported correlation with course goals. Students suggested that Raptor should be implemented in the course. However, utilising the Raptor tool increased course workload.

It was noted that the respondents in this category felt that more direction and time were required to complete the tutorial using the Raptor tool, particularly for those with minimal experience in flowcharts and algorithms, as discussed in question 31 and 33. Respondent 7 specified: "Honestly, I don't think there will be any difficulties if the author can at least list few steps on how to start an activity for people who are not familiar with algorithm and flowcharts. I understand the language of flowcharts and algorithm, so for me, it was easy to apply my mind and play around with the tool. I like it." Possibly, the use of workshops to familiarise students with Raptor would be beneficial as suggested.

6.5.2.2 Some

These respondents indicated that they have some previous programming experience.

These students perceived the Raptor tool to have added value, particularly in assisting to enhance understanding and easing the difficulty of OOP. Students enjoyed programming with the Raptor tool and suggested that Raptor should be implemented in the course. Nevertheless, a few students – as with those with no experience – indicated they needed more time to master the tool. All students

irrespective of experience level in OOP suggested that the tool would be most appropriate for novices. Respondent 18 revealed: “I think the Raptor tool should be considered within the syllabus. It would be a great experience mostly for new students.” It was worth noting that one of the students suggested that the instruction to participate in the survey should be instructed through the UNISA portal or through the lecturer, not through email, as it seemed to appear as spam mail.

6.5.2.3 Experienced in OOP

These respondents indicated that they have previous programming experience, and details are noted of courses completed.

It seems that all users irrespective of experience level perceived the Raptor tool to have added value, particularly in that Raptor assisted in easing the difficulty of OOP and problem-solving. Additionally, these students also indicated that the tool helped to enhance understanding and reinforced key object oriented concepts. Some of the respondents – as also indicated by those with no experience – commented that Raptor is an effective design tool to enhance algorithmic thinking of programming tasks without the need to consider the syntax of the programming language though understanding of syntax is beneficial. Furthermore, respondents noted that testing and debugging programs could be done in the Raptor environment; thus, program tasks could be accomplished more efficiently and effectively. More experienced students and those with no experience felt positive with respect to visualisation and usability aspects of Raptor. Respondent 12 stated: “I think this tool would be useful to programmers as a whole, as it simplifies the process of developing an application.”

All students irrespective of experience level in OOP suggested that the tool would be most appropriate for novices in easing the difficulty of OOP. Students enjoyed programming with the Raptor tool even though they felt more instructions and time were required in order to become familiar with using Raptor as with those with no and some experience. Furthermore, students felt strongly that Raptor assisted in easing the difficulty of OOP through its visual nature and should be implemented in the course to ease the difficulty of OOP.

6.5.3 QUALITATIVE QUESTION 3 – SURVEY QUESTION 33

Additional Comments

The third qualitative question corresponds to question 33 on the survey. The question examines the students' additional comments of their experience with the Raptor tool. Qualitative responses were interpreted and consolidated to present and explain data. These have been captured, grouped and described into a small set of themes, discussed below. Appendix E indicates respondents' responses for this theme and categorised data reported for question 33 of the survey. Table 6.6 summarises qualitative results reported for question 33 of the survey. The themes are categorised and the corresponding number of respondents are displayed for each theme.

Table 6.6: Qualitative question 3 themes corresponding to question 33 of survey

Additional Comments	Number
No comments: These responses were not considered in the analysis of themes, as they had no additional comments.	15
Visualisation and usability aspects: Respondents commented on the ease of use, flexibility, general usability, visual appearance and graphical user interface of Raptor.	4
Lack of time: Respondents indicated that probably more time was needed in order to familiarise themselves with using Raptor.	5
Pedagogical concept understanding: Raptor assisted in enhancing understanding of OOP and problem-solving.	9
User levels and guidance provided: This entails the audience that would most benefit and whether sufficient guidance was provided when doing the tutorial.	7

6.5.3.1 No comments/not applicable

A total of 15 out of 40 respondents provided no comments, possibly as they felt they were not experienced enough to comment.

6.5.3.2 Visualisation and usability aspects

This theme corresponds to question 31. From the student point of view, the current system was perceived positively with respect to simplicity and ease, and it resulted in an enhanced user experience. The visual nature of flowcharts in Raptor assists in enhancing understanding of how computer programs operate. These comments are in line with those discussed in question 31 for the current theme. It was interesting to note that students commented that Raptor offers flexibility and support for self-study, allowing access independent of time and place. Furthermore, testing and debugging programs can be done in the Raptor environment; therefore, program tasks could be accomplished more efficiently and effectively. Respondent 22 commented: “Raptor is a good visual aid of the code, though. You’ll still need to have a good understanding of the programming syntax to write the code, but all of the logic, sequence and debugging of the program can be done with Raptor. This alone improves time spent troubleshooting program logic while coding. Raptor also makes it easy to explain your logic and even algorithm to other students/fellow programmers without going through various lines of code. I’ve found that most of the time is spent on logic and layout of a particular application/program by completing the outline of the program; in a Raptor flowchart, you can save a lot of time coding.”

6.5.3.3 Lack of time

This theme indicates that respondents felt that probably more time was needed for becoming familiar with using Raptor. To address this concern perhaps in the future, more time can be given and appended to the course goals. Possibly, a workshop could even be implemented as suggested in the comments. Furthermore, students indicated that they enjoyed and will continue to use the Raptor tool in order to become more familiar with the tool. However, it is important to note that comments indicated that utilising the Raptor tool increased their course workload. Respondent 10 indicated: “It was quite an experience using Raptor; my time was limited, but I was able to read and understand.”

6.5.3.4 Pedagogical concept understanding

This theme reveals that the Raptor tool assisted in understanding OOP fundamental concepts and developing problem-solving abilities.

Students perceived the tool as beneficial and suggested Raptor to be implemented in the course in future to ease OOP difficulty. In addition, respondents indicated that Raptor is an effective design tool to enhance algorithmic thinking of programming tasks without the need to consider the syntax of the programming language though understanding of syntax is beneficial. Respondent 39 revealed: “This is a great tool. With the correct guidelines, I feel that many people would stop looking at programming as such a daunting task. The flowcharts help you understand how the program actually works, thus making programming easier.” Utilising Raptor in the course resulted in an enhanced user experience. These comments are reiterated here once again as mentioned in the discussion of question 31 of the current theme.

6.5.3.5 User levels and guidance provided

This theme entails the audience that would most benefit and whether sufficient guidance was provided when doing the tutorial. Respondent 40 indicated: “I think it would be more beneficial to us (students) if there could be a class or two with the lecturer to help us when such initiatives are rolled out. “ With respect to target audience, students emphasised greater benefit for novices studying OOP but that it can be used for diverse user levels. This is an important comment, as it highlights that the tool can be used in DE because students have diverse knowledge understanding.

Furthermore, students suggested that workshops to familiarise themselves with the Raptor tool would be beneficial in easing the difficulty of OOP. A few respondents indicated that probably more direction and instruction were needed for becoming familiar with using Raptor. Respondent 4 stated: “This is an interesting program. It would indeed be of help but as with anything else, a little direction would be in order. Reason being, as with any programming language, it has its own words, its own world that one needs to steer through. Overall, for computing students, this would be easier. Nonetheless, one of the students indicated that the additional support documentation provided and available online help are appropriate and assisted to

effectively create programs and flowcharts.

6.6 CONCLUSION

Students' perceptions collectively were positive concerning usability, user experience and learning perception. Further, the subjective results clearly demonstrate that novices can use Raptor to ease the difficulty of OOP and in enhancing and understanding complex OOP concepts. Despite the fact that the sample size was too small to achieve statistical significance, these quantitative and qualitative results provide the practical basis for implementing Raptor in future. They demonstrate that Raptor positively impacts easing the difficulty of OOP through its visualisation, ease of use and creating a more enhanced learning experience. Raptor may be applicable for the purpose of teaching DE novices how to design algorithms, understand concepts and ease the difficulty of OOP.

The final chapter will focus on the summary of findings, conclusions drawn and recommendations made on issues raised.

CHAPTER 7
CONCLUSION AND RECOMMENDATIONS

7.1 INTRODUCTION

The penultimate chapter was on the results and discussion of the study. This final chapter provides the conclusion and recommendations of this study. This study was motivated by the need to overcome the pedagogical hindrances experienced by introductory OOP students and to address the high attrition rate evident among novice programmers.

This study aimed to provide the necessary tool support to effectively support knowledge construction and learning in distributed education instructional processes to ease the difficulty of OOP design and programming among novices. The approach that was utilised involved the use of an e-learning tool to ease programming difficulty with the aim of supporting the instructional process of OOP design and programming. The hope was that it could lead to the creation of a content-rich, enhancing learning experience that is useful for computer science programming students at UNISA. Further, the desire in this study was to enhance understanding and to create an enhanced perception in novices' of their ability to learn programming, ensuring they master the fundamental concepts of OOP software development.

With the current technology focus being on both academic and professional developments in object-oriented design and programming, it is acknowledged that object-oriented design and programming are still challenging for students. The main research question that had to be addressed was: Will the approach of using technology-enhanced e-learning tools improve the student experience by assisting to circumvent pedagogical hindrances experienced by DE students in introductory OOP?

The main components of the research are organised into three successive intersecting activities corresponding to the three sub-questions derived from the main question illustrated in Figure 7.1.

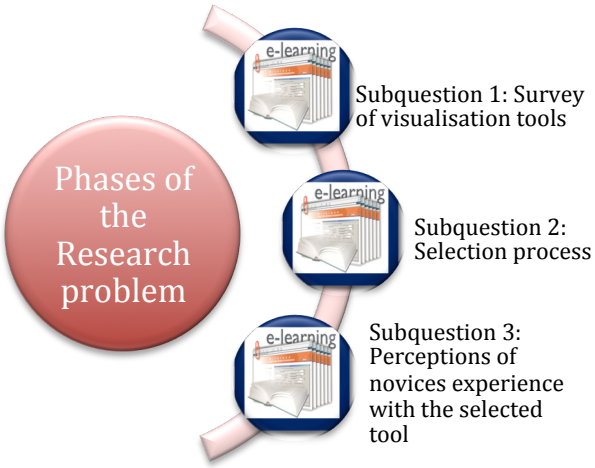


Figure 7.1: Phases of research process and corresponding research questions

The initial phase of the research process involved exploring and presenting a variety of alternative, appropriate visual programming environments in an attempt to lower novice programmers' barriers to OOP. The first sub-question addresses research conducted on what forms of technology-enhanced e-learning tools could be used to supplement the instructional process of OO design and programming. Moreover, the research indicates the impact of e-learning tools in assisting to create content-rich, engaging and easily accessible applications that are suitable for addressing diverse novice programmers' understanding and to master the fundamental concepts of object-oriented software development. This sub-question (phase 1) is addressed in Chapter 2.

This second sub-question corresponding to phase 2 of the research process addresses the selection process, through detailing criteria for the selection process. Thereafter, a detailed discussion was given on why the tool was selected. This was followed by an explanation of what the Raptor tool is about, which was addressed in Chapter 4, focusing on tools platform. An educational tool Raptor developed to lower novice hindrances when learning to program was specifically selected and presented. The Raptor tool supports the instructional process of OOP paradigm through limiting syntax and offering concrete visual representations. Further, it provides a more personalised learning approach and requires greater learner autonomy, thus creating an enhanced learning experience and resulting in a more positive attitude towards learning to program (Esteves et al., 2009).

Lastly, the core focus of this research was to gain an understanding of the perceptions of students through gathering experimental empirical data on the

subjective views of novices' experience and effectiveness of learning to program with asynchronous and web-based visual learning tool Raptor. This phase investigates the impact on students' perceptions of the process of integrating the *selected* technology-enhanced tool approach in distance education. It was hoped that a determination would be made regarding student perceptions on the impact of a visual e-learning tool support environment to overcome difficulties in learning OOP in distance education, as highlighted in Chapters 5 and 6.

Two aspects of the final phase were addressed in this study:

a) Will the selected visually engaging e-learning tool approach that illustrates solutions to algorithm design problems assist novices by motivating them through creating a more positive, enhanced user learning experience with technology? Novices' perceptions of the Raptor tool with respect to its usability, engaging user experience and learning capability of tools to support efficient and effective learning in understanding fundamental OOP concepts were given attention. These aspects of the research question were discussed in terms of the quantitative results, which included the statistical analysis for the particular questions in section 6.4.
b) To what extent does the student believe the tool environment has contributed to overcoming difficulties in learning OOP? These aspects of the research question were discussed in terms of the qualitative open-ended question results in section 6.5.

7.2 SUMMARY OF FINDINGS OF THE RESEARCH QUESTIONS AND ASSOCIATED SUB-QUESTIONS

7.2.1 SUB-QUESTION 1: WHAT IS AN APPROPRIATE E-LEARNING TOOL FOR TEACHING OOP?

The initial phase of this research corresponding to research question 1 was on the utilisation of educational tools to assist students to alleviate the learning barriers in understanding the OOP philosophy and its concepts. In this initial phase, the hope was to achieve a more personalised, motivating and enhanced learner experience in order to effectively support knowledge construction and learning in DE to ease the difficulty of OOP design and programming among novices. Tool support relating to OOP pedagogy to address difficulties in OOP among novices is extensively

discussed. There are several visualisation tools with a diverse range of functionalities. There is a rich set of visualisation tool paradigms for novice programmers to choose from; these range from allowing students to control both symbols and code to those which reject text-based programming. A literature review of various e-learning tools that have been successfully implemented at various institutions were presented. The purpose of the survey of visual environments focused on their potential effect on teaching programming and being an acceptable effective tool to consider in order to ease the difficulty of OOP among novices. While the survey of visualisation tools is not exhaustive, this researcher feels that tools have been identified which represent the spectrum of programs available for the purpose of easing the difficulty of OOP.

7.2.2 SUB-QUESTION 2: WHAT CRITERIA ARE INVOLVED IN THE SELECTION PROCESS TO DEFINE AN APPROPRIATE TOOL?

Phase 2 of the research addresses question 2 and involved the selection process for an appropriate tool to ease novices' understanding of OOP fundamental concepts. The selection process entailed detailing several requirements that would define the ideal choice of the most appropriate tool. The primary goal of the selected tool is to ease OOP difficulties experienced by novices through improving student problem skills and minimising syntactic complexities. The key feature is ensuring the tool is well suited for distance education novice programmers. Consideration was given to the following requirements:

1) *Support of a visual algorithm tool representation* through enhancing algorithmic problem-solving thinking.
2) *Use of syntax corresponding to Java* with some visualisation system.
3) *Easy* for novice distance education students to use.
4) *Minimalise cognitive overload of students*.
5) *Availability* either as open-source software or freeware.
6) Ease of *installation* for novices.
7) Provide constructive and corrective *feedback*.
8) Tool should *cognitively engage students* by actively engaging students and suit the population.
9) *Flexibility with respect to learning processes* in order to meet the diverse needs of individuals to stimulate individuals' motivation and development.

Raptor was designed by Carlisle et al. (2005) primarily to minimise barriers in programming, focusing on limiting syntax complexity and offering concrete visual representations on which to work. The goal of Raptor is to design and execute algorithms independent of a programming language. Raptor is an iconic programming environment where programs are created visually using UML and flowcharts. A combination of the following features makes Raptor an ideal choice of educational programming environment for the purpose of this study:

a. Raptor's visual development environment enables users to follow the flow of instruction execution of programs one symbol at a time. Furthermore, Raptor enables visual representation of the algorithm. *These features of Raptor apply to and satisfy criterion 1.*
b. The resulting programs can be executed visually within the environment and converted to Java code for novice, intermediate and object-oriented modes. Raptor uses a Java-like syntax (Giordano & Carlisle, 2006). *These features of Raptor apply to and satisfy criterion 2.*
c. Raptor reduces the complexity experienced by students, as it provides a simple easy-to-use visual environment for OOP algorithm design, development and problem-solving (Carlisle et al., 2005). *Thus, this feature of Raptor applies to and satisfies criterion 3.*
d. Raptor reduces the complexity of writing programs, hence allowing students to focus on the problem to be solved and engaging them more actively in the learning process (Carlisle et al., 2004). *Thus, this feature of Raptor applies to and satisfies criterion 4.*
e. Raptor is a free, open-source tool that entirely supports introducing object-oriented programming, comprising the complex features of polymorphism and inheritance (Carlisle, 2009). Most importantly, the strength of the selected e-tool is that it provides an anytime, anywhere, easy access learning environment for novice students (Truong et al., 2005). *These features of Raptor apply to and satisfy criterion 5.*
f. The Raptor e-learning system is a *web-based tool* for teaching novice programmers; as such, it eliminates the programming environment installation difficulties usually experienced by novice programmers. Raptor enables students to execute their algorithms within the environment; thus, there is no need for multiple tools for compiling and executing programs. *These features of Raptor apply to and satisfy criterion 6.*

g. Recognition of appropriate error messages that clearly inform about the error. *These features of Raptor apply to and satisfy criterion 7.*
h. Giordano and Carlisle (2006) showed that the effectiveness of the Raptor visual programming environment through the use of flowcharts when developing algorithms actively engaged students, having a positive impact on student learning. *This feature of Raptor applies to and satisfies criterion 8.*
i. The Raptor tool offers flexibility in its development for specific academic purposes, as instructors can customise the environment and facilitate more stimulating implementations by adding to the built-in procedures (Carlisle et al., 2004). *These features of Raptor apply to and satisfy criterion 9.*

These results influenced some of the opinions this researcher had when they considered the use of the web-based tool Raptor as an ideal visualisation tool platform in creating both a positive programming learning experience and assisting to lower a distance education novice programmer's hindrances when learning to program.

7.2.3 SUB-QUESTION 3: WHAT ARE STUDENTS' PERCEPTIONS OF THE TOOL?

The final phase of the research corresponding to research sub-question 3 entails evaluating the subjective perceptions of the students' experience with the web-based, e-learning platform Raptor supplemented with distance education. The target population constituted students registered for the undergraduate semester module of Interactive Programming (ICT2612) and Introduction to Web Design (ICT1513) in 2014 and 2015 at UNISA.

The mixed method approach was used in this study to address research sub-question 3 more completely, where use was made of qualitative as well as quantitative methods to collect and analyse data. The study utilises convergent parallel designs that are characteristic of independent concurrent data collection and analysis of quantitative and qualitative data in the same stage of the research procedure followed by the merging of the two data sets into the final interpretation phase (Creswell & Clark, 2011). For the purpose of this study, the value of the mixed method responses from the subjective view of open-ended questions were mainly used to triangulate with the questionnaire's close-ended questions in order to

ascertain the accuracy and validity of the responses and allow for a more complete, comprehensive analysis of the research situation.

The aim of this study was to support the experience of novice undergraduate students by presenting programming concepts through a succession of clearly designed steps in tutorials so that students can gain understanding and to reinforce programming concepts by completing the tasks through a specifically designed tutorial work plan. The Raptor tutorial task of polymorphism and inheritance allows students to graphically experiment with these and related programming concepts. To assess the tool support, an online survey was developed. Students were required to complete tutorial tasks at their own pace asynchronously, as discussed fully in the previous chapters. At the completion of the tutorial, the survey was administered to novices. This research utilised an empirical research design and primary data (new data collected) using a specially designed electronic survey to determine learner perceptions and experience of the web-based, e-learning Raptor tool.

Research sub-question 3's summarised findings are discussed below in terms of the following: quantitative results, which involve the statistical analysis for the particular questions, and qualitative results of open-ended questions.

7.2.3.1 Quantitative findings of research sub-question 3

This study used descriptive research in the analysis process to address the quantitative questions. The quantitative questions examined the extent to which novice-computing students evaluate the following three key variables:

A. The user interface factors that include ***perceived usability*** of the Raptor tool with respect to the following subcategory findings:

   i. Effectiveness

   The evaluation of quantitative results for this subcategory showed that Raptor was effectively utilised in the learning process, hence assisting to increase productivity. Students highly appreciated Raptor for its simplicity and ease of use to effectively accomplish tasks. Furthermore, the tool was flexible in its ability to accommodate different user levels from novice to more advanced user level students.

ii. Efficiency

The quantitative results established that students were able to operate Raptor efficiently in order to create and run flowcharts easily as well as to complete program tasks effortlessly and confidently.

iii. Learnability

The quantitative analysis of the evaluation forms regarding learnability revealed that students' views were positive with respect to Raptor's consistency in its representations of symbols and that the actions used were easily understood and meaningful, within the perspective of learning OOP tasks. Additionally, information and instructions were retrievable and visible whenever appropriate, and there was no need to remember information. Raptor was perceived to be an effective design tool to enhance algorithmic thinking (problem-solving) of programming tasks without the need to consider the syntax of the programming language.

iv. Error-recognition-diagnosis-recovery cycle

Raptor was rated positively in its ability to display and recover quickly and easily from errors given through clear, precise and appropriate error messages and instructions. However, students' responses varied with respect to the ease of generating corresponding Java code once a Raptor flowchart was created.

B. The ***user experience*** of the Raptor tool is discussed with reference to the following subcategories and findings:

i. Affect

Students felt the Help system in Raptor and the support information provided are moderately useful in helping students to effectively create programs and flowcharts. This was unexpected, as the anticipation was for more positive responses with respect to the effectiveness and usefulness of the help provided by the Raptor system as well as the support information provided. This could be attributed to these being optional in the tutorial task schedule. Nevertheless, it was encouraging that results indicated that students would highly recommend the tool to both occasional and regular users.

ii. Visualisation appearance

The quantitative assessment of visual appearance ratings was surprising, as more positive responses were expected regarding visual appearance, considering that Raptor is an iconic programming environment where programs are created visually using UML and flowcharts. Students were satisfied and felt moderately positive regarding the general look and feel of the graphical user interface of the tool that was pleasant and user-friendly.

iii. Learner motivation and interactivity

The quantitative analysis established that Raptor created a challenging, engaging, and stimulating programming experience. The results suggest students felt interactions were used at the right level for effectiveness in studying through the DE learning process. Responses with respect to an increase in course workload, duration of experimenting and learning pace were received moderately well.

C. The ***learning perception*** with respect to the following subcategory and findings was given attention:

i. Learner control

Students felt that Raptor supports various ways or learning styles. They were also satisfied with the self-controlled learning process of taking control of the learning process with respect to time, place, content and sequence of learning. The quantitative assessment of learning perception indicated the tool's flexibility in its ability to be customised to personal learning approaches according to learner ability and styles. However, the quantitative assessment of learning control ratings were surprisingly lower than anticipated, as the tool is a web-based, e-learning platform that is accessible anytime, anywhere. Consequently, it provides students with greater hands-on time and more flexibility; it was expected to receive more positive feedback with regard to self-learning control aspects. Furthermore, students tended to enjoy using Raptor, which is viewed as an important result, as it shows that Raptor encouraged enhanced feelings about learning to program.

ii. Learning capability of the web-based tool that was utilised

This subcategory implies effectiveness of teaching capability of the tool, referring to both concept understanding and self-learning tool capability (Jourjon et al., 2011).

Students were moderately satisfied with the effectiveness of the teaching capability of the tool with respect to their performance in tutorials, feedback and task guidance provided. The quantitative ratings concerning concept understanding were received moderately well and led to understanding key concepts it was meant to address. Results suggest that Raptor helped to reinforce a design methodology, as the flowcharting approach assisted in easing the programming process. Furthermore, findings indicated that students perceived Raptor to be efficient and effective in helping to understand how computer programs operate and in developing problem-solving skills.

Students' perceptions of their ability to effectively utilise the Raptor tool as a self-learning tool and teaching capability of the tool was perceived positively. This is important, indicating that the tool can be utilised in the DE instruction process, as it is viewed positively in terms of its teaching capability. It was found that the variable watch feature in Raptor was received well by students. The variable watch feature enabled students to view contents of variables, which assisted in testing and debugging programs. This variable watch feature improved problem-solving skills and improved knowledge understanding of how object-oriented programming operates.

Important and noteworthy results indicated that experimenting with the Raptor tool supports correlation with course goals. This indicates that students felt confident about taking what they learnt from this tool and applying it to the course.

7.2.3.2 Qualitative findings of research sub-question 3

The qualitative questions examined the students' subjective evaluation of Raptor with respect to the perception of whether the tool environment contributed to overcoming difficulties in OOP as well as to determine any previous object-oriented programming experience details. The qualitative analysis of research provided the potential benefit of analysis of the impact e-learning platform tools as seen by students in addressing difficulties of OOP. Qualitative questions provide emergent themes and interesting quotes, thus making it possible to validate and enhance the quantitative survey findings. Qualitative responses were interpreted and consolidated to present and explain data. These have been captured, grouped and described into a small set of themes, findings that are discussed below. The following themes emerged.

A. Lack of time

This theme indicates that respondents felt that probably more time was needed for becoming familiar with using Raptor and to master the tool. It was noted that respondents felt more direction was required to complete the tutorial using the Raptor tool, particularly for those with minimal experience in flowcharts and algorithms. Respondents suggested that workshops to familiarise themselves with the Raptor tool would be beneficial in easing the difficulty of OOP. Furthermore, students indicated that they enjoyed and will continue to use the Raptor tool to become more familiar with the tool. However, utilising the Raptor tool increased students' course workload. These qualitative comments correlate with quantitative findings on affect and learning capability of tool subcategories already discussed.

B. Pedagogical concept understanding

This theme indicates that the Raptor tool assisted in understanding OOP fundamental concepts and problem-solving abilities.

Data showed that most of the students irrespective of programming experience felt positive that Raptor was beneficial in easing the difficulty of OOP and further that it assisted in understanding and reinforcing the fundamental concepts of inheritance and polymorphism. They also indicated that the visual representation enabled understanding of the logical flow of arguments, hence assisting in identifying logic errors, which assisted in problem-solving abilities. Furthermore, it was interesting to

note that one of the respondents indicated that Raptor is an effective design tool to enhance algorithmic thinking of programming tasks without the need to consider the syntax of the programming language though understanding the syntax is beneficial. Moreover, respondents noted that testing and debugging programs could be done in the Raptor environment; thus, program tasks could be accomplished more efficiently and effectively.

Students felt confident about taking what they learnt from the Raptor tool and applying it to the course. Raptor assisted in easing Java programming, reinforcing fundamental concepts within course goals. Therefore, the tool supports correlation with course goals, as Raptor assisted in easing Java programming through its visual nature. Students perceived the tool as beneficial and suggested Raptor to be implemented in the course in the future in order to ease OOP difficulty. Furthermore, experimenting Raptor in the course resulted in an enhanced user experience.

The qualitative discussion of this theme is consistent with quantitative results for learner motivation and interactivity category. This has already been discussed.

The findings of this study are in agreement with Carlisle et al. (2005), who established that students preferred and were more successful at designing algorithms visually when compared to traditional language or writing of flowcharts, as it assisted students to easily track the program control flow and to solve problems. Further, these authors noted that in comparison to the Cots tool, Raptor was the preferred tool and easier for the development of Java applications (Giordano & Carlisle, 2006).

C. User levels and guidance provided

This theme entails the audience that would most benefit and whether sufficient guidance was provided when doing the tutorial. Quantitative discussions relating to target audience emphasised greater benefit for novices studying OOP but can be used for diverse user levels. This is an important comment, as it indicates that the tool can be used in DE because students have diverse knowledge understanding. As expected, students irrespective of experience level in OOP suggested that the tool would be most appropriate for novices. This was confirmed by the majority of the students with no experience who indicated that Raptor provided the greatest benefit for concept understanding in comparison to other user levels.

Respondents' comments regarding user levels and guidance provided are in agreement with the quantitative findings relating to the effectiveness and affect subcategories. These have already been discussed.

The findings are in line with Giordano and Carlisle (2006), who showed that the effectiveness of the Raptor visual programming environment, use of flowcharts when developing algorithms actively engaged students, thus having a positive impact on student learning (and enhanced learning outcomes). Giordano and Carlisle (2006) further emphasised that it is perfectly suitable for teaching novices how to design algorithms to enable them to write structured programs and reinforcing design methodology.

D. Visualisation and usability aspects

This theme mainly focused on current systems perceived as easy to use, flexible, generally usable, visual appearance and graphical user interface of Raptor.

Discussions based on the qualitative findings indicated that students felt positive with respect to visualisation and usability aspects of Raptor. Respondents emphasised that the visual nature of Raptor was beneficial, as it was simple and had an easy-to-use navigation structure. Furthermore, the GUI of the tool is pleasant and user-friendly. However, one of the respondents suggested improving the GUI to attract more users. Moreover, students indicated ease of creating flowcharts, which resulted in creating an enhanced and stimulating programming experience. Students commented that the visual nature of flowcharts in Raptor assists in enhancing understanding of how computer programs operate and solve problems efficiently and effectively.

Qualitative discussion regarding visualisation and usability aspects are consistent with the quantitative results regarding user interface and user experience categories. The findings are in line with Carlisle et al. (2005) who found that Raptor offers a simple environment for developing algorithms.

7.3 RECOMMENDATIONS

In light of the findings, the following recommendations are made:

- The findings reported in the study justify the support of the Raptor visualisation tool in engaging individuals in order to enhance their learning experience to ease OOP difficulties experienced among novices and to enhance OOP concept understanding.
- The results support using Raptor be used in other courses and be employed in different institutions where it makes sense to consider Raptor, or another visualisation platform discussed in the dissertation, as a design platform in a course that teaches novices programming.
- The findings have implications for students in creating an engaging e-learning experience; thus, learning to program could result in an increase in students' learning confidence and could thus lead to a reduced dropout rate.
- As e-learning technology is motivated by learner needs, it can provide learning opportunities suited to learner interests and needs; therefore, it can lead to an enormous potential to stimulate individuals' motivation and development in creating a more positive learning experience to overcome barriers in programming and enhance concept understanding to address diverse needs of students in DE.
- The significance of combining e-learning tools and distributed education is its potential to transform the way one learns, improve the learning experience, improve quality in education, as well as provide support for lifelong learning.
- The findings of this research show that by providing an effective, appealing e-learning experience through integration with the Raptor tool, this can lead to enhanced learner interactivity, personalisation and engagement of students.
- This study will particularly benefit IT educators about various tools researched as well as encourage them to be become more knowledgeable about the use of the Raptor tool in creating a rich learning experience to overcome OOP hindrances experienced among novices.
- Raptor perceptions that impact on DE have proved to be an effective tool in creating more accessible environments that are independent of time, pace, place or physical location, and can be customised to students' needs.
- The results of this research could suggest implementing Raptor in the course to ease OOP understanding.

- The findings indicated that the visual nature of Raptor assisted in enhancing understanding of how computer programs operate and was beneficial in easing the difficulty of OOP.

7.4 SUGGESTIONS FOR FUTURE RESEARCH

Based on the findings, this study suggests the following areas for further research:

- Reducing tasks involved in the tutorial to cater for reducing the workload should be possibly considered in the future.
- In future experiments, it may be recommended to perhaps extend the time the students have to experiment with the tool and to master the tool, especially for those with little or no OOP experience.
- Respondents suggested that workshops to familiarise themselves with the Raptor tool would be beneficial in easing the difficulty of OOP. In future, workshops could be administered to provide more guidance on how to use the tool and tutorial activities.
- It is suggested to introduce theoretical concepts initially before implementing Raptor.
- A request to participate in the survey should be made through the UNISA portal or through lecturers, not through email, as it may appear to be spam mail.
- The results of this study open the need for further research, perhaps to allow students to choose from a few different visualisation tools to serve a highly diverse group in DE and investigate its effectiveness.
- It is recommended that Raptor be carefully planned within the context of a Java course curriculum, as it could result in an enhanced learning experience.
- Investigate if students learn better using Raptor tracking performance measure pre- and post-test instrument for the course to assess the learning key computing concepts and skills.
- Compare failure and dropout rates of current and previous exposure to Raptor to see if it resulted in increased motivation by tracking the number of students that have moved on to take the next level of courses.

- Future work can be devoted to applying Raptor tool to upcoming offerings of the same courses in order to progressively build more significant, quantitative and qualitative measures over an increased population size.
- As a solution to improving the self-learning aspect of the platform, researchers can implement a more structured tutorial from the beginning of the semester that will be in correlation with the course.

7.5 CONCLUSION

This study was motivated by the need to overcome the pedagogical hindrances experienced by introductory object-oriented programming students in order to address the high attrition rate evident among novice programmers in distance education. The main research question that had to be addressed was: Will the approach of using technology-enhanced e-learning tools improve the student experience by assisting to circumvent pedagogical hindrances experienced by DE students in introductory OOP?

The main components of the research are organised into three successive intersecting activities corresponding to the three sub-questions derived from the main question. The initial phase of the research process involved exploring a variety of alternative visual programming environments for novices. Thereafter the selection process detailed several requirements that would define the ideal choice of the most appropriate tool. An educational tool Raptor was selected. The Raptor tool supports the instructional process of OOP paradigm through limiting syntax and offering concrete visual representations. Further, it provides a more personalised learning approach and requires greater learner autonomy, thus creating an enhanced learning experience and resulting in a more positive attitude towards learning to program (Esteves et al., 2009). Lastly, the core focus of this mixed method research was to evaluate undergraduate UNISA students' perceptions of the Raptor e-learning tools with respect to the perceived effectiveness in enhancing novices' learning experience, in an attempt to lower the barriers to object-oriented programming.

Students' perceptions collectively were positive concerning usability, user experience and learning perception. Further, the subjective results clearly demonstrate that novices can use Raptor to ease the difficulty of OOP and in enhancing and understanding complex OOP concepts. Despite the fact that the sample size was too small to achieve statistical significance, these quantitative and qualitative results

provide the positive impact for implementing Raptor in future. They demonstrate that Raptor positively impacts easing the difficulty of OOP through its visualisation, ease of use and creating a more enhanced learning experience for novices. Raptor may be applicable for the purpose of teaching DE novices how to design algorithms, understand concepts and ease the difficulty of OOP.

Raptor carefully planned within the context of a Java course curriculum at various institutions could result in an enhanced learning experience of novice programmers. Furthermore a more structured tutorial from the beginning of the semester that will be in correlation with the course can be implemented in order to improve on the self-learning aspect of the platform and ensure students have sufficient time and guidance is provided. The findings of this research show that by providing an effective, appealing e-learning experience through integration with the Raptor tool, can lead to enhanced learner interactivity, personalisation and engagement of students. Furthermore the results indicated that the visual nature of Raptor assisted in enhancing understanding of how computer programs operate and was beneficial in easing the difficulty of OOP. The findings reported in the study justify the support of the Raptor visualisation tool in engaging individuals in order to enhance their learning experience to ease OOP difficulties experienced among novices and to enhance OOP concept understanding.

Raptor perceptions that impact on DE have proved to be an effective tool in creating more accessible environments that are independent of time, pace, place or physical location, and can be customised to students' needs. As e-learning technology is motivated by learner needs, it can provide learning opportunities suited to learner interests and needs; therefore, it can lead to an enormous potential to stimulate individuals' motivation and development in creating a more positive and engaging learning experience to overcome barriers in programming and enhance concept understanding to address diverse needs of students in DE that could lead to a reduced dropout rate.

LIST OF REFERENCES

Abdelhakim, M.N.A. & Shirmohammadi, S. (2007) A web-based group decision support system for the selection and evaluation of educational multimedia. In: *Proceedings of the international workshop on Educational multimedia and multimedia education* (*EMME'07), Germany, 28 September 2007*. New York: ACM pp. 27-36.

Baldwin, R.G. (2007) *Learn to Program using Alice* [WWW]*. Class-Level Methods and Inheritance.* Available from: http://www.dickbaldwin.com/alice/Alice0150.htm [Accessed: 01 June 2015].

Ben-Ari, M., Ragonis, N. & Ben-Bassat Levy, R. (2002) A Vision of Visualization in Teaching Object-Oriented Programming. In: *Proceedings of 2nd Program Visualization Workshop.* HornstrupCentret, Denmark, pp. 83-89.

Bednarik, R., Moreno, A. & Myller, N. (2006) Various Utilizations of an Open-Source Program Visualization Tool, Jeliot 3. *Informatics in Education,* 5(2), pp.195–206.

Bijlani, K., Jayahari, K.R. & Mathew, A. (2011) A view: Real-time multimedia E-Learning. In: *Proceedings of the third international ACM workshop on Multimedia technologies for distance learning MTDL'11, Arizona,1 December 2011*. New York: ACM, pp.13-18.

Bouvier, D., Chen, T.Y., Lewandowski, G., McCartney, R., Sanders, K. & VanDeGrift, T. (2012) User interface evaluation by novices. In: *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education ITiCSE'12, Israel, 3-5 July 2012.* New York: ACM, pp. 327-332.

Brinkman, W.P., Payne, A., Patel, N., Griggin, D. & Underwood, J. (2007) Design, use and experience of E-Learning systems. In: *Proceedings of the 21st BCS HCI Group Conference HCI 2007, UK, 3-7 September 2007*. UK, British Computer Society pp. 201-202.

Brown, W. (n.d.(a)) *Introduction to programming with Raptor* [WWW]. Available from: http://www.raptor.martincarlisle.com [Accessed: 07 June 2015].

Brown, W. (n.d.(b)) *Lesson 6 – Introduction to Algorithmic Thinking.* Available at: http://www.raptor.martincarlisle.com [Accessed: 07 June 2015].

Brown, W. (n.d.(c)) *Programming Control Structures.* Available at: http://www.raptor.martincarlisle.com [Accessed: 07 June 2015].

Carlisle, M.C. (2009) RAPTOR: A visual programming environment for teaching object-oriented programming. *Journal of Computing Sciences in Colleges JCSC*, 24 (4), pp.275-281.

Carlisle, M.C., Wilson, T.A., Humphries, J.W. & Hadfield, S.M. (2005) RAPTOR: A Visual Programming Environment for Teaching Algorithmic Problem-Solving. In: *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education SIGCSE'05, USA, 23-27 February 2005*. New York: ACM, pp. 176-180.

Carlisle, M.C, Wilson, T.A., Humphries, J.W. & Hadfield S.M. (2004) RAPTOR: Introducing Programming to Non-Majors with Flowcharts. *Journal of Computing Sciences in Colleges,* 19 (4), pp.52-60.

Cooper, S., Nam, Y.J. & Si, L. (2012) Initial Results of Using an Intelligent Tutoring System with Alice. In: *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education ITiCSE'12, Israel, 3-5 July*. New York: ACM, pp.138-143.

Creswell, J.W. (2009). *Research design: qualitative, quantitative, and mixed methods approaches*. Los Angeles: Sage.

Creswell, J.W. & Clark, V.L.P. (2011). *Designing and conducting mixed methods research*. Los Angeles: Sage.

Dillon, E., Anderson, M. & Brown, M. (2012) Comparing feature assistance between programming environments and their "effect" on novice programmers. *Journal of Computing Sciences in Colleges JCSC*, 27 (5), pp. 69-77.

Dorn, B. & Sanders, D. (2003) Using Jeroo to introduce object-oriented programming. In: *Proceedings of 33rd ASEE/IEEE Frontiers in Education Conference, Boulder, CO, 5-8 November 2003*. pp. 22-27.

Duan, X. & Jiang, P. (2008) Research of a virtual 3D study pattern based on constructive theory. In: *Proceedings of the 1st international conference on Forensic applications and techniques in telecommunications, information, and multimedia and workshop e-Forensics '08, Australia, 21-23 January 2008.* Belgium: ICST.

Esteves, M., Fonseca, B., Morgado, L. & Martins, P. (2009) Journal of Virtual Worlds Research – Using Second Life for Problem Based Learning in Computer Science Programming. *Pedagogy, Education and Innovation in 3-D Virtual Worlds*, 2 (1), pp. 1-25.

Fletcher, R. & Guillen, R. (2012) Sample courseware for introductory OO Programming. In: *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education ITiCSE '12, Israel 3-5, July 2012*. New York: ACM, pp. 370-370

Gallant, R.J. & Mahmoud, Q.H. (2008) Using Greenfoot and a Moon Scenario to Teach Java Programming in CS1. In: *Proceedings of the 46th Annual Southeast Regional Conference on XX ACM-SE 46, USA, 28-29 March2008.* New York: ACM, pp. 118-121.

Georgantaki, S. & Retalis, S. (2007) Using educational tools for teaching OO programming. *Journal of information technology impact JITI,* 7(2), pp. 111-130.

Gestwicki, P.V. (2004) Interactive Visualization of Object-Oriented Programs. *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications OOPSLA '04, Canada, 24-28 October 2008.* New York: ACM, pp. 48-49.

Giordano, J.C. & Carlisle, M. (2006) Toward a More Effective Visualization Tool to Teach Novice Programmers. In: *Proceedings of the 7th conference on Information technology education SIGITE '06, USA, 19-21 October 2006.* New York: ACM, pp. 115-121.

Gomes, A., Santos, A. & José Mendes, A. (2012) A Study on Students' Behaviours and Attitudes towards Learning to Program. In: *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education ITiCSE '12 Israel, 3-5 July 2012*. New York: ACM, pp. 132-137.

Govender, D.W. & Govender, I. (2012) Are students learning object-oriented programming in object-oriented programming course? Student voices, *ITiCSE'12*395. *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education ITiCSE '12, Israel, 3-5 July 2012.* New York: ACM , pp 395-395.

Graven, O.H. & MacKinnon, L. (2009) Prototyping Games-Based Environments for learning C++ programming. In: *Proceedings of Human-Computer Interaction HCI Educators 2009 - Playing with our education, UK, 22 - 24 April.* pp. 32-39.

Haupt, M., Perscheid, M., Hirschfeld, R. & Kessler, L. (2010) PhidgetLab: Crossing the border from virtual to real world objects. In: *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education ITiCSE '10, Turkey, 26-30 June 2010. .* New York: ACM, pp. 73-77.

Henriksen, P. (2006) Greenfoot Demonstration. In: *ITICSE '06 Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education, Italy, 26-28 June 2006.* New York: ACM, pp. 355

Henriksen, P., & Kölling, M. (2004) Greenfoot: Combining Object Visualization with Interaction. In: *Proceedings companion of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '04), Canada, 24-28 October 2204*. New York: ACM, pp. 73-82.

Howard, L., Johnson, J. & Neitzel C. (2010) Reflecting on online learning designs using observed behaviour. In: *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education ITiCSE '10, Turkey, 26-30 June 2010.* New York: ACM, pp. 179-183.

Hundhausen, C.D. & Brown, J.L. (2005) Personalizing and Discussing Algorithms within CS1 Studio Experiences: An Observational Study. In: *Proceedings of the first international workshop on Computing education research ICER '05, USA, 1-2 October 2005.* New York: ACM, *pp. 45-56*

Hundhausen, C.D., Farley, S. & Brown, J.L. (2006) Can Direct Manipulation Lower the Barriers to Programming and Promote Positive Transfer to Textual Programming? An Experimental Study. *ACM Transactions on Computer-Human*

Interaction (TOCHI), 16(3), pp. 157-164.

Hundhausen, C.D, Narayanan, N.H. & Crosby, M.E. (2007) Exploring Studio-based instructional models for computing education. In: *Proceedings of the 39th SIGCSE technical symposium on Computer science education SIGCSE '08, USA, 7 September 2007.* New York: ACM, pp. 392-396

International Organization for Standardization (ISO). (1998) *ISO 9241-11: 1998: ergonomic requirements for office work with visual display terminals (VDTs) - part 11: guidance on usability*. Geneva, Switzerland, International Organization for Standardization.

Ivanov, S. & Peneva, J. (2007) Distance learning course in computer science – initiation and design. In: *Proceedings of International Conference on Computer Systems and Technologies CompSysTech'2007, place, 14-15 June 2007.* New York: ACM, pp. IV.8-1-IV.8-6.

Jourjon, G., Kanhere, S. & Yao, J. (2011) Impact of an e-learning platform on CSE lectures. In: *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education ITiCSE '11, Germany, 27-29 June 2011.* New York: ACM, pp. 83-87.

Kölling, M. (2008) Greenfoot – A Highly Graphical IDE for Learning Object-Oriented Programming. *In: Proceedings of the 13th annual conference on Innovation and technology in computer science education ITiCSE '08, Spain, 30 June–2July.* New York: ACM, pp. 327-327.

Lahtinen, E., Ala-Mutka, K. & Jarvinen, H.M. (2005) A study of difficulties of novice programmers. In: *Proceedings of the 10th annual SIGCSE conference on innovation and technology in computer science education ITiCSE '05, Portugal, 27-29 June 2005*. New York: ACM, pp. 14-18.

Li, Q., Lau, R.W.H., Shih, T.K. & Li, F.W.B. (2008) Technology supports for distributed and collaborative learning over the internet. *ACM Transactions on Internet Technology (TOIT),* 8(2), pp. 10.1-10.24.

Li, F.W.B. & Watson, C. (2011) Game-Based Concept Visualization for Learning Programming. In: *MTDL '11 Proceedings of the third international ACM workshop on Multimedia technologies for distance learning, USA, 1 December 2011.* New York: ACM ,pp. 37-42.

Matthews, P., Hin, H.S. & Choo, K.A. (2009) Multimedia Learning Object to Build Cognitive Understanding in Learning Introductory Programming. In: *Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia MoMM '09, Kuala Lumpur, 14-16 December 2009.* New York: ACM, pp. 396-400.

Mehdi, M. & Feiznia, A.F. (2011) Issues in E-Learning quality assurance. In: *Proceedings of the Second Kuwait Conference on e-Services and e-Systems KCESS '11, 5-7 April 2011.* New York: ACM.

Minton, L., Boyle, R. & Dimitrova, V. (2004) If diversity is a problem could e-learning be part of the solution? A case study. In: *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education ITiCSE '04, United Kingdom, 28-30 June 2004.* New York: ACM, pp. 42-46.

Moreno, A., Myller, N., Sutinen, E. & Ben-Ari, M. (2004) Visualizing Programs with Jeliot 3. In: *Proceedings of the Advanced Visual Interfaces AVI '04, Italy, May 25-28*. New York: ACM, pp. 373–376.

Moritz, S.H., Wei, F., Parvez, S.M. & Blank, G.D. (2005) From Objects-First to Design-First with Multimedia and Intelligent Tutoring. In: *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education ITiCSE '05, Portugal, 27-29 June 2005* . New York: ACM, pp. 99-103.

Mouton, J. (2001) *How to succeed in your master's and doctoral studies: A South African guide and resource book*. Pretoria: Van Schaik.

Mussai, Y. & Liberman, N. (2012) An animation as an illustrate tool for learning concepts in OOP, *ITiCSE'12,* 386. In: *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education ITiCSE '12, Israel, 3-5 July 2012.* New York: ACM, pp. 386-386

Myller, N. & Nuutinen, J. (2006) JeCo: Combining Program Visualization and Story Weaving Department of Computer Science. *Informatics in Education*, 5(2), pp. 255-264.

Olivier, M.S. (2009) *Information technology research: a practical guide for computer science and informatics*. 3rd ed. Pretoria: Van Schaik.

Parker, M. (2003) Technology enhanced e-learning: Perceptions of first year's information systems students at Cape Technikon. In: *Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology SAICSIT '03.* South Africa: South African Institute for Computer Scientists and Information Technologists, pp. 316-319.

Piccolli, G., Ahmad, R. & Ives, B. (2001) Web-based virtual learning environments: A research framework and a preliminary assessment of effectiveness in basic IT skills training, *MIS Quarterly,* 25(4), pp. 401-426.

Radošević, D., Orehovački, T. & Lovrenčić, A. (2009) New Approaches and Tools in Teaching Programming. In: *Proceedings of the 20th Central European Conference on Information and Intelligent Systems, CECIIS* 2009, Croatia, *23-25 September 2009. pp*. 49-57.

Rajala, T., Laakso, M.K., Kaila, E. & Salakoski, T. (2008) Effectiveness of Program Visualization: A Case Study with the ViLLE Tool. *Journal of Information Technology Education*, 7, pp. 15-32.

Rößling, G., Joy, M., Moreno, A., Radenski, A., Malmi, L., Kerren, A., Naps, T., Ross, R.J., Clancy, M., Korhonen, A., Oechsle, R. & Velázquez, J.A. (2008) Enhancing Learning Management Systems to Better Support Computer Science Education. *ACM SIGCSE Bulletin,* 40(4), pp. 142-166.

Sanders, D. & Dorn, B. (2003) Classroom experience with Jeroo. *Journal of Computing Sciences in Colleges CCSC,* 18(4), pp. 308-316.

Santos, A.L. (2011) AGUIA/J A tool for interactive experimentation of objects. In: *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education ITiCSE'11, Germany, 27-29 June 2011*. pp. 43-47.

Ssemugabi, S. & De Villiers, R. (2007) A comparative study of two usability evaluation methods using a web-based e-learning application. In: *Proceedings of the 2007 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries SAICSIT '07, South Africa, 2-3 October 2007*. New York: ACM, pp. 132-142.

Stephen, M., Franklin, W., Patrick, O., Peter, A. & Elizabeth, A. (2012) Classifying Program Visualization Tools to Facilitate Informed Choices: Teaching and Learning Computer Programming. *International Journal of Computer Science and Telecommunications,* 3(2), pp. 42-48*.*

Torrente, J., Del Blanco, A., Moreno-Ger, P., Martinez-Ortiz, I. & Fernandez-Manjon, B. (2009a) Implementing accessibility in education video games with <e-Adventure>. In: *Proceedings of the first ACM international workshop on Multimedia technologies for distance learning MTDL'09, China, 23 October 2009*. New York: ACM, pp. 57-66.

Torrente, J., Moreno-Ger, P., Martínez-Ortiz, I. & Fernandez-Manjon, B. (2009b) Integration and Deployment of Educational Games in e-Learning Environments: The Learning Object Model Meets Educational Gaming. *Educational Technology & Society*, 12 (4), pp. 359-371.

Truong, N., Bancroft, P. & Roe, P. (2005) Learning to Program Through the Web, In: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education ITiCSE '05*, Portugal, 27-29 June 2005*. New York: ACM, pp. 9-13.

UNISA. (2012). [Online] Available from: *http://www.unisa.ac.za/default.html* . [Accessed: 2012]

Vilner, T., Zur, E. & Tavor, S. (2011) Integrating Greenfoot into CS1 – A case study, In: *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education ITiCSE'11, Germany, 27-29 June 2011*. New York: ACM, pp. 350-350.

Vrasidas, C. (2004) Issues of pedagogy and design in e-Learning systems. In: *Proceedings of the 2004 ACM symposium on Applied computing SAC '04, Cyprus, 14-17 March 2004*. New York: ACM, pp. 911-915

Wang, Y.D. (2011). Teaching web development at a distance. In: *Proceedings of the 2011 conference on Information technology education SIGTE'11, USA, 20-22 October 2011*. New York: ACM, pp. 91-96.

Wei, F., Moritz, S.H., Parvez, S.M. & Blank, G.D. (2005) A student model for object-oriented design and programming. *Journal of Computing Sciences in Colleges archive*, 20(5), pp. 260-273.

Weissman, E.J. (2002) An evaluation of online learning environments (OLE) on the adult-at-risk population. *Journal of Computing Sciences in Colleges*, 18(3), pp. 142-154.

Wells, J. (2000) Effects of an online computer-mediated communication course, prior computing experience and internet knowledge, and learning style on student's internet attitudes computer-mediated technologies and new educational challenges. *Journal of Industrial Teacher Education*. Available from: http://scholar.lib.vt.edu/ejournals/JITE/v37n3/wells.html.

Wolf, C. (2012) iWeaver: Towards 'Learning Style'-based e-Learning in Computer Science Education. In: *Proceedings of the fifth Australasian conference on Computing education ACE '03.* Australia: Australian Computer Society, pp. 273-279

Yukselturk, E., Yazici, A., Sacan, A. & Kaya, O. (2010) Online education experiences: Information Technologies Certificate Program at METU. In: *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education ITiCSE'10, Turkey, 26-30 June 2010*. New York: ACM, pp. 83-87.

Zhang, D., Zhao, J.L., Zhou, L. & Nunamaker, J.F. (Jnr.) (2004) Can E-Learning replace classroom learning? *Communications of the ACM,* 47(5), pp. 75-79.

APPENDIX A

LETTERS OF CONSENT TO CONDUCT THE RESEARCH STUDY

Mrs Saadia Essa (30965799)
School of Computing
UNISA
Pretoria

2013-08-06

Permission to conduct research project

Ref: 081/SE/2013

The request for ethical approval for your MSc (Computing) research project entitled "The impact of an e-learning tool support to overcome difficulties in learning Object Orientated Programming." refers.

The College of Science, Engineering and Technology's (CSET) Research and Ethics Committee (CREC) has considered the relevant parts of the studies relating to the abovementioned research project and research methodology and is pleased to inform you that ethical clearance is granted for your study as set out in your proposal and application for ethical clearance.

Therefore, involved parties may also consider ethics approval as granted. However, the permission granted must not be misconstrued as constituting an instruction from the CSET Executive or the CSET CREC that sampled interviewees (if applicable) are compelled to take part in the research project. All interviewees retain their individual right to decide whether to participate or not.

We trust that the research will be undertaken in a manner that is respectful of the rights and integrity of those who volunteer to participate, as stipulated in the UNISA Research Ethics policy. The policy can be found at the following URL:
http://cm.unisa.ac.za/contents/departments/res_policies/docs/ResearchEthicsPolicy_apprvCounc_21Sept07.pdf

Please note that if you subsequently do a follow-up study that requires the use of a different research instrument, you will have to submit an addendum to this application, explaining the purpose of the follow-up study and attach the new instrument along with a comprehensive information document and consent form.

Yours sincerely

Chair: School of Computing Ethics Sub-Committee

University of South Africa
College of Science, Engineering and Technology
Preller Street, Muckleneuk Ridge, City of Tshwane
PO Box 392 UNISA 0003 South Africa
Telephone + 27 12 429 6122 Facsimile + 27 12 429 6848
www.unisa.ac.za/cset

PROF L LABUSCHAGNE
EXECUTIVE DIRECTOR: RESEARCH DEPARTMENT
***Tel**: +27 12 429 6368 / 2446* ***Fax**: +27 12 429 6960*
***Email**: llabus@unisa.ac.za*
***Address**: Theo van Wijk Building, 10th Floor, Office no. 50 (TvW 10-50)*

16 May 2014

Ms SF Essa
School of Computing
College of Science, Engineering and Technology

Dear Ms Essa

PERMISSION TO DO RESEARCH INVOLVING UNISA STAFF, STUDENTS OR DATA

A study into "The impact of an E-Learning tool support to overcome difficulties in learning Object Orientated Programming"

Your application regarding permission to conduct research involving Unisa staff, students or data in respect of the above study has been received and was considered by the Unisa Senate Research and Innovation and Higher Degrees Committee (SRIHDC) on 17 April 2014.

It is my pleasure to inform you that permission has been granted for this study as set out in your application.

We would like to wish you well in your research undertaking.

Kind regards

PROF L LABUSCHAGNE
EXECUTIVE DIRECTOR: RESEARCH

University of South Africa
Preller Street, Muckleneuk Ridge, City of Tshwane
PO Box 392 UNISA 0003 South Africa
Telephone: +27 12 429 3111 Facsmile: +27 429 12 429 4150
www.unisa.ac.za

APPENDIX B

SURVEY

BACKGROUND: DEMOGRAPHIC INFORMATION:
1. University currently registered:
2. Course Name:
3. Course Code:
4. Year of study:
5. Occupation:
6. Age: ○ 18-20 yrs ○ 21-23 yrs ○ 24-26 yrs ○ 27-30 yrs ○ 30 and above
7. Gender: ○ Female ○ Male

QUANTITATIVE QUESTIONS					
Statement	**Strongly Agree**	**Agree**	**Neutral**	**Disagree**	**Strongly Disagree**
"User interface factor":					
Category 1: Perceived usability of tool					
Effectiveness: defined as the correctness and comprehensiveness with which users accomplish specified objectives (ISO, 1998).					
Using the Raptor tool in my studying would enable me to accomplish program tasks more quickly and efficiently.	Using the tool in distance education learning process would be useful in assisting to increase productivity and effectiveness in studying.				
I think the tool would be useful for novices and more advanced users.	The tool was flexible in its ability to accommodate different user levels, from novice to more advanced user levels.				
The tool has a simple and easy-to-use navigation structure.	Tool required minimal user control actions to accomplish the task. The tool was flexible to interact with enabling shortcuts to accelerate completing a task by expert users.				
Efficiency: defined as resources used in relation to the correctness and comprehensiveness with which users achieve objectives (ISO, 1998).					
Using the Raptor tool in my studying would enable me to accomplish program tasks more easily and effectively.					
It was easy to use the tool to create flowcharts.					
It was easy for me to run Raptor program flowcharts					

once created.	
It would be easy for me to become skilful at learning to operate the Raptor tool to complete program tasks effortlessly and confidently.	
Learnability: user ability to use the tool interface correctly in order to accomplish a task (Bouvier et al., 2012).	
The symbols (assignment, call return, input, output, selection and loop) used in Raptor tool are easily understood and meaningful within the perspective of learning object-oriented programming tasks.	Metaphors and language correspond to the real-world daily environment, so they are understandable and meaningful. Consistency in its representation of symbols and actions.
There was no need to recall/remember information and instructions, as it was retrievable and visible whenever appropriate.	
I found Raptor to be an effective design tool to enhance algorithmic thinking of programming tasks without the need to consider the syntax of the programming language.	Raptor is an effective design tool to improve problem-solving skills.

Error-recognition-diagnosis-recovery cycle and support information: interface recovers from errors and undesirable conditions (Bouvier et al., 2012).	
Once Raptor flowchart was created, it was easy for me to generate the corresponding Java code.	
Raptor displays appropriate error messages that clearly inform me about the error in plain language.	Recognition of appropriate error messages that clearly informs me about the error.
I am able to recover quickly and easily from errors given through precise, simple, efficient and effective error messages.	Ability to recover quickly and easily from errors given through precise, simple, efficient and effective instructions.
"User interface factor":	
Category 2: User experience	
Affect: the subjective quality of how interacting with an interface feels to the user (Bouvier et al., 2012).	
The information (such as online help, on-screen messages, and other documentation) provided with this tool was useful, clear and easy to understand in order for me to effectively create programs and flowcharts.	Help system in Raptor is effective and useful. The support documentation was appropriate and presented in a structured, well-organised manner. The support information was effective in assisting to effectively create programs and flowcharts. Easily searchable ability with clearly defined steps to complete a task.
I would recommend the tool to others.	Implying both occasional and regular users would like it.

Visualisation appearance: relates to general user experience (Bouvier et al., 2012).	
The graphical user interface of the tool is pleasant and user-friendly.	Look and feel of tools was good.
Learner motivation and interactivity: content and interactive features (Bouvier et al., 2012).	
It was a challenging, engaging and stimulating experience learning to program with Raptor.	Interaction with the tool was clear and understandable through constructive, appropriate and timely feedback informed about the visibility of the system. It was an interactive experience. The illustrations provided and interactions were used at the right level to enable effectiveness in studying through DE learning process.
Utilising the Raptor tool increased my course workload.	
The duration of experimenting and learning with the Raptor tool was at the right pace.	
Category 3: "Learning perception"	
Learner control: with respect to time, place, content and sequence of learning (Ssemugabi & De Villiers, 2007).	
I am satisfied with the self-controlled learning with respect to time, place, content and sequence of learning.	Flexible to customise personal approaches to learning according to learner ability and styles. Supports various ways or learning styles. I am satisfied with the self-controlled learning process, so taking control of the learning process with respect to time place, content and sequence of learning.
I enjoyed programming with the Raptor tool.	The tool enhances your feelings about learning to program.

Learning capability of tool: as an effective teaching capability of tool, it refers to both concept understanding and self-learning tool capability (Jourjon et al., 2011).	
I performed well on the program exercises and examples in the object-oriented tutorials.	
There was sufficient feedback on the activities and knowledge construction guiding me as I created program flowcharts.	There was sufficient feedback of activities and knowledge construction; thus, task guidance was provided as students created program flowcharts.
The tool helped to enhance understanding and reinforced key object oriented concepts (inheritance and polymorphism).	
I was able to effectively utilise the Raptor tool as a self-learning tool.	Effective teaching capabilities of the tool.
As I was able to view contents of variables, it assisted me in testing and debugging programs.	Helped to improve problem-solving skills. The tool improved knowledge and understanding of the subject of how OOP operates.
Raptor tool made Java programming easier.	
Once the design in Raptor was implemented, generating the Java program would be easy for me.	
I felt confident about taking what I learnt from this tool and applying it to my course.	Tool supports correlation with course goals.

Raptor assisted in enhancing my understanding of how computer programs operate.	
QUALITATIVE QUESTIONS	
To what extent do you consider the tool environment contributed to overcoming difficulties in object-oriented programming?	
Do you have any previous object-oriented programming experience? If yes, please give details.	
Additional comments:	

APPENDIX C

SURVEY QUESTION 31 – QUALITATIVE QUESTION 1

Respondent	Response	Categorised Data
Response 1	I have to master the tool nicely in order to determine.	More time was needed for becoming familiar with using Raptor.
Response 2	Great deal.	Tool was beneficial in easing the difficulty of OOP but was not specific.
Response 3	It helped in understanding how inheritance and polymorphism worked as well as with aid in the visual representation of the logical flow of arguments.	Raptor assisted in understanding the fundamentals concepts of inheritance and polymorphism. The visual representation enabled understanding of the logical flow of the programs.
Response 4	Instead of only words, one gets to work with diagrams that show the interrelation of things. I think given more time, I can get to understand the program better.	The visual representation enabled understanding of the logical flow of the programs. More time was needed for becoming familiar with using Raptor.
Response 5	It makes it more visual than just written words and algorithms.	The visual nature of Raptor was beneficial. The graphical user interface of the tool is pleasant and user-friendly.
Response 6	It simplified my thinking through graphics and, as a result, understood object oriented easier.	The graphical user interface of the tool was simple and easy to use, resulting in enhanced understanding of OOP.

Response 7	Honestly, I don't think there will be any difficulties if the author can at least list few steps on how to start an activity for people who are not familiar with algorithm and flowcharts. I understand the language of flowcharts and algorithm, so for me, it was easy to apply my mind and play around with the tool. I like it.	More direction was required to complete the tutorial using the Raptor tool, particularly for those with minimal experience in flowcharts and algorithms. Student enjoyed programming with the Raptor tool.
Response 8	I believe in my own personal capacity it would make programming interesting and make problem-solving quickly, especially with flowcharts.	It was easy to use the tool to create flowcharts assisting in creating a stimulating programming experience and ability to solve problems efficiently.
Response 9	I think it would be better if there would be some classes for programming-related modules.	Students suggested that workshops to familiarise themselves with Raptor would be beneficial.
Response 10	It's easier.	Tool was beneficial in easing the difficulty of OOP but was not specific.
Response 11	In programming languages.	Tool was beneficial in easing the difficulty of OOP but was not specific.
Response 12	To a greater extent.	Tool was beneficial in easing the difficulty of OOP but was not specific.
Response 13	Fair	Tool was moderately beneficial in easing the difficulty of OOP but was not specific.
Response 14	The program is user-friendly. It can help you reinforce your Java or Eclipse codes more easily.	The graphical user interface of the tool is pleasant and user-friendly. Raptor tool made Java programming easier.
Response 15	The tool is perfect for people learning how to code. The visuals of the application help to contribute in helping to understand how an	The tool would be most appropriate for novice programmers. The visual nature of Raptor assists in

	application executes.	enhancing understanding of how computer programs operate.
Response 16	When there are logic errors.	The visual representation enabled understanding of the logical flow of the programs, so it assisted in identifying logic errors.
Response 17	It contributed 50%.	Tool was moderately beneficial in easing the difficulty of OOP but was not specific.
Response 18	I think the Raptor tool should be considered within the syllabus. It would be a great experience mostly for new students.	Raptor should be implemented in the course. The tool would be beneficial for novice programmers.
Response 19	Raw Java coding can be very scary just by looking at it, but the Raptor tool eliminates that code fear because immediately when you see tons of code, you begin to feel intimidated and to think that you cannot do it. But with the Raptor tool, programming becomes easier because you deal directly with shapes, playing around with them to achieve your intended results and create Java code after. The tool is very important because it teaches and trains your mind to understand the concept of programming because if you understand the concepts, then code implementation becomes easier.	The graphical user interface of the tool is pleasant and user-friendly. Raptor assisted in easing Java programming through its visual nature. The tool helped to enhance understanding and reinforced key object oriented concepts.
Response 20	Simple and easy to use and navigate.	The tool has a simple and easy-to-use navigation structure.
Response 21	-	No comment
Response 22	It provides a good visual aid for programming that helps improve logic	The graphical user interface and visual nature of the tool

	and program flow.	are pleasant and user-friendly, enabling one to solve problems and follow the logical flow of programs easily.
Response 23	It gave me a good understanding of classes, objects and methods that will be useful to me while developing my Android application in ICT2612. I think it will be very useful when creating complex algorithms.	Raptor assisted in enhancing understanding and reinforced key object oriented concepts. Tool supports correlation with course goals. Raptor is an effective design tool to enhance algorithmic thinking of programming tasks.
Response 24	It made learning easy, as everything was explained in greater detail.	Tool was beneficial in easing the difficulty of OOP.
Response 25	It has been very helpful in OOP.	Tool was beneficial in easing the difficulty of OOP but was not specific.
Response 26	It made my work easier, and I passed my module.	Tool was beneficial in easing the difficulty of OOP. Tool-supported correlation with course goals.
Response 27	This program is very good in terms of overcoming difficulties in regard to coding. It is, however, good for beginners, but on the other hand, I think future programmers will lack certain skills for coding because Raptor almost does everything for you; it gives you the output to notepad. From my point of view, I think theory should be introduced first before introducing Raptor so that individuals can understand the whole process. I'm very pleased that it is however offloading work!	Tool was beneficial in easing the difficulty of OOP. The tool would be most appropriate for novice programmers. Student suggested introducing theoretical concepts initially before implementing Raptor. The visual nature of Raptor is helpful and makes it easier to create programs.
Response 28	Honestly, I never used Raptor in my studies, unless if it has another name, maybe I used it without knowing.	No comment
Response 29	Frankly, I haven't used the tool or	No comment

	been introduced to it. Thank you, I now know about it, and I am going to start using it. Please remember that my responses are not honest because I have no experience of the Raptor tool.	
Response 30	The tool made it easier and easier to understand.	Tool was beneficial in easing the difficulty of OOP but was not specific.
Response 31	No comment	No comment
Response 32	Graphic user interface	The graphical user interface of the tool is pleasant and user-friendly.
Response 33	I have never used it.	No comment
Response 34	The tool environment helps with understanding and visualising OOP concepts and is great for learning.	Raptor assisted in enhancing understanding and reinforced key object oriented concepts through its visual nature. The tool was beneficial in easing the difficulty of OOP.
Response 35	It helps with learning to program.	Tool was beneficial in easing the difficulty of OOP but was not specific.
Response 36	It helped me to understand difficult programs that were not easy for me before. It helps one to go step by step through the program.	Tool was beneficial in easing the difficulty of OOP as well as the ability to solve problems easily. The visual representation enabled understanding of the logical flow of the programs.
Response 37	The tool made it easier for me to learn and structure my studies in Java development and thus eased my ability to learn the basics of the course workload.	Raptor assisted in easing Java programming, reinforcing fundamental concepts within course goals. Thus, the tool supported correlation with course goals.
Response 38	It contributed a lot; it made object-oriented programming much easier than before.	Tool was beneficial in easing the difficulty of OOP but was not specific.

Response 39	The Raptor tool helped considerably in overcoming “challenges”. The tool is easy to understand and easy to use. I felt that the user interface could do with a few cosmetic upgrades, as that would attract new users more successfully.	The tool was beneficial in easing the difficulty of OOP. The tool was simple and easy to use. Student suggested improving on the GUI to attract more users.
Response 40	Have not used it.	No comment

APPENDIX D

SURVEY QUESTION 32 – QUALITATIVE QUESTION 2

Respondents	Response	Categorised Data
Response 1	Not much because during my studies, it was more theory than practical.	Some experience
Response 2	I've done few ICT2622, which was about object-oriented analysis; it was a bit challenging especially when creating the diagrams.	Experienced
Response 3	No	No experience
Response 4	I have done C++, JavaScript and Visual Basic (VB) 6.	Experienced
Response 5	No only what I learnt this semester with Java.	No experience
Response 6	I did VB in college and also C++ through UNISA and technikon.	Experienced
Response 7	I do not have experience in object-oriented programming, but I do understand the concepts and the design of the OOP and flowchart and how it works with methods, classes, inheritance, etc.	No experience
Response 8	No	No experience
Response 9	No	No experience

Response 10	I did a project using JavaScript.	Experienced
Response 11	Yes, in JavaScript and Introduction to Programming.	Experienced
Response 12	Yes, Visual Basic and Pascal programming.	Experienced
Response 13	No	No experience
Response 14	No	No experience
Response 15	This question is very open-ended, but I will try. Yes, I have been developing without a qualification for 20+ years. Back in the days, almost all banking systems were in COBOL, and some are still today. My primary OOP language today for work is mostly C++, C# and Objective-C, with private development in Python.	Experienced
Response 16	Yes, I use it more often at work and when modifying my web page (blogger).	Experienced
Response 17	No	No experience
Response 18	I have done Information Technology at high school (grade 10-12). If I remember correctly, we did a little bit of it as a chapter.	Some experience.
Response 19	I have spent much time on Visual Basic .NET learning object-oriented programming concepts and how it differs with other languages, but I think Java is the advanced version of OOP, and one of its advantages is that it is platform-dependent, unlike VB.NET. I have also spent much time on internet blogs and websites seeking answers to questions, which were laid before me by curiosity and the eagerness to learn more.	Experienced

Response 20	No	No experience
Response 21	-	No experience
Response 22	Yes. I've learnt various other programming languages such as C#. I like to mostly program database-oriented applications in conjunction with SQL.	Experienced
Response 23	Not applicable	No experience
Response 24	No	No experience
Response 25	Yes, I have basic experience, and I am currently gaining further experience about OOP in different learning institutions.	Experienced
Response 26	No	No experience
Response 27	Java! I'm still getting around Java, and yes, Raptor makes things when you use Java, because with Java, you almost input every single code, but with Raptor, you can quickly generate the code.	No experience
Response 28	Not really experience per say, but theory lessons. I have studied OOP in some of my modules.	Some experience
Response 29	I am a beginner.	No experience
Response 30	No	No experience
Response 31	No	No experience
Response 32	No	No experience

Response 33	No	No experience
Response 34	No	No experience
Response 35	Yes and No. Or just a little.	Some experience
Response 36	No	No experience
Response 37	No, this was the first time I have used and covered object-oriented programming.	No experience
Response 38	Yes, I did graphical user interface programming in 2004 (Visual Basic).	Experienced
Response 39	Yes, I do. I have programmed a fully functional website. At the moment, I am programming an Android app.	Experienced
Response 40	No! I'm still learning.	No experience

APPENDIX E

SURVEY QUESTION 33 – QUALITATIVE QUESTION 3

Respondents	Response	Categorised Data
Response 1	I love programming as such, but I should just practise more. Thanks.	Student enjoyed programming with the Raptor tool. More time was needed for becoming familiar with using Raptor.
Response 2	Overall, it would be great to continue using the Raptor tool for programming modules; it would make an IT student's life easier if it is well understood.	The tool can be beneficial in easing the difficulty of OOP.
Response 3	I mainly focus on the infrastructure and build environment in the ICT sector. The Raptor software helped with the understanding of what OOP is and how to lay out the logic and algorithm of a problem.	The tool can be beneficial in easing and understanding OOP. Raptor is an effective design tool to enhance algorithmic thinking of programming tasks.
Response 4	This is an interesting program. It would indeed be of help but as with anything else, a little direction would be in order. Reason being, as with any programming language, it has its own words, its own world that one needs to steer through. Overall, for computing students, this would be easier.	Student enjoyed programming with the Raptor tool, though felt more direction and instruction were required in order to become familiar with using Raptor. The tool can be beneficial in easing the difficulty of OOP.
Response 5	Nice piece of software – could look nicer, though.	Student enjoyed programming with the Raptor tool. Students suggested improving

		the GUI.
Response 6	Keep on creating these groundbreaking technologies.	Student enjoyed programming with the Raptor tool.
Response 7	This is an easy tool that in my opinion UNISA can apply to "Introduction to Programming" course (ICT1511), the basics. If you go through the book ("Basic Programming Principles"), the activities in that book can also be applied using the Raptor tool. It will also help students who do not understand OOP.	Raptor tool is a simple and easy to use. Students suggested that Raptor should be implemented in the course. The tool can be beneficial in easing the difficulty of OOP.
Response 8	I have no additional comments.	No comments
Response 9	I was unable to log in on my blog.	Not applicable
Response 10	It was quite an experience using Raptor; my time was limited, but I was able to read and understand.	Student enjoyed programming with the Raptor tool. More time was needed for becoming familiar with using Raptor.
Response 11	Programming is a good skill for anyone, and I like Raptor.	Student enjoyed programming with the Raptor tool.
Response 12	I think this tool would be useful to programmers as a whole, as it simplifies the process of developing an application.	The tool can be beneficial in easing the difficulty of OOP.
Response 13	Not applicable	Not applicable
Response 14	Is it a must for each and every master's student to do or research this Raptor tool.	Not applicable
Response 15	The Raptor tool is great for adults with any or no programming experience. It is	The tool can be beneficial in easing the difficulty of OOP, particularly among novice

	very useful to see the code layout and choices that some are making even before a line of code is written. A similar tool that I use with my daughter is Scratch (from http://scratch.mit.edu/); it is helpful to teach kids to program without realising that they are learning.	programmers.
Response 16	I am eager to learn more object-oriented.	Student enjoyed programming with the Raptor tool.
Response 17	I will use the tool to do my work and get more experience.	Student enjoyed and will continue to use the Raptor tool to become more familiar with the tool.
Response 18	I recommend the Raptor to be used within the courses.	Student suggested that Raptor should be implemented in the course.
Response 19	I believe the Raptor tool will help students to understand programming without being intimidated by the code or name of the language, since it uses shapes, which everyone can work with. I would also recommend Raptor for novice students, and I believe when the tool is exclusively implemented, it will yield the results expected by the institution.	Raptor assisted in easing the difficulty of OOP through its visual nature. The tool would be beneficial for novice programmers. Raptor should be implemented in the course to ease the difficulty of OOP.
Response 20	None	No comment
Response 21	-	No comment
Response 22	Raptor is a good visual aid of the code, though. You'll still	Raptor assisted in easing OOP through its visual nature.

	need to have a good understanding of the programming syntax to write the code, but all of the logic, sequence and debugging of the program can be done with Raptor. This alone improves time spent troubleshooting program logic while coding. Raptor also makes it easy to explain your logic and even algorithm to other students/fellow programmers without going through various lines of code. I've found that most of the time is spent on logic and layout of a particular application/program by completing the outline of the program; in a Raptor flowchart, you can save a lot of time coding. Plus, I would also like to evaluate the tool based on the amount of resources available to help when faced with a particular problem, and I've found that there are lots of training videos (on YouTube) and documentation available to do just that, i.e. to help and assist with developing Raptor flowcharts.	Testing and debugging programs can be done in the Raptor environment; thus, program tasks can be accomplished more efficiently and effectively. Raptor is an effective design tool to enhance algorithmic thinking of programming tasks without the need to consider the syntax of the programming language, though understanding of syntax would be beneficial. Raptor is an effective design tool to improve problem-solving skills. The additional support documentation provided and available online is appropriate and assisted to effectively create programs and flowcharts.
Response 23	Not applicable	Not applicable
Response 24	It will be a great pleasure for the tool to be implemented in all the programming courses, as it	Student suggested that Raptor should be implemented in the

	makes life easier.	course. The tool can be beneficial in easing the difficulty of OOP.
Response 25	I enjoy working with programming languages.	Not applicable
Response 26	None	No comments
Response 27	I'm worried about skills for certain individuals because we don't have the same level of understanding. As per research, the Raptor program should be in the market too, just so when it is introduced, other companies should be aware that it is useful. I compliment it because we are living in a digital world that almost makes everything easier, so the Raptor does.	The tool can be beneficial in easing the difficulty of OOP and can be used for diverse user levels.
Response 28	Please, next time when you want us students or anyone for that matter to complete any kind of survey, please let it come through to us from maybe our lectures or the university portal or make an announcement that selected students will be chosen to participate in a survey. Reason I am saying this is because honestly, the email I received seemed somehow doggy in a way and the constant reminders to complete the survey really were not necessary at all; those reminders are usually made by those people who scam.	This student suggested instruction to participate in the survey should be instructed through the UNISA portal or through lecturers, not through email, as it seemed to be spam mail.

Response 29	I haven't used the programme, but I believe it is of greater help and use; otherwise, you wouldn't have developed it.	Not applicable
Response 30	Not applicable	Not applicable
Response 31	I recommend Raptor tool.	The tool can be beneficial in easing the difficulty of OOP.
Response 32	No comment	No comment
Response 33	None	No comment
Response 34	I found the fact that I have to make time to learn the system and new ways to make this program work for me to be time-consuming and detracting from my study time.	Utilising the Raptor tool increased course workload.
Response 35	Nope!	No comment
Response 36	It will be helpful to those who are beginning to learn programming, and for those who want to do advanced programming, it is also helpful.	The tool can be beneficial in easing the difficulty of OOP for both novices and more advanced user levels.
Response 37	Not applicable	Not applicable
Response 38	I wish you could improve the program by having more hints as you type the code.	Students suggested improving on the guidance and hints provided with the tool.
Response 39	This is a great tool. With the correct guidelines, I feel that many people would stop looking at programming as such a daunting task. The flowcharts help you understand how the program actually works, thus making programming easier.	The tool can be beneficial in easing the difficulty of OOP with additional guidance. The visual nature of flowcharts in Raptor assists in enhancing understanding of how computer programs operate.
Response 40	I think it would be more beneficial to us (students) if there could be a class or two with the lecturer to help us when such initiatives are rolled out.	Student suggested that workshops to familiarise them with the Raptor tool would be beneficial.

APPENDIX F

DEFINITION OF TERMS

Affect: The subjective quality of how interacting with an interface feels to the user (Bouvier et al., 2012).

Design-first approach: In introductory courses, it involves students learning critical aspects of software development that include object-oriented analysis and design as problem-solving skills (Wei et al., 2005; Moritz et al., 2005).

Distributed learning: This applies to computer and communication technologies, allowing students and facilitators to be involved in learning activities independent of time and place (Li et al., 2008). Furthermore, Li et al. (2008) noted that a distributed learning environment is where resources can be shared and distributed students may learn.

Effectiveness: defined as the correctness and comprehensiveness with which users accomplish specified objectives (ISO, 1998).

Efficiency: defined as resources used in relation to the correctness and comprehensiveness with which users achieve objectives (ISO, 1998).

E-learning: defined as computer or network transfer of skills and knowledge (Duan & Jiang, 2008). According to Mehdi and Feiznia (2011), as an all-embracing term, e-learning encompasses using different media types in the delivery of instructional materials to achieve learning outcomes over the distance learning approach.

Error-recognition-diagnosis-recovery cycle: Interface recovers from errors and undesirable conditions (Bouvier et al., 2012).

Learnability: User ability to use the tool interface correctly in order to accomplish a task (Bouvier et al., 2012).

Learner control: Learner control with respect to time, place, content and sequence of learning (Ssemugabi & De Villiers, 2007).

Learner motivation and interactivity: This refers to content and interactive features (Bouvier et al., 2012).

Learning capability of tool: An effective teaching capability of tool refers to both concept understanding and self-learning tool capability (Jourjon et al., 2011).

Multimedia: It is defined as a "computer program that includes text with at least one of the following; audio/sophisticated sound, music, video, photographs, 3-D graphics, animation or high resolution graphics" (Abdelhakim & Shirmohammadi, 2007:27)

Object-Oriented Programming (OOP): A contemporary approach to computer programming which mimics the real-world environment. The emphasis of this approach is the use of objects, which includes both actions and data. Objects communicate by passing messages (Stephen et al., 2012)

Objects-first: This is an effective approach to teaching object-oriented programming by introducing key concepts of OOP, which emphasise coding (Wei et al., 2005; Moritz et al., 2005).

Usability: This is defined as, "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context", according to ISO (1998).

Visualisation appearance: Relates to general user experience (Bouvier et al., 2012).

APPENDIX G

INFORMED CONSENT FORM

Questionnaire on the perceptions of the Raptor Visual Tool Environment by UNISA distance education students registered for the semester module Interactive Programming (ICT2612) as well students studying Web Development (ICT1513) in the undergraduate Diploma in Information Technology.

This study is being conducted by Saadia F. Essa, who is undertaking a Masters in Computing at the University of South Africa (UNISA). This researcher seeks to determine the impact of an E-Learning tool support environment to overcome difficulties in learning Object-Oriented Programming in distance education. This research will utilise a structured electronic survey to determine learner perceptions of interaction and experience of the Raptor visual learning tool environment, with respect to its usability, as a self-learning tool and whether the tool contributed to overcoming difficulties in OOP. The target application will involve UNISA students registered for the semester module Interactive Programming (ICT2612) as well students studying Web Development (ICT1513) in the undergraduate Diploma in Information Technology. This researcher hopes that by combining technology integration of E-Learning tool support environment and distance education prescribed works, the information that will be obtained from this study will be very useful in assisting students to augment learning resources, to bridge existing knowledge gaps, to reinforce concepts, or to get more suitable explanations in order to overcome difficulties in learning Object-Oriented Programming.

Informed consent

You have been invited to participate in this study, to observe the impact of Raptor tool to overcome difficulties in Object-Oriented Programming. Your perception of working through Raptor Tutorial is important and valuable to us, in determining the impact of Raptor tool in our study.

1. Voluntary participation: You are under no obligation to participate: You may skip any questions you are not comfortable answering. You may also withdraw your participation at any time.

2. Confidentiality: The information produced by this study will be confidential and private. While reporting the results of the study in the dissertation, presentations, reports, or publications, we will not use your name or university name.

3. Benefits: We do not anticipate a direct benefit to you for completing the study. However, you will be providing valuable information that will enable the researcher to complete a master's degree dissertation.

4. Contact: Should you experience any issues with the download and installation or anything else regarding this project, please feel free to contact the researcher (details below).

5. The completion of this will have no effect on the outcomes for this module other than to benefit you in enhancing understanding of Object-Oriented Programming. It will have no effect on your year mark, portfolio or exam mark.

Name and contact details of the researcher

Saadia F. Essa
P.O. Box 1997, 0700
Polokwane
Tel: 0839786866

Name and contact details of the supervisor

Prof. Ian Sanders
University of South Africa
Tel: 011 471 2858

I have read and understood this consent form, and I agree to participate in this study.

_____ _____

Signature Date

Name of the participant

_____

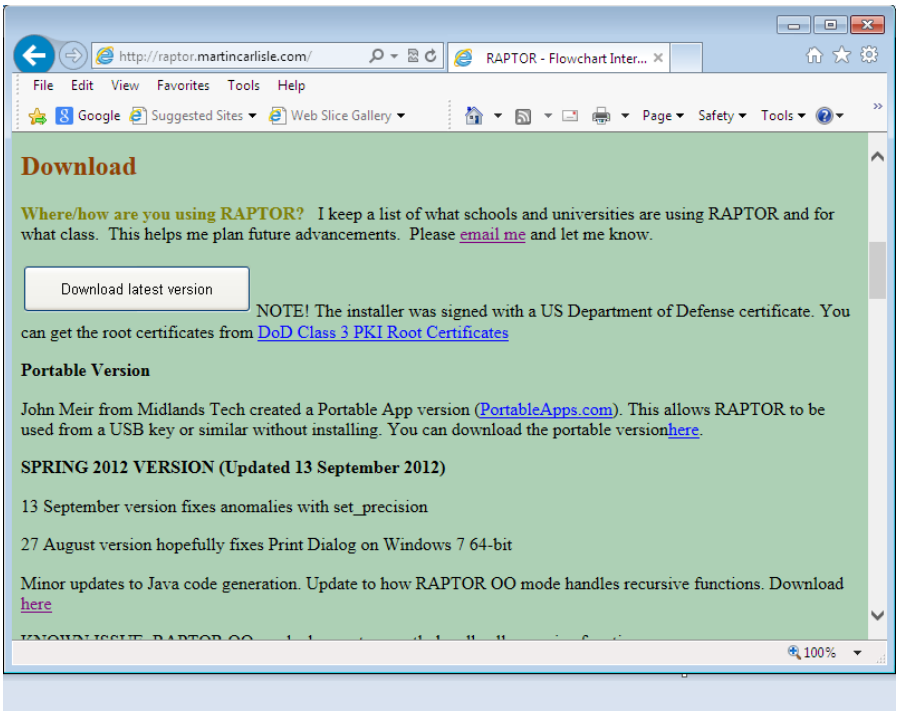
APPENDIX H

RAPTOR TUTORIAL WORK PLAN

The following are the seven steps entailed in the designed Raptor Tutorial Work Plan:

1. (REQUIRED) Download Raptor:

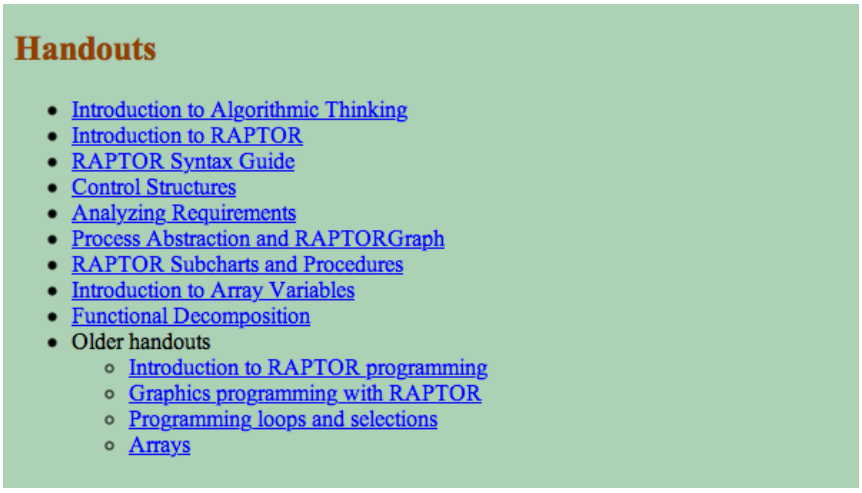
Students were required to download the latest version of Raptor from the following site: http://raptor.martincarlisle.com/.



2. (OPTIONAL) *Step 2:*

Additional optional support documentation was provided to students for enhancing understanding of basic OOP principles and concepts. These documents, which are listed below, are available from http://raptor.martincarlisle.com/ in the Handouts section:

- Lesson 6 – Introduction to Algorithmic Thinking
- Introduction to programming with Raptor
- Programming Control structure



Handouts

- Introduction to Algorithmic Thinking
- Introduction to RAPTOR
- RAPTOR Syntax Guide
- Control Structures
- Analyzing Requirements
- Process Abstraction and RAPTORGraph
- RAPTOR Subcharts and Procedures
- Introduction to Array Variables
- Functional Decomposition
- Older handouts
  - Introduction to RAPTOR programming
  - Graphics programming with RAPTOR
  - Programming loops and selections
  - Arrays

3. (OPTIONAL) Step 3:

Students had the option to work through the following general Help sections of Raptor as specified below.

- Introducing Raptor
- Programming Concepts
- Using Raptor

4. (OPTIONAL) *Step 4:*

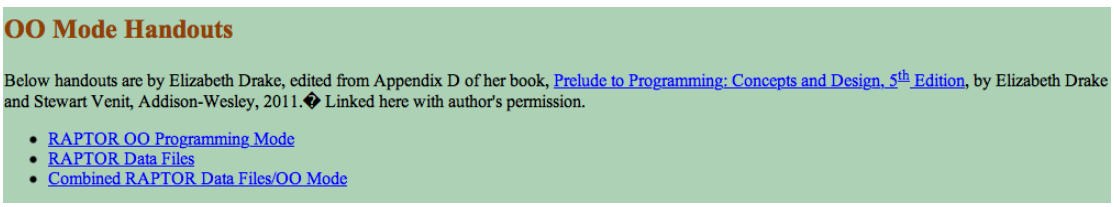
Students had the option to work through the general Help section of Raptor – Building a flowchart. This enabled students to visualise and to be actively involved in playing around with the tool, thus reinforcing the use of the tool through practising concepts.

This tutorial is available in the general Help section of Raptor – Generate Java code.

5. (OPTIONAL) *Step 5:*

To further reinforce the use of the tool through practising concepts, students had the option to work through:

- The Tutorial Introduction to Raptor OO Programming Mode available from http://raptor.martincarlisle.com/ in the OO Mode Handouts section:



OO Mode Handouts

Below handouts are by Elizabeth Drake, edited from Appendix D of her book, Prelude to Programming: Concepts and Design, 5th Edition, by Elizabeth Drake and Stewart Venit, Addison-Wesley, 2011.□ Linked here with author's permission.

- RAPTOR OO Programming Mode
- RAPTOR Data Files
- Combined RAPTOR Data Files/OO Mode

- To generate the Java code of this example, generate Java code as shown in Figure A1. The Java code will be saved in the same folder as the current Raptor application.

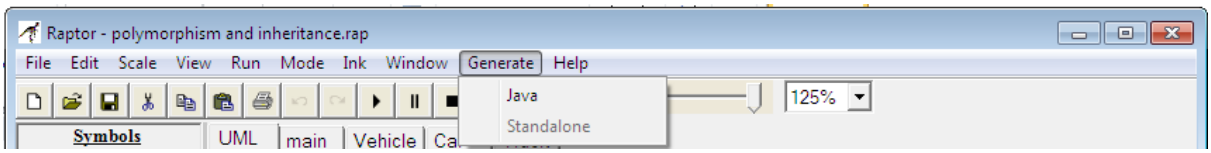


Figure A1: Generate Java Code

6. (OPTIONAL) *Step 6:*

Students had the option to view the following supporting resources of OOP concepts:

- http://www.horton.com/portfolio/OOPs/index.html

7. (REQUIRED) *Step 7:* Applying and synthesising

Students were required to construct applications using what they have learnt by working through the tutorial on Raptor (Object-Oriented Mode Demonstrating Inheritance and Polymorphism – Refer to Appendix I) that was designed in alignment with previous tutorials.

8. (REQUIRED) *Step 8:* Survey Perceptions of the Raptor Tool

Lastly, students were *required* to respond to the web-based questionnaire. The questionnaire aims to determine the impact of an e-learning tool support environment to overcome difficulties in learning OOP in distance education, through student perceptions. The link to the survey is http://goo.gl/lWPSTv.

APPENDIX I

RAPTOR TUTORIAL

Raptor: Object-Oriented Mode
Demonstrating Inheritance and Polymorphism
BY SAADIA ESSA

In this tutorial, we will use the features of the OOP mode in Raptor to create and run programs to demonstrate the object-oriented concepts of Inheritance and Polymorphism.

The aims of this tutorial are to address the following issues:

- OO Mode
- Create Classes
- Create Members
- Code Class Methods
- Demonstrate Inheritance Concepts
- Demonstrate Polymorphism Concepts
- Generate Java Code

Inheritance:

In object-oriented programming, inheritance enables new classes (objects) to take on the properties and methods of existing classes (objects). The existing class is the *base class, and the new class* that inherits from a base is called the *derived class*. A derived class can inherit from its base class as well as extend additional properties and methods of its own.

Polymorphism:

Within the context of object-oriented programming, polymorphism refers to the ability of different objects to respond to the same message in different ways. Thus, a derived class can use base class functionality or override this functionality and implement their own. In order for a method to be polymorphic the action performed

by the method depends on the actual type of the object to which the method is applied.

We consider the following application as an example to demonstrate inheritance and polymorphism concepts. We will be creating a base class named Vehicle and its association with two derived classes named Car and Truck.

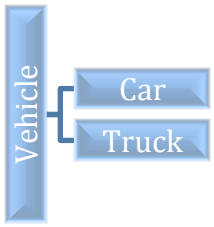


Figure B1: Base class

Step 1: Object-Oriented Mode

Select object-oriented mode, as shown in Figure B2.

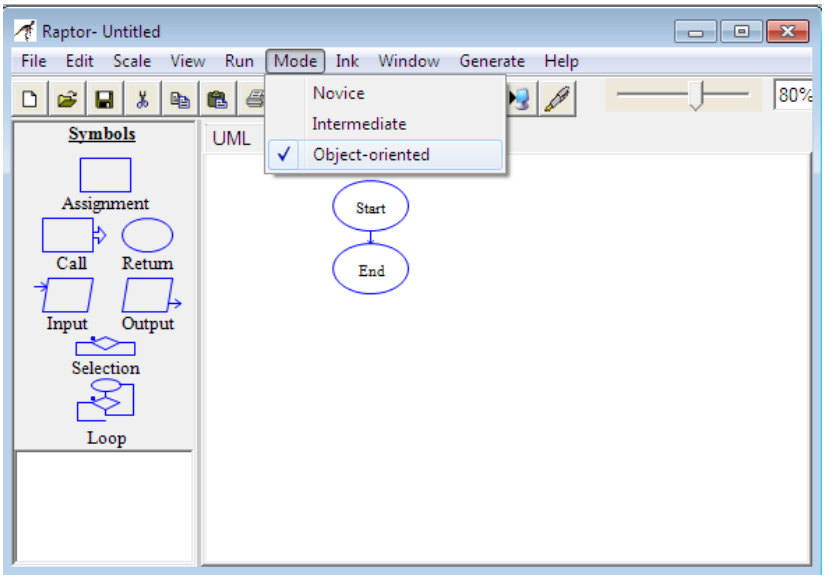


Figure B2: Selecting object-oriented mode

Step 2: Create Classes

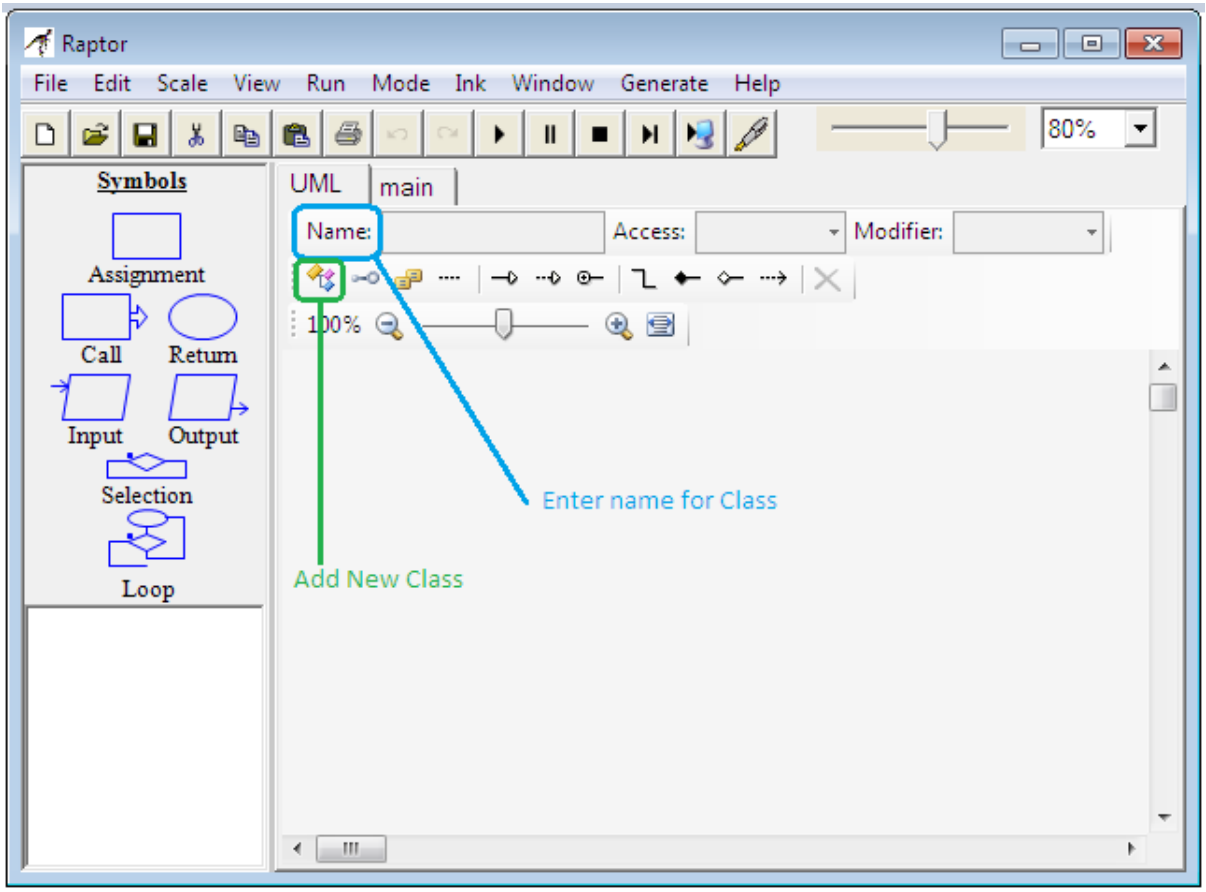


Figure B3: Creates Class

Click the UML tab.

a. Click the Add New Class button as indicated in Figure B3 to add a new Class. Create three Classes.
b. Enter a name for the Class in the Name box as indicated in Figure B3 with the following names respectively.

i. Vehicle
ii. Car
iii. Truck

Vehicle is a base class for derived classes Car and Truck. Use the inheritance From button to set the inheritance between the derived class Car - base class Vehicle and between the derived class Truck - base class Vehicle, as indicated in Figure B4.

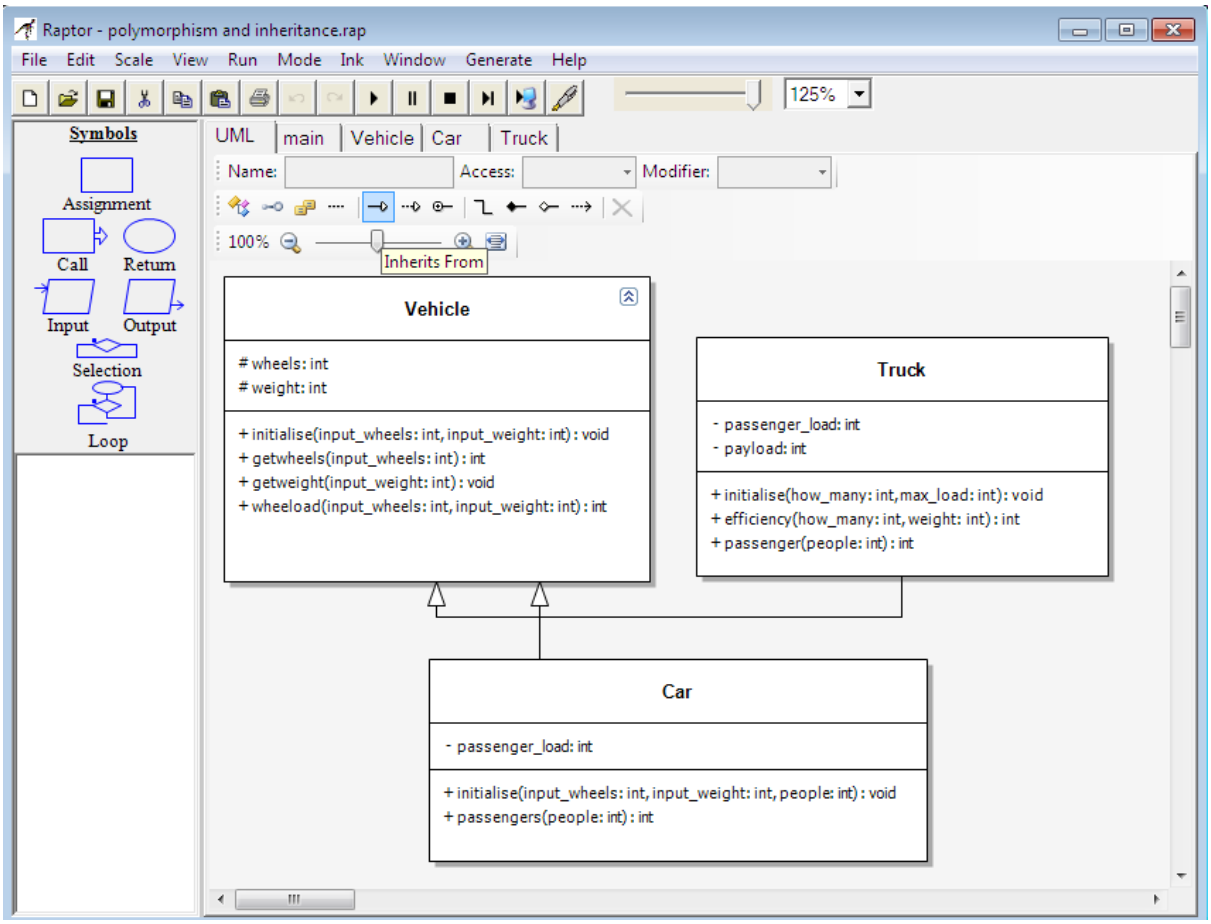


Figure B4: Inheritance relationship association between a base Class Vehicle and derived classes Car and Vehicle

Step 3: Add Members to the Classes

Members include:

a. Attributes: A field includes the following three components:

- An access specifier as discussed in Table B1
- A data type
- Attribute name

b. Methods: Method components specified in this order include:

- An access specifier as discussed in Table B1
- A return type:
  - Is a data type of the value returned by the method or
  - Is void if the method does not return a value.
    - The method name
    - The parameter list is a comma-separated list of input parameters and data types enclosed in parenthesis. Include parameter/s if the method receives a value passed from the main. Empty parenthesis () are used if there are no input parameters.

Table B1: Rules for defining member access specifiers

Public	Attributes and Methods declared as public are directly accessible to any derived class inheriting this class and accessible outside of the base class or derived classes.
Private	Attributes and Methods declared as private are not accessible to any derived class inheriting this class and not accessible outside of the base class or derived classes.
Protected	Attributes and Methods declared as protected are directly accessible to any derived class inheriting this class but not accessible outside of the base class or derived classes.

Double-clicking inside the respective classes opens a window, allowing you to enter the members as seen in Figure B5.

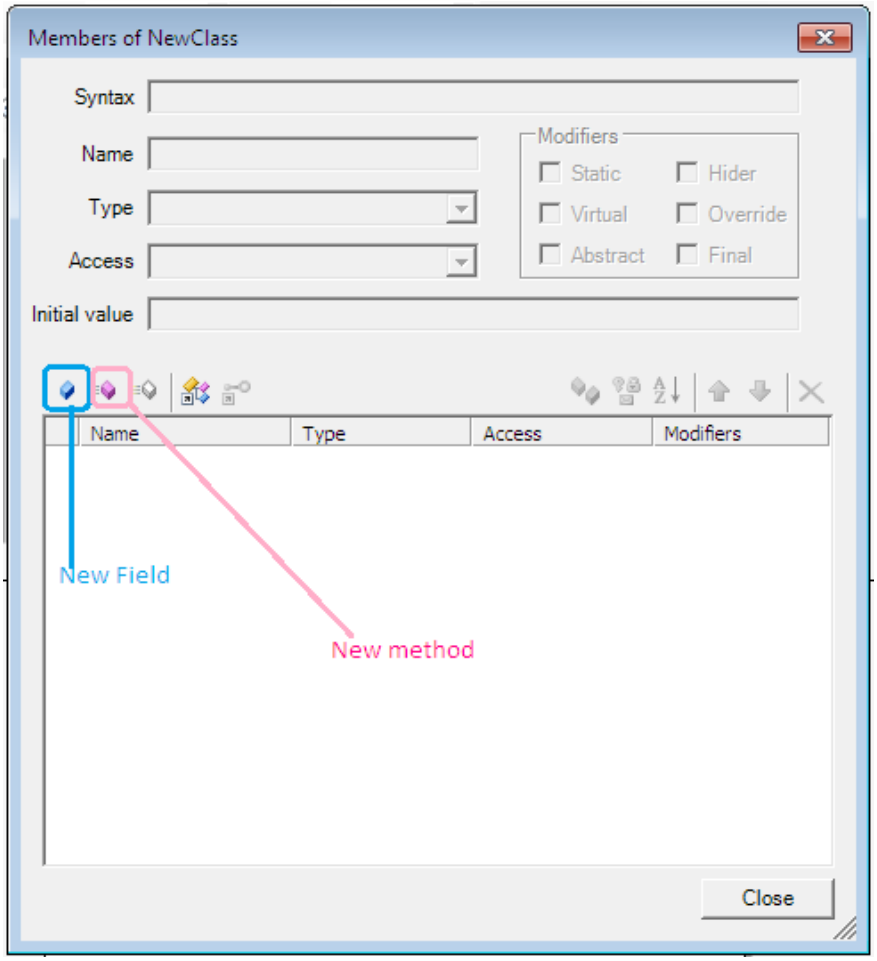


Figure B5: Add members

3.1 Add Members to Vehicle Class

a. Attributes: protected: wheels (int); weight (int). (As shown in Figure B6)

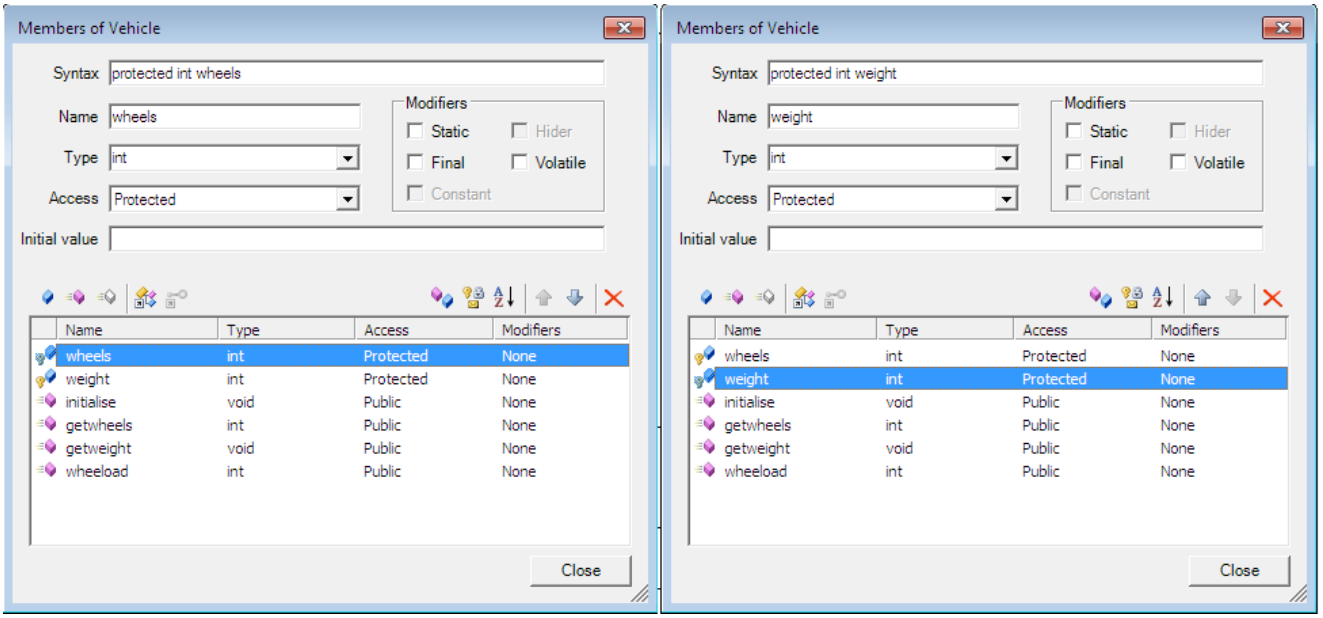


Figure B6: Vehicle Class Add Fields Syntax

b. Methods syntax are as follows:

- public void initialise (int input_wheels, int input_weight): sets the value of wheels and weight of vehicle as passed a value from the main program to the variables input_wheels and input_weight. (As shown in Figure B7)
- public int get_wheels (int input_wheels) retrieves the value of the wheels of the vehicle as passed a value from the main program to the variables input_wheels and returns this result when requested. (As shown in Figure B7)
- public int get_weight(int input_weight) retrieves the value of the weight of the vehicle as passed a value from the main program to the variables input_weight and returns this result when requested.
- public int wheel_load(int input_wheels, int input_weight): First, the method retrieves the value of the wheels and weight of the vehicle as passed a value from the main program to the variables input_wheels and input_weight. Secondly, the method computes the load of vehicle by dividing weight/wheels. Lastly, the wheel_load method returns this result when requested. (As shown in Figure B8)

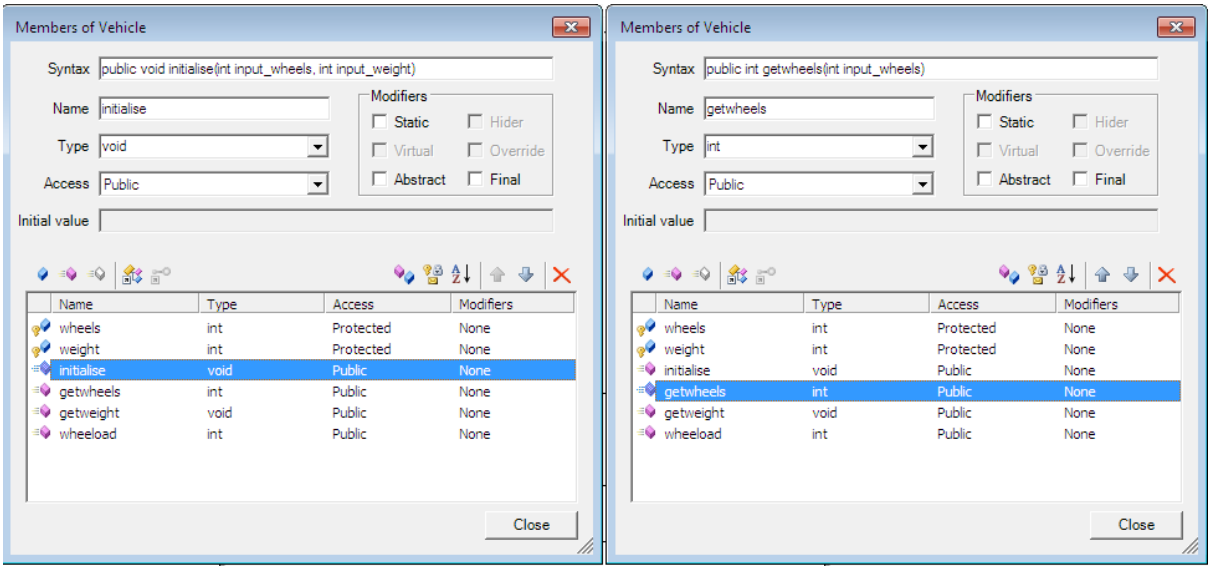


Figure B7: Vehicle Class Method Syntax

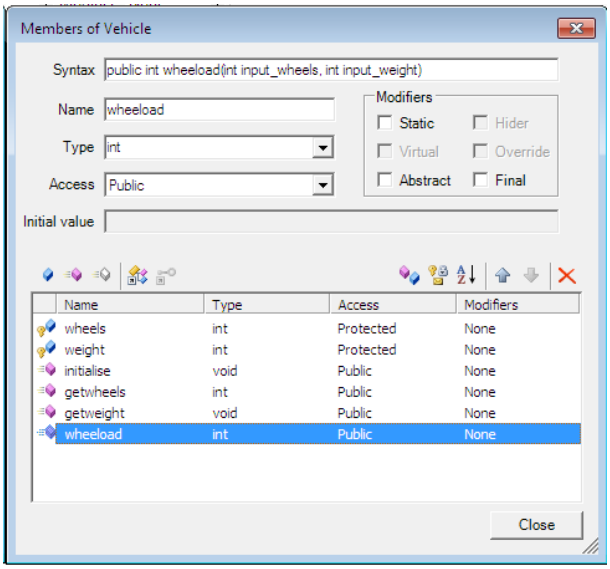


Figure B8: Vehicle Class Method Syntax

3.2 Add Members to Car Class

a. Attributes: private: passenger_load (int). (As shown in Figure B9)

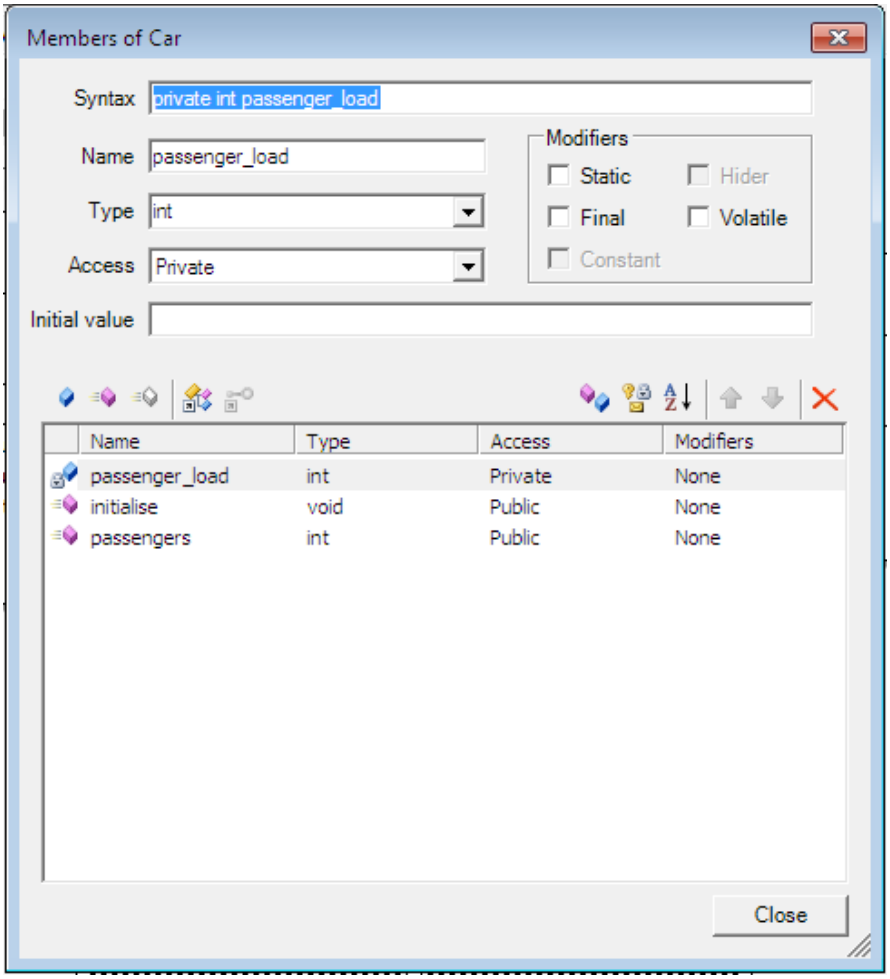


Figure B9: Car Class Attributes Syntax

b. Methods syntax are as follows:

- public void initialise(int input_wheels, int input_weight, int people): sets the value of wheels, weight and number of passengers of car as passed a value from the main program to the variables input_wheels, input_weight and people. (As shown in Figure B10)

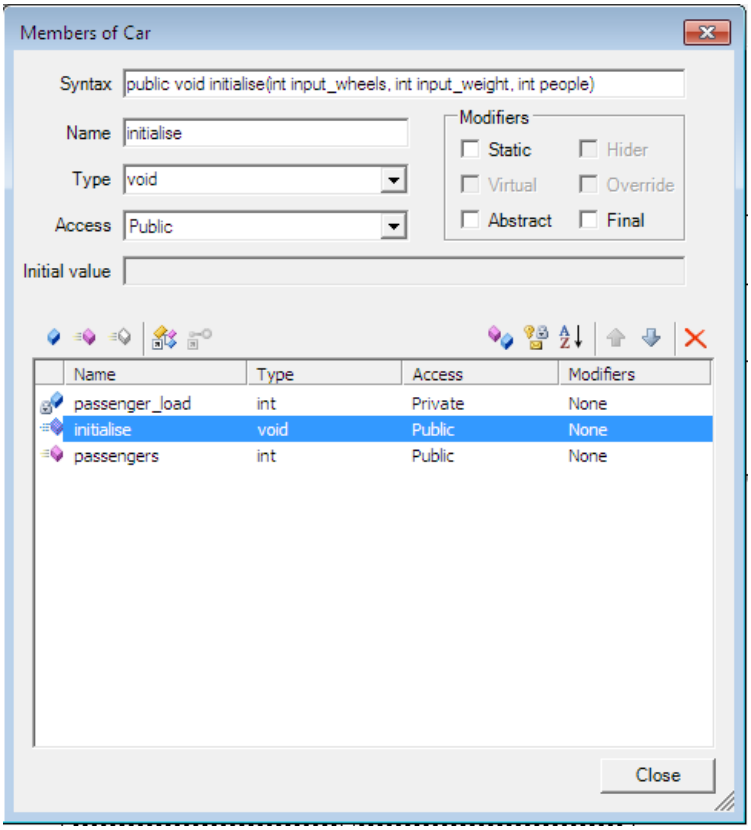


Figure B10: Car Class Method Syntax

- public int passengers(int people) retrieves the value of the passenger_load of the vehicle as passed a value from the main program to the variables people and returns this result of the number of passengers when requested. (As shown in Figure B11)

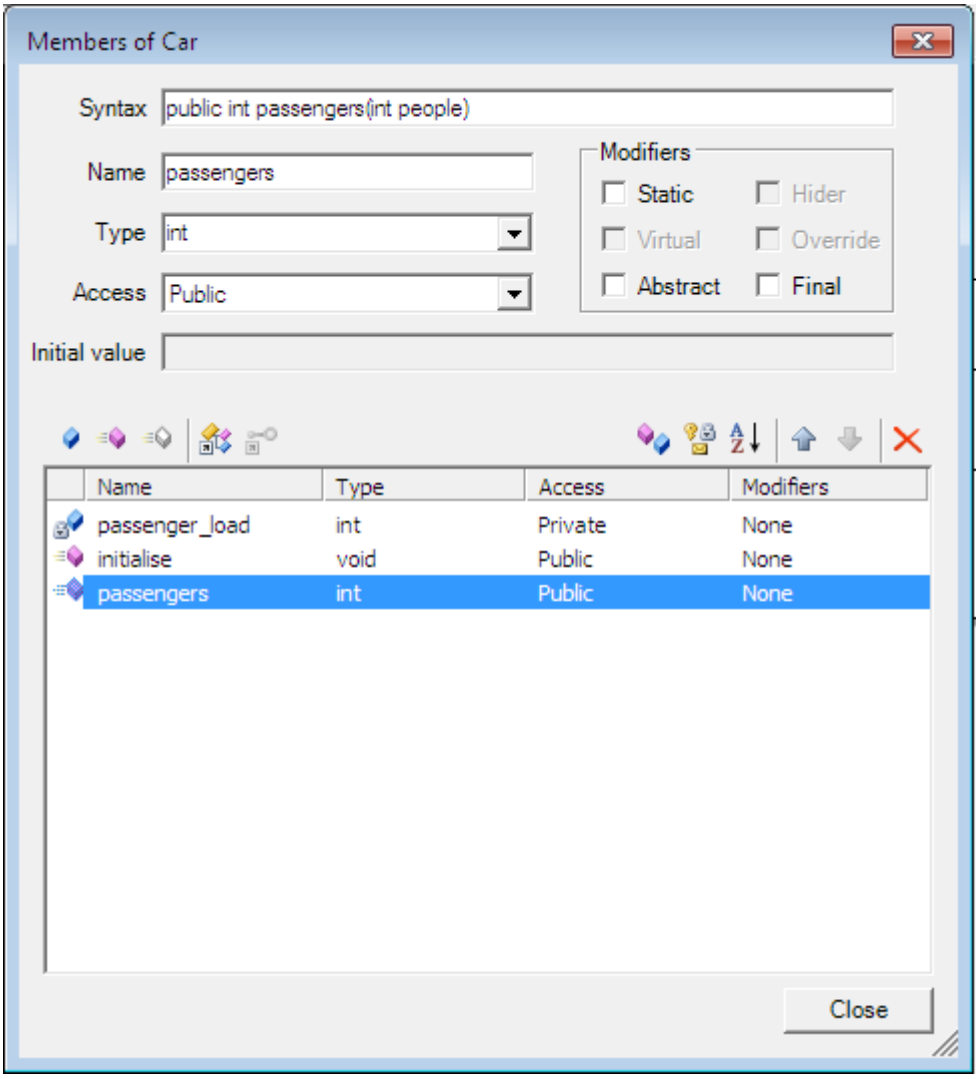


Figure B11: Car Class Method Syntax

3.3 Add Members to Truck Class

a. Attributes: private: passenger_load (int); payload (int). (As shown in Figures B12 and B13)

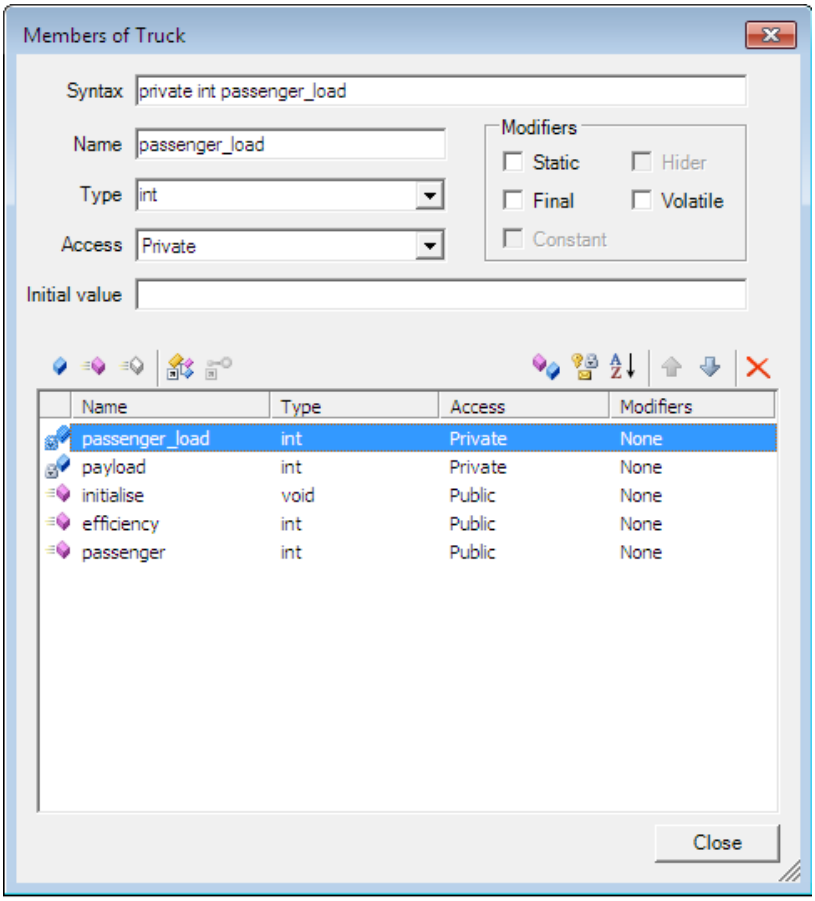


Figure B12: Truck Class Attributes Syntax

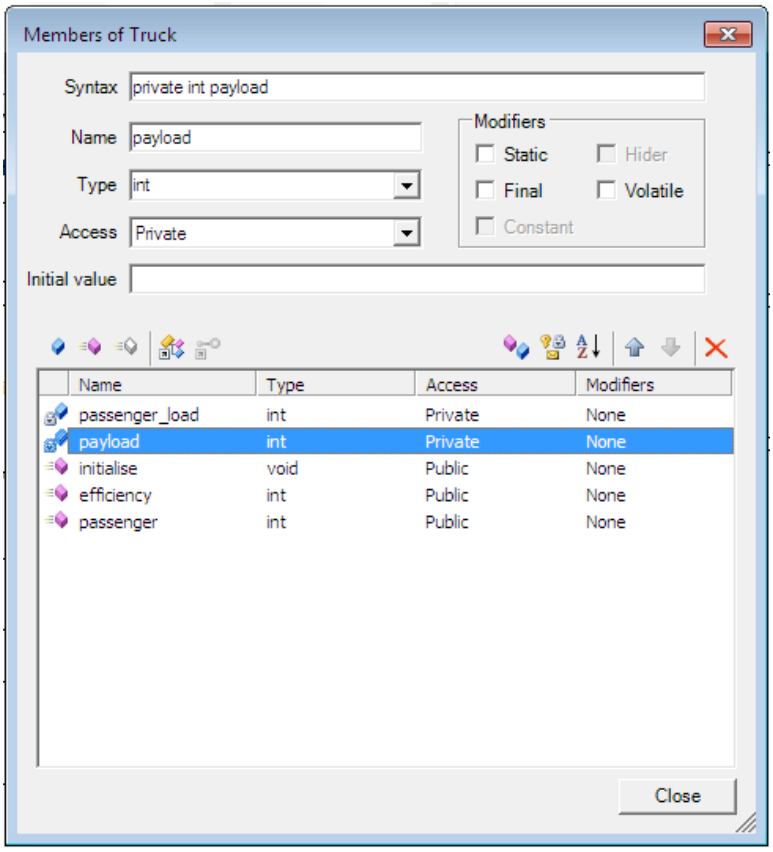


Figure B13: Truck Class Attribute Syntax

b. Methods syntax are as follows:

- public void initialise(int how_many, int max_load): sets the number of passengers and maximum load of truck as passed a value from the main program to the variables how_many and max_load. (As shown in Figure B14)

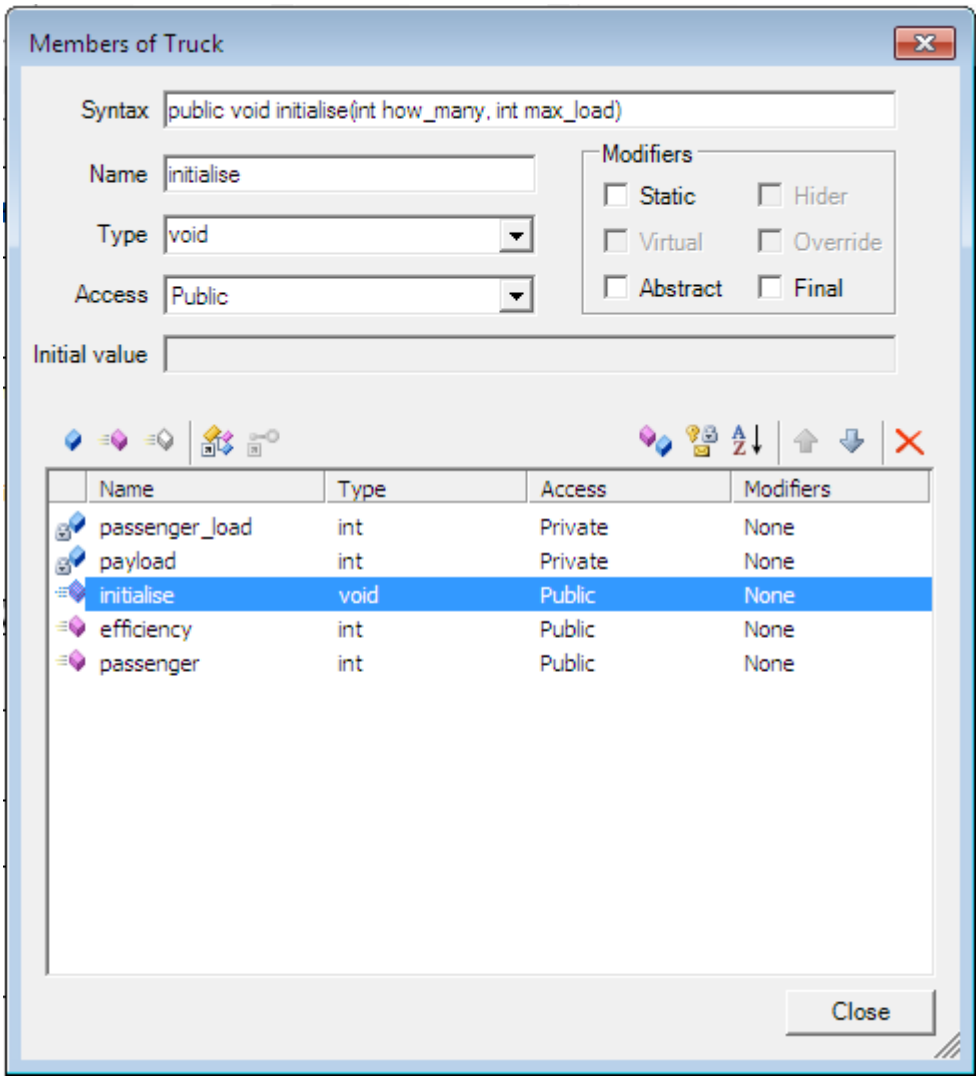


Figure B14: Truck Class Method Syntax

- public int efficiency(int how_many, int weight): First, the method retrieves the value of the number of passengers and weight of truck as passed a value from the main program to the variables how_many and weight. Secondly, the method computes the efficiency of Truck by dividing how_many / how_many + weight. Lastly, the efficiency method returns this result when requested. (As shown in Figure B15)

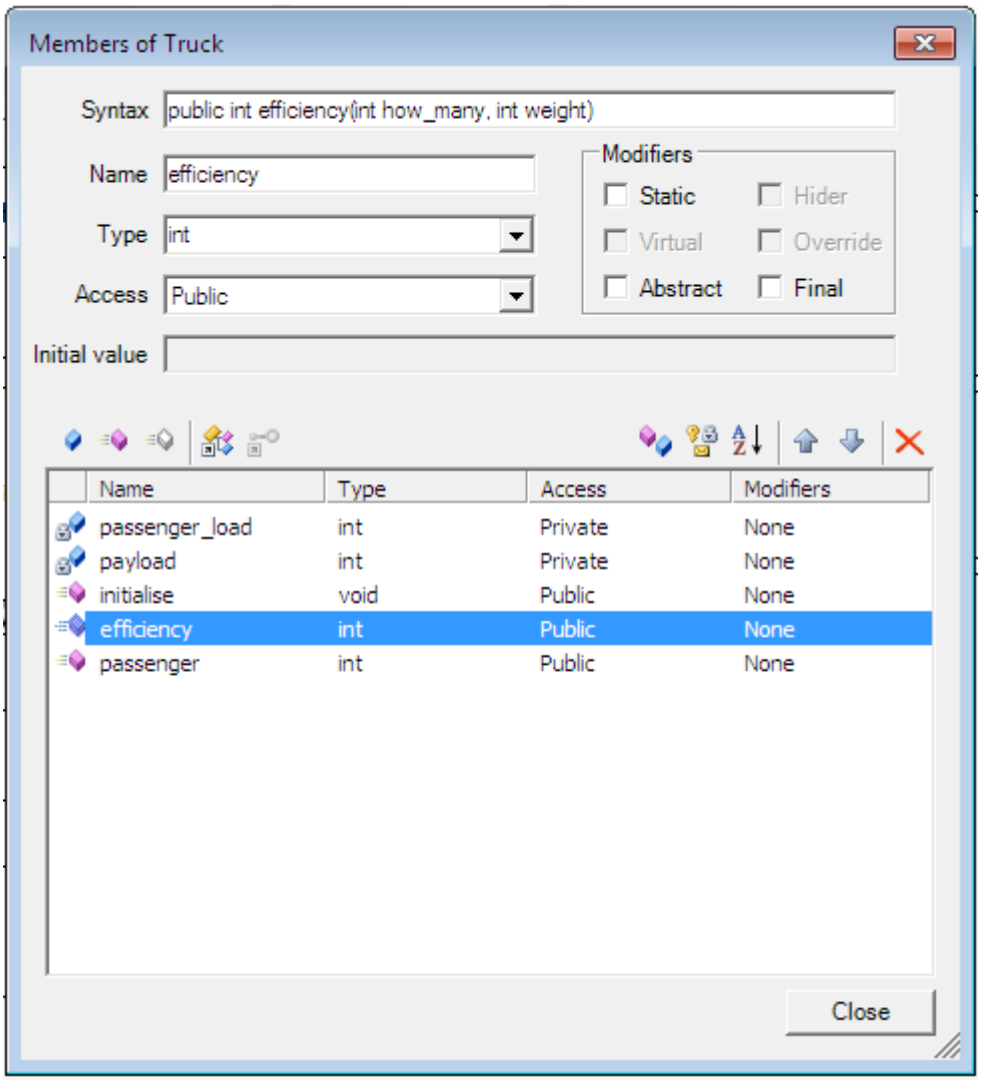


Figure B15: Truck Class Method Syntax

- public int passenger(int people): sets the number of passengers of truck as passed a value to it from the main program to the variable people. (As shown in Figure B16)

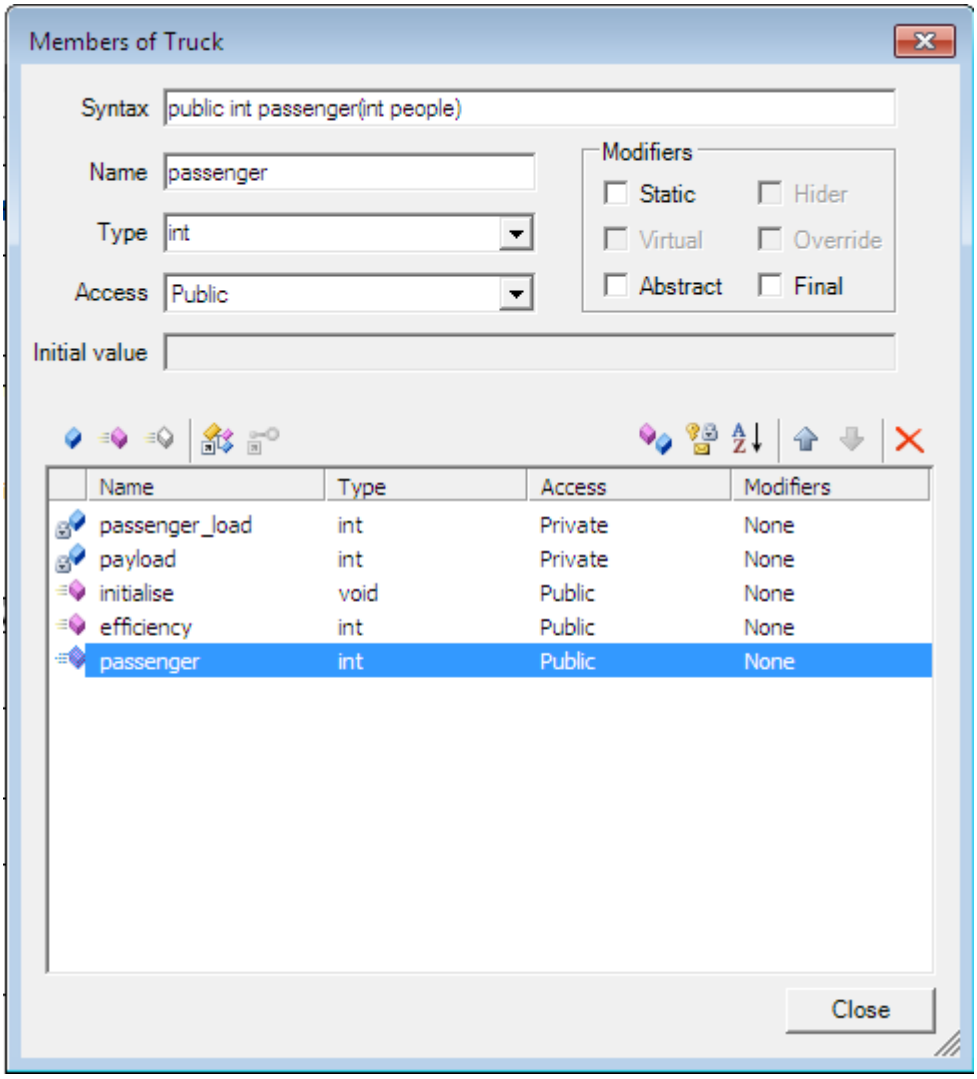


Figure B16: Truck Class Method Syntax

Step 4: Code Class Methods

The new operator instantiates a class and invokes the instance method, thus allocating the objects before using them.

The ‘this’ keyword is used in an assignment, which references the current object within an instance method.

Each class has its own tab as seen in Figure B17.

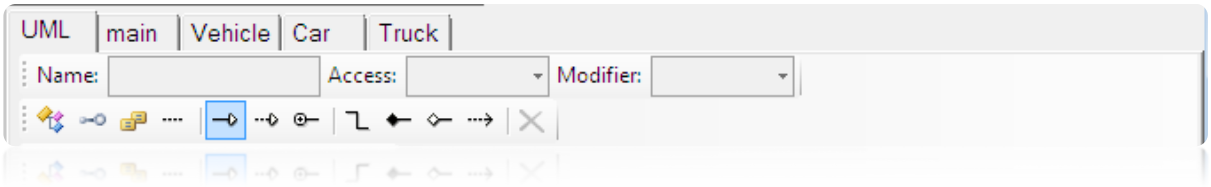


Figure B17: Class Tab

4.1 Code Vehicle Class Methods

Click the Vehicle tab to code each of method of Vehicle Class – four new tabs, one for each Method:

a. public void initialise(int input_wheels, int input_weight). Code as seen in Figure B18.

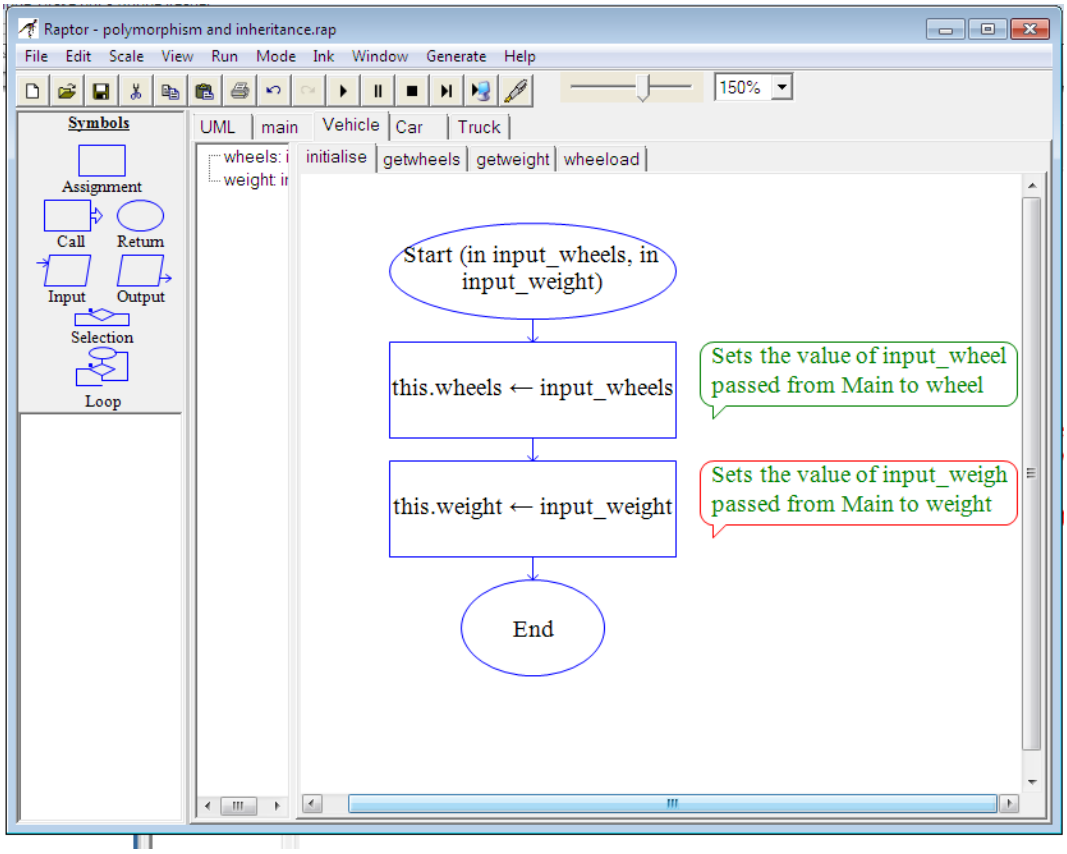


Figure B18: Vehicle Class initialise tab to code initialise method

b. public int get_wheels(int input_wheels). Code as seen in Figure B19.

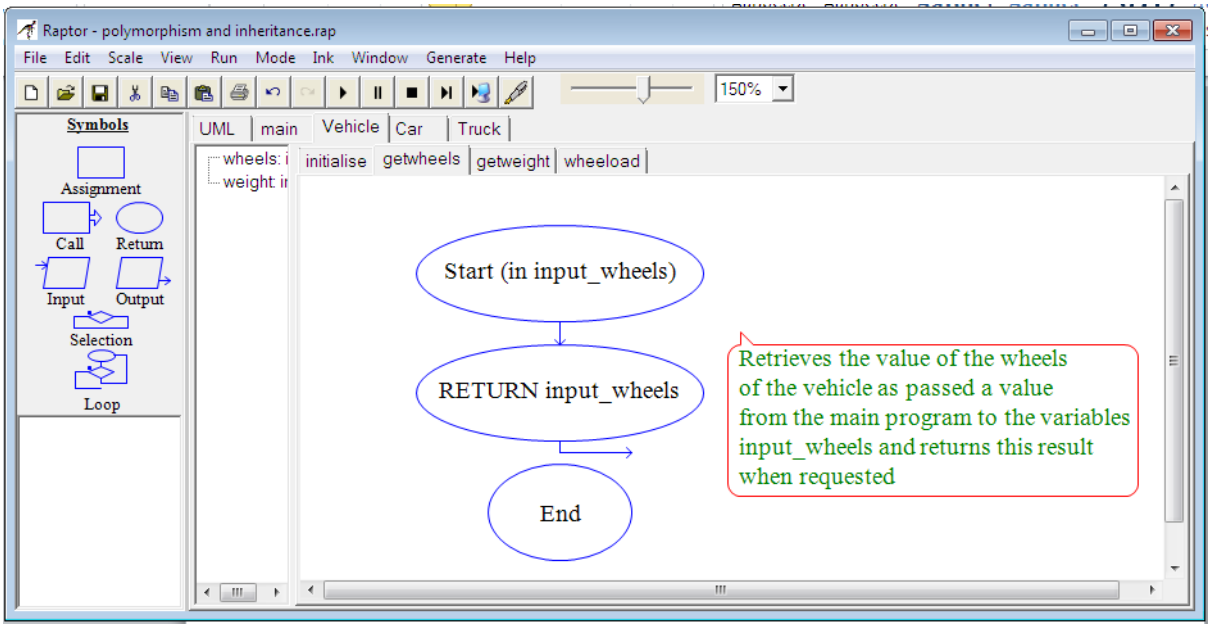


Figure B19: Vehicle Class getwheels tab to code getwheels method

c. public int get_weight(int input_weight). Code as seen in Figure B20.

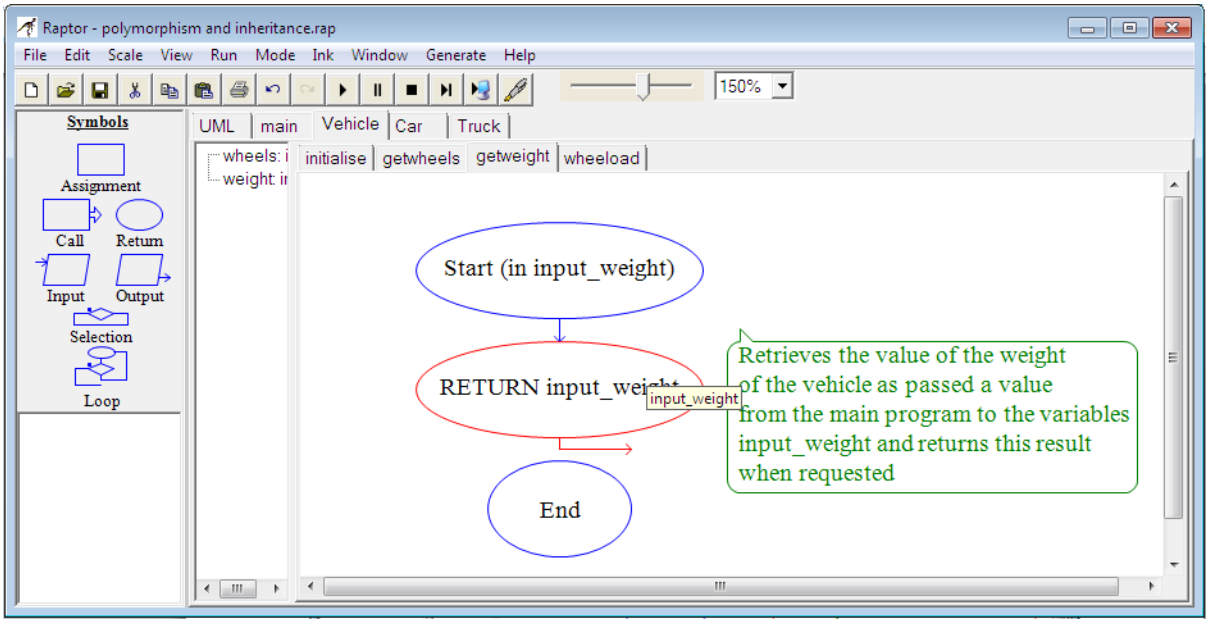


Figure B20: Vehicle Class getweight tab to code getweight method

d. public int wheel_load(int input_wheels, int input_weight). Code as seen in Figure B21.

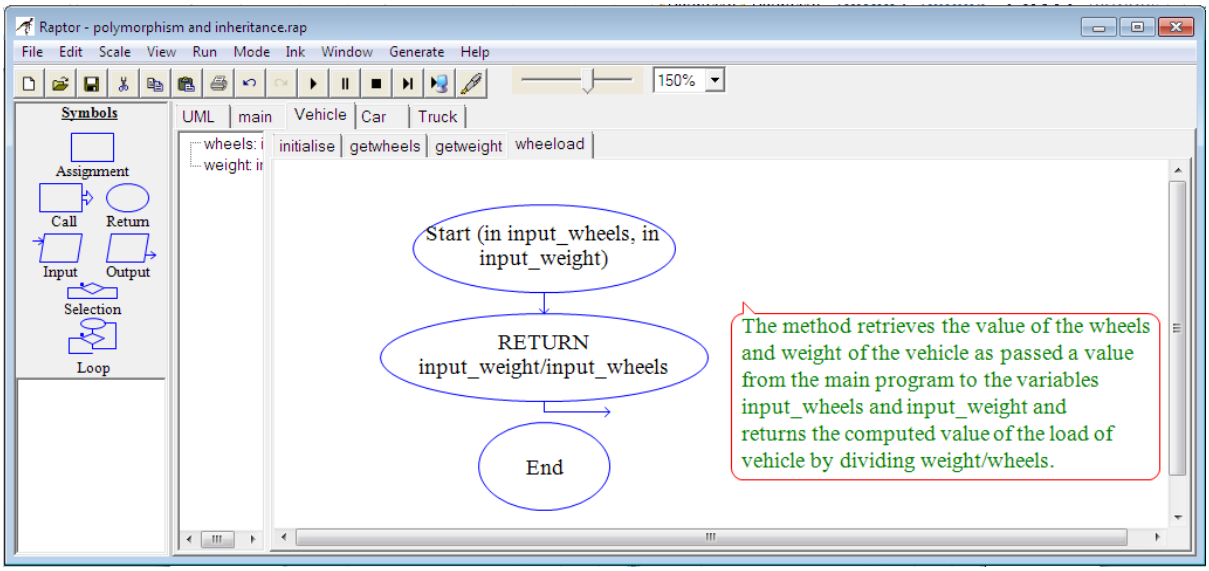


Figure B21: Vehicle Class wheeload tab to code wheeload method

4.2 Code Car Class Methods

Click the Car tab to code each of method of Vehicle Class – two new tabs, one for each Method:

a. public void initialise(int input_wheels, int input_weight, int people). Code as seen in Figure B22.

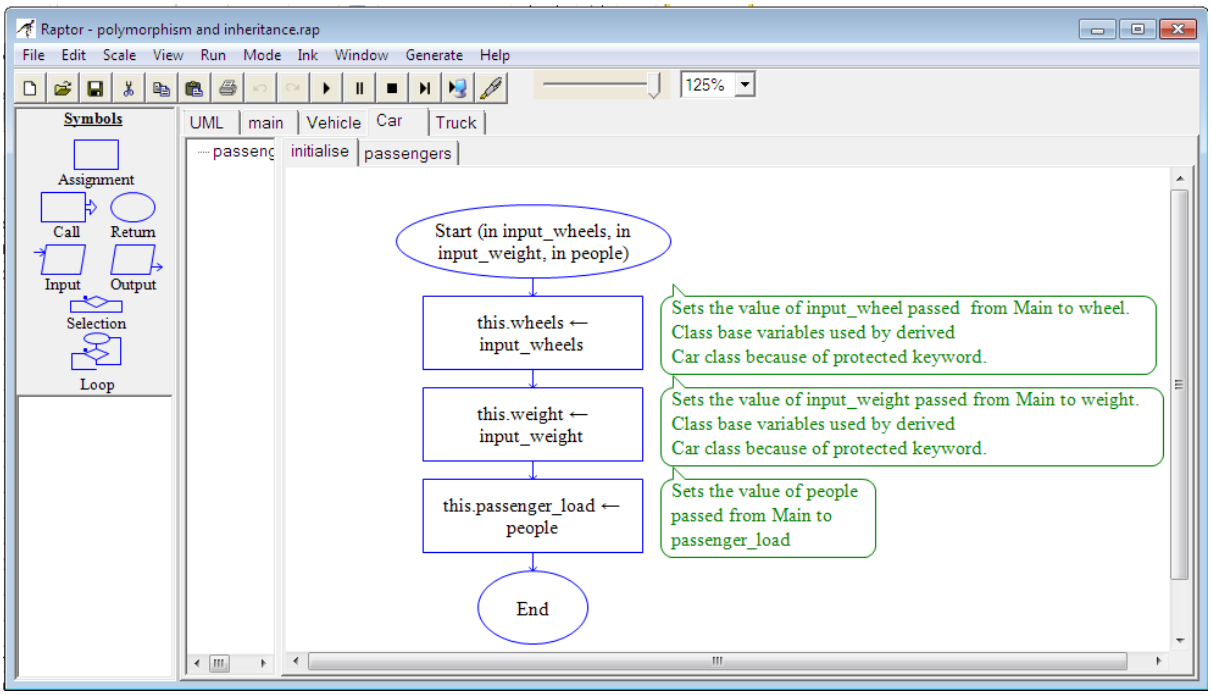


Figure B22: Car Class initialise tab to code initialise method

b. public int passengers(int people). Code as seen in Figure B23.

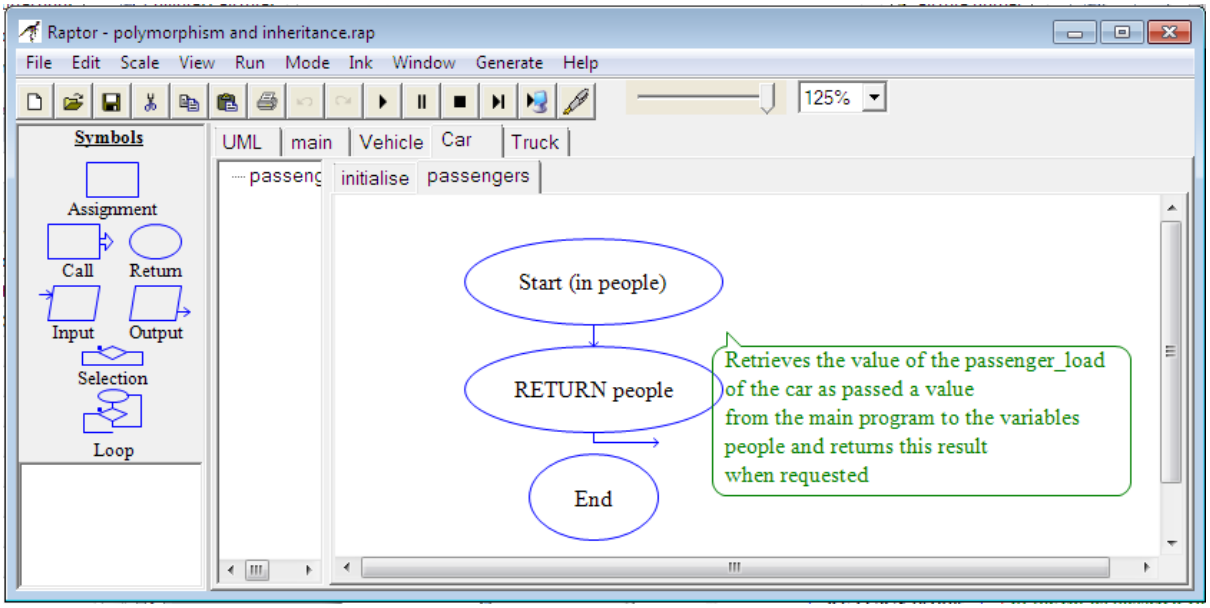


Figure B23: Car Class passengers tab to code passengers method

4.3 Code Truck Class Methods

Click the Truck tab to code each methods of Truck Class – three new tabs, one for each Method:

a. public void initialise(int how_many, int max_load). Code as seen in Figure B24.

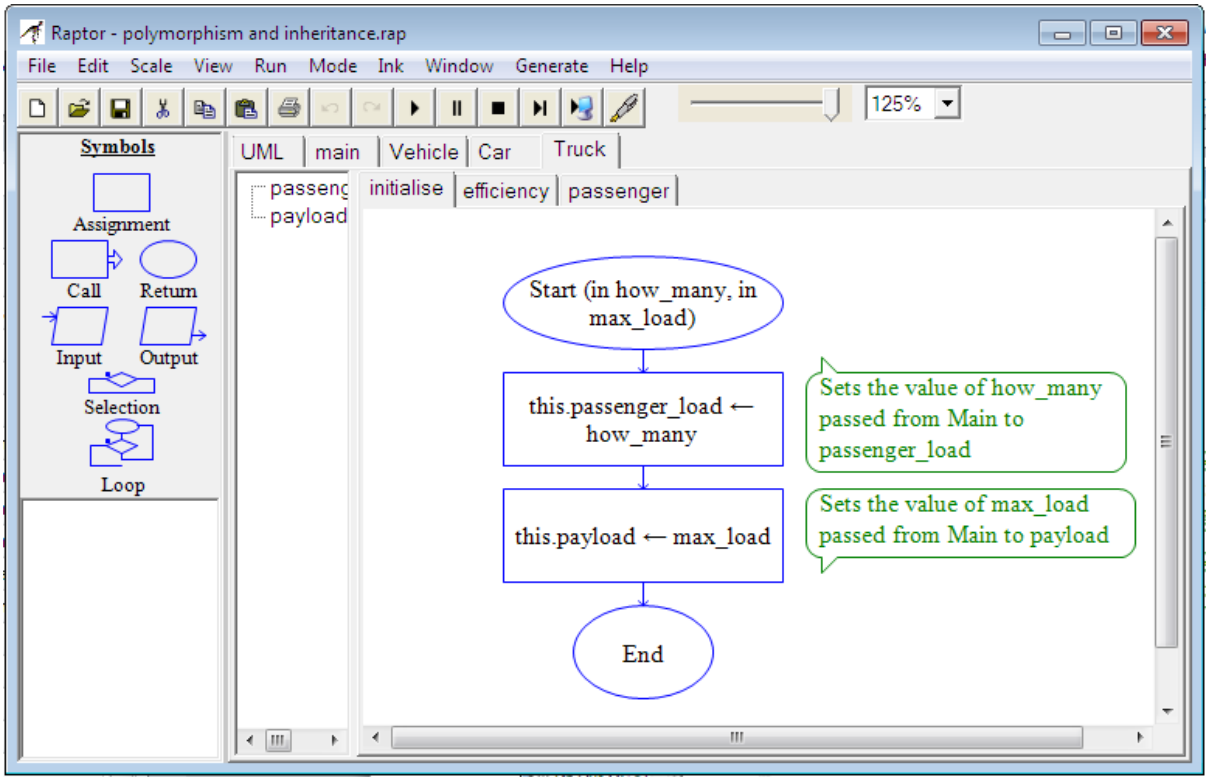


Figure B24: Truck Class initialise tab to code initialise method

b. public int efficiency(int how_many, int weight). Code as seen in Figure B25.

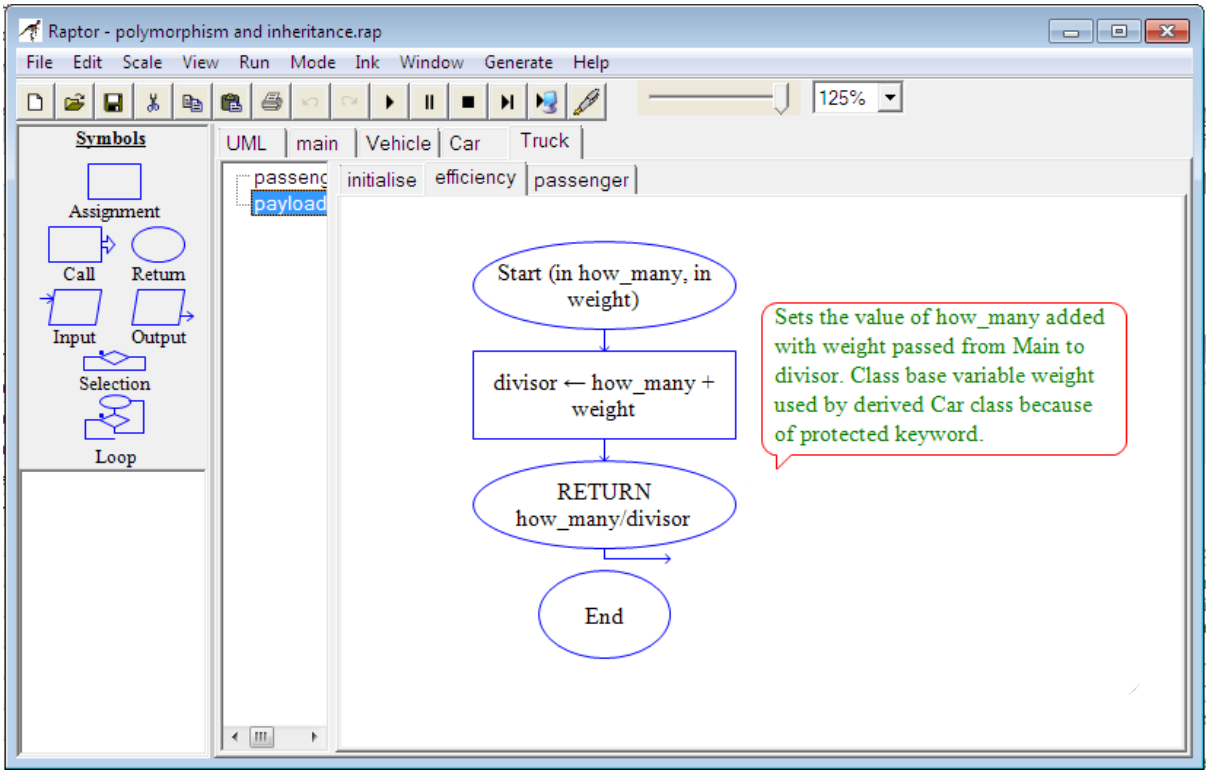


Figure B25: Truck Class efficiency tab to code efficiency method

c. public int passenger(int people). Code as seen in Figure B26.

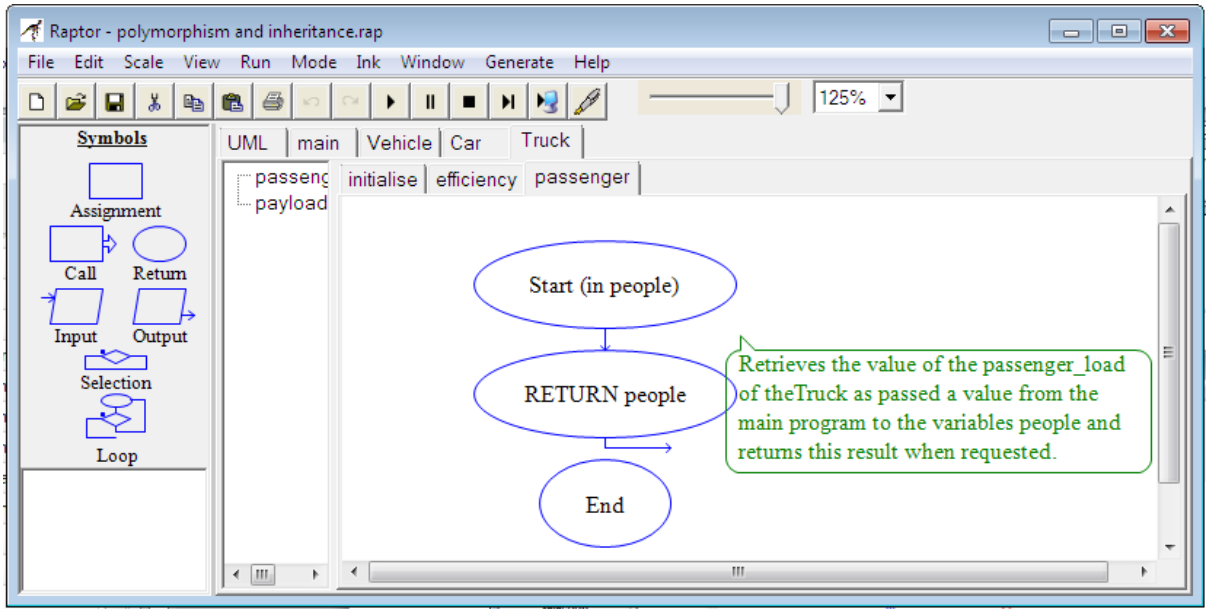


Figure B26: Truck Class passenger tab to code passenger method

Step 5: Inheritance and Polymorphism

Derived Classes Car and Truck inherit the field wheel, field weight, and method wheeload() from base class Vehicle. Both Car and Truck classes have additional behaviour that is unique to them. Car Class additional behaviour determines passenger load of the car. On the other hand, Truck class additional behaviour determines passenger load of the Truck and determines efficiency of truck by dividing the number of passengers of Truck from weight and number of passengers.

Furthermore, each derived class has its own implementation of Method for initialise(). Base class Vehicle initialise() method sets the value of wheels and weight of vehicle. Derived class Car initialise() method sets the value of wheels, weight and number of passengers of car. Derived class Truck initialise() method sets the number of passengers and maximum load of truck. Thus, polymorphic behaviour displayed as different objects can respond to the same initialise() method in different ways.

Step 6: The Main Program

Now the Main program can be created.

The first part of the main program demonstrates Base Class Vehicle. The user enters the number of wheels of the Vehicle and the weight of the Vehicle in kilograms (kg); computes Vehicles wheel loading in kilograms per tire; and displays the results of the number of wheels, the weight in kg and wheel loading in kg/tire. This is accomplished by instantiating an object of type Vehicle (using the keyword new) called Unicycle, using the methods and attributes of Vehicle. Furthermore, note the method initialise() is called once for each object. In this case, Base class Vehicle initialise() method sets the value of wheels and weight of vehicle. Unicycle has a method initialise() associated with it. Figure B27 shows the first part of main program implemented using Raptor OOP way.

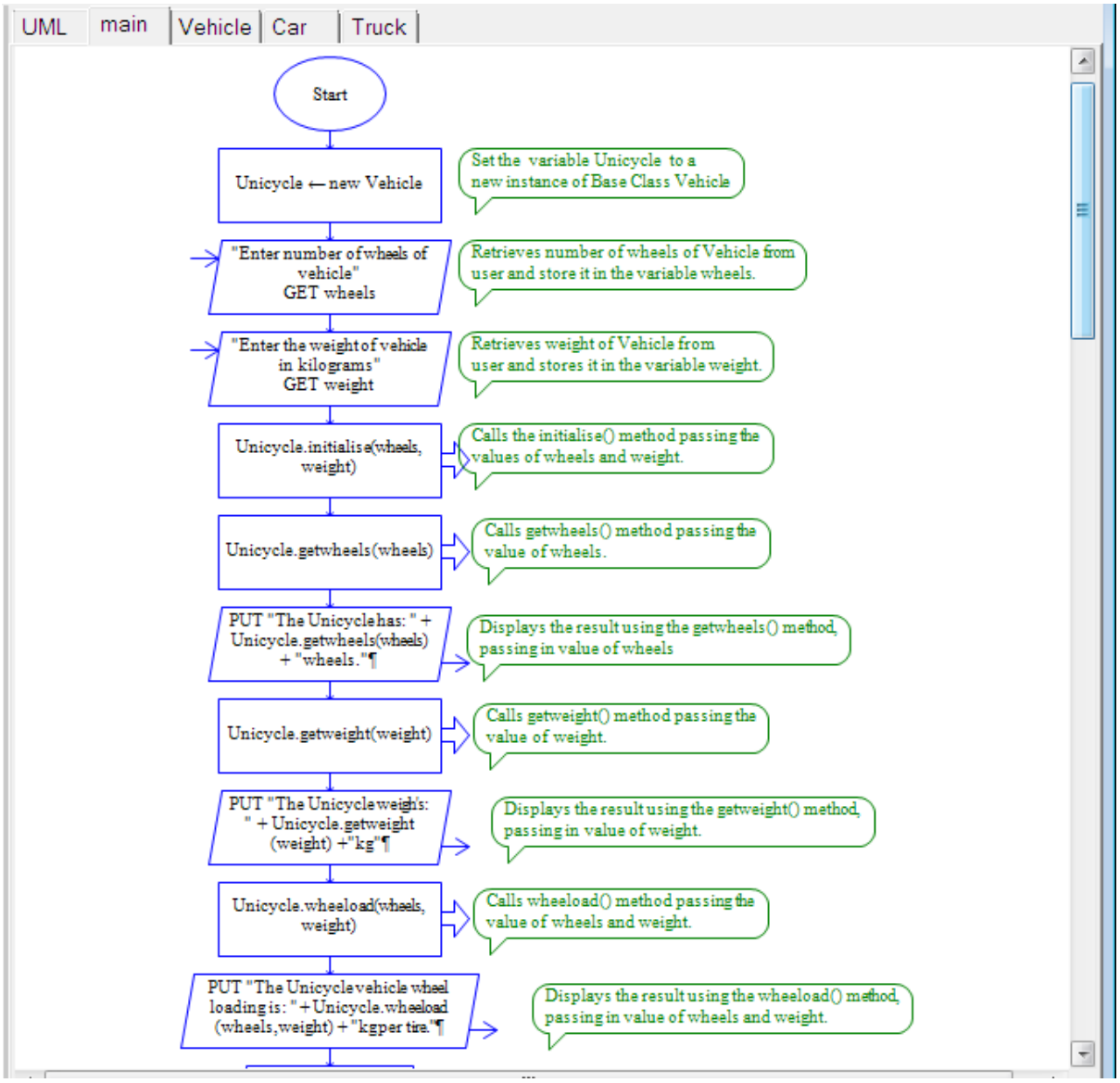


Figure B27: Main program demonstrates Base Class Vehicle

The second part of the Main program demonstrates Derived Class Car. The user enters the number of wheels of the Car, the weight of the Car in kilograms and number of passengers of Car; computes Car wheel loading in kilograms per tire; and displays the results of the number of wheels, number of passengers the Car carries, the weight in kg and wheel loading in kg/tire. This is accomplished by instantiating an object of type Car (using the keyword new) called Sedan, using the methods and attributes of Car. Furthermore, note that the method initialise() is called once for each object. In this case, Derived class Car initialise() method sets the value of wheels, weight and number of passengers of the car. Sudan has a method initialise() associated with it. Figure B28 shows the second part of the main program implemented using Raptor OOP way.

Figure B28: Main program demonstrates Derived Class Car

The third part of the Main program demonstrates Derived Class Truck. The user enters the number of wheels of the Truck, the weight of the Truck in kilograms, maximum load of Truck and the number of passengers of Truck; computes Car wheel loading in kilograms per tire and efficiency of Truck; and displays the results of the number of wheels, the weight in kg, number of passengers the Truck carries, wheel loading in kg/tire and efficiency. This is accomplished by instantiating an object of type Truck (using the keyword new) called Trailer, using the methods and attributes of Truck. Furthermore, it is to be noted that the method initialise() is called once for each object. In this case, Derived class Truck initialise() method sets the number of passengers and maximum load of truck. Trailer has a method initialise() associated with it. Figure B29 shows the third part of the main program implemented using Raptor OOP way.

Figure B29: Main program demonstrates Derived Class Truck

Step 7: Generate Java Code

To generate the Java code of this example, generate Java as shown in Figure B30. The Java code will be saved in same folder of current Raptor application.

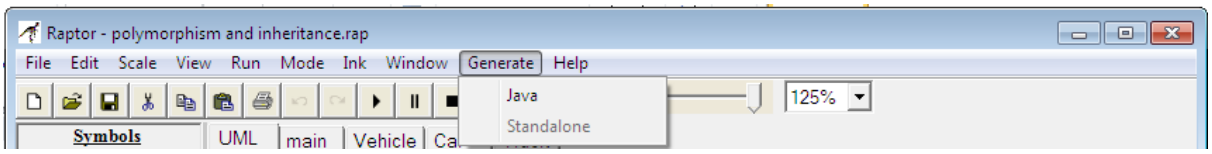


Figure B30: Generate Java Code

Some of the generated Java Code for the above example is displayed below:

```
/**
 * NAME:
 * DATE:
 * FILE:
 * COMMENTS:
 */

public class polymorphism and inheritance_main
{
  public static void main(String[] args)
  {
    // declare variables
    String raptor_prompt_variable_zzyz = null;
    ?? Unicycle = ??;
    ?? wheels = ??;
    ?? Sudan = ??;
    ?? passenger_load = ??;
    ?? payload = ??;
    ?? Trailer = ??;
    ?? weight = ??;

    Unicycle = new Vehicle();
raptor_prompt_variable_zzyz = "Enter number of wheels of vehicle";
    wheels = get???(raptor_prompt_variable_zzyz);
raptor_prompt_variable_zzyz = "Enter the weight of vehicle in kilograms";
    weight = get???(raptor_prompt_variable_zzyz);
    Unicycle.initialise(wheels,weight);
    Unicycle.getwheels(wheels);
    printLine("The Unicycle has: " + Unicycle.getwheels(wheels) + "wheels.");
    Unicycle.getweight(weight);
    printLine("The Unicycle weigh's: " + Unicycle.getweight(weight) + "kg");
    Unicycle.wheeload(wheels,weight);
    printLine("The Unicycle vehicle wheel loading is: " +
Unicycle.wheeload(wheels,weight) + "kg per tire.");
```

```
    Sudan = new Car();
raptor_prompt_variable_zzyz = "Enter number of wheels of Car";
    wheels = get???(raptor_prompt_variable_zzyz);
raptor_prompt_variable_zzyz = "Enter the weight of Car in kilograms";
    weight = get???(raptor_prompt_variable_zzyz);
raptor_prompt_variable_zzyz = "Enter number of passengers of Car";
    passenger_load = get???(raptor_prompt_variable_zzyz);
    Sudan.initialise(wheels,weight,passenger_load);
    Sudan.getwheels(wheels);
    printLine("The Sudan has: " + Sudan.getwheels(wheels) + " wheels.");
    Sudan.passengers(passenger_load);
    printLine(" The Sudan car carries: " + Sudan.passengers(passenger_load)
+ " passengers.");
    Sudan.getweight(weight);
    printLine("The Sudan Car weigh's: " + Sudan.getweight(weight) + "kg.");
    Sudan.wheeload(wheels,weight);
    printLine("The Unicycle vehicle wheel loading is: " +
Sudan.wheeload(wheels,weight) + "kg per tire.");
    Trailer = new Truck();
raptor_prompt_variable_zzyz = "Enter number of wheels of Truck";
    wheels = get???(raptor_prompt_variable_zzyz);
raptor_prompt_variable_zzyz = "Enter the weight of Truck in kilograms";
    weight = get???(raptor_prompt_variable_zzyz);
    Trailer.getwheels(wheels);
    printLine("The Trailer has: " + Trailer.getwheels(wheels) + "wheels.");
    Trailer.getweight(weight);
    printLine("The Trailer weigh's: " + Trailer.getweight(weight) + "kg");
raptor_prompt_variable_zzyz = "Enter the maximum load of Truck in
kilograms";
    payload = get???(raptor_prompt_variable_zzyz);
raptor_prompt_variable_zzyz = "Enter number of passengers of Truck";
    passenger_load = get???(raptor_prompt_variable_zzyz);
    Trailer.initialise(passenger_load,payload);
```

APPENDIX J

CERTIFICATE OF LANGUAGE EDITING

DECLARATION BY LANGUAGE EDITOR

11 September 2015

TO WHOM IT MAY CONCERN

DECLARATION: LANGUAGE EDITING of MSc Dissertation

I hereby declare that I have edited the Master of Science in Computing dissertation of SAADIA FAHIM ESSA entitled *"**USING AN E-LEARNING TOOL TO OVERCOME DIFFICULTIES IN LEARNING OBJECT-ORIENTATED PROGRAMMING**"* and found the written work to be free of ambiguity and obvious errors. It is the responsibility of the student to address any comments from the editor or supervisor. Additionally, it is the final responsibility of the student to make sure of the correctness of the dissertation.

Khomotso Bopape

Full Member of the Professional Editors' Group

Professional EDITORS Group | An association of editors in educational, academic and general publishing

Address: **P.O. Box 40208, Arcadia, Pretoria, 0007**
Tel No.: **012 753 3670,** Fax No.: **086 267 2164** and Email Address: **khomotso@letsedit.co.za**