Taking first year college students' programming skills from the visual to the procedural

by

Tijani Fatimah Yetunde

Submitted in fulfilment of the requirements for the degree

PHILOSOPHIAE DOCTOR

in

Department of Science, Mathematics and Technology Education
Faculty of Education
UNIVERSITY OF PRETORIA

Supervisor: Dr Ronel Callaghan
Co-supervisor: Prof. Rian de Villiers

October 2018

Declaration

I, **Tijani Fatimah Yetunde,** hereby declare that this thesis, which I hereby submit for the degree of Philosophiae Doctor in the department of Science Mathematics and Technology Education, at the University of Pretoria, is my own original work and has not previously been submitted by me to any other institution of higher learning. All sources cited or quoted in this thesis are indicated and acknowledged with a comprehensive list of references.

.....

Tijani Fatimah Yetunde

Acknowledgements

Let God Almighty be praised for seeing me through each phase of this four year journey, despite the challenges I faced.

Many people worked by my side during this four year journey and, without them, this thesis would not have been a success. They have shown me doors of opportunity and how to unlock them. I want to especially thank my supervisor, Dr Ronel Callaghan, for her support, valuable time and constructive comments which helped to shape this thesis. I extend my unreserved gratitude to my co-supervisor, Prof. Rian de Villers, who critically read my work and corrected all the dots and commas and to Prof. Tony Clear of the Auckland University of Technology, New Zealand, for his constructive criticism on my thesis. I sincerely wish to thank the college management of Michael Otedola College of Primary Education (MOCPED) for affording me the opportunity to study, as well as recommending me for the TETFund sponsorship which I enjoyed during the programme. I express my profound gratitude to my students who participated in this study (between 2015/2016 and 2016/2017 academic sessions). In addition, I extend my gratitude to the non-participant observers and video recorder for their assistance and support toward the success of this research study.

I wish to unreservedly thank my parents, Mr and Mrs Taofeek Tijani, for their support, prayers, parental advice and for believing in me. You are one in a million. I express my gratitude to my spiritual parents, Pastor and Dr (Mrs) Emmanuel for their support, prayers, counsel and hospitality. Also, to Grandpa and Grandma Ayanda and Mrs Oni, I appreciate your prayers and love. In addition, I express my heartfelt thanks to my heartthrob, Evangelist Temitope Akinrinola, for his prayers, love and support. You were there every step of the journey, thank you so much for being a darling. To my lovely siblings, Mr Tijani Olatunji, Mrs Oyekanmi Olapeju, Mrs Kolawole Opeyemi, Tijani Kehinde, Tijani Taiwo, Tijani Temitayo, Tijani Afeez, Mohammed Modinat and Mohammed Ganiu, I appreciate your prayers, calls and anticipation in completing the study. Your words, *"When are you coming home? We miss you*", meant so very much. I appreciate the support, prayers and love expressed to me by my South African family, Pastor and Dr (Mrs) Aluko, and all the members of His Alive and Faithful Ministry, as well as Dr Funke Omidire and my PhD colleagues. I want to extend a big thank you to Mrs Akinrodoye, all the Computer Science departmental staff and the entire MOCPED family for their support. Thank you all!

Dedication

I especially dedicate this research study to all first year programming students in schools, colleges and universities.

"Excellent instruction is less about what a teacher does
and more about what students can do and know as a result of the lesson"

– Tony Wagner

Abstract

This study investigated first year college students' programming skills, from the visual to the procedural. The main research question sought to investigate *how* the integration of a visual programming environment (Scratch) could support the design of a new teaching and learning process framework towards the teaching of procedural (QBASIC) programming in Nigeria. In a quest to answer this central question, four secondary research questions guided the study which was situated in an interpretative philosophy and supported by a multi-method qualitative design. Using hermeneutic phenomenological inquiry, two cycles of practical action research (AR) strategy, involving five stages, were used. Data were collected through open ended and structured classroom observations, interviews, artefacts, instruments and documents. While the whole class was observed, the lived experiences of thirteen students were obtained. Data were thematically analysed based on the hermeneutic cycle principles. In the first AR cycle, the teaching and learning process framework ($TLPF_1$) guided teaching and learning. Outcomes from the first AR cycle influenced the design of the $TLPF_2$. Findings from the study revealed that the teaching intervention using the $TLPF_1$ and $TLPF_2$ promoted student learning and engagement in diverse forms. Scratch supported and enabled students to learn procedural (QBASIC) programming concepts whilst, simultaneously, enhancing their interest and motivation in this field. Understanding gained in Scratch helped students to build a correct mental representation of the Block model for procedural programming, with exception of the relations. The study also established that students learned programming through different means including: accommodation and disequilibrium, social interaction, self-regulation and problem-solving techniques. Students, therefore, suggested that Scratch programming be included in the computer science curriculum for colleges of education in Nigeria. However, contextual issues, student well-being, teachers' personalities, student behaviour, affective and behavioural states also emerged from the study as factors which can influence the learning of programming. These findings led to the design of a new $TLPF_3$ for programming in Nigeria.

Key terms: block model, hermeneutic phenomenology, practical action research, procedural programming, programming skills, teaching and learning process framework, visual programming.

Ethical clearance certificate

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Faculty of Education

RESEARCH ETHICS COMMITTEE

CLEARANCE CERTIFICATE	CLEARANCE NUMBER: **SM 15/11/02**
DEGREE AND PROJECT	PhD (CIE) Taking first year college students' programming skills from the visual to the procedural
INVESTIGATOR	Ms Fatimah Yetunde Tijani
DEPARTMENT	Science, Mathematics and Technology Education
APPROVAL TO COMMENCE STUDY	18 April 2016
DATE OF CLEARANCE CERTIFICATE	04 October 2017

CHAIRPERSON OF ETHICS COMMITTEE: Prof Liesel Ebersöhn

CC
Ms Bronwynne Swarts
Dr Ronel Callaghan

This Ethics Clearance Certificate should be read in conjunction with the Integrated Declaration Form (D08) which specifies details regarding:

- Compliance with approved research protocol,
- No significant changes,
- Informed consent/assent,
- Adverse experience or undue risk,
- Registered title, and
- Data storage requirements.

Language editor clearance

Estee Wiese
Proofreading of academic texts
BA HDE
estee.wiese@gmail.com
Mobile: +27 (0)83 6070376

Certificate of Editing

To whom it may concern

This is to certify that the manuscript detailed below was edited by an English language academic editor.

Estee Wiese
estee.wiese@gmail.com

Date:	*18/09/2018*
Manuscript Title:	*Taking first year college students' programming skills from the visual to the procedural*
Manuscript Author:	*Tijani Fatimah Yetunde*
Institution:	*University of Pretoria. Department of Science, Mathematics and Technology Education, Faculty of Education*

Esteè Wiese BA HED
estee.wiese@gmail.com
Skype: estee.wiese

Table of Contents

List of figures

List of tables

List of abbreviations

AC	Abstract conceptualisation
ADDIE	Analyse design develop implement evaluate
AE	Active experimentation
ANFE	Adult and non-formal education
AT	Assessment tasks
APOS	Action process object schema
AR	Action research
BRACE	Building research in Australian Computing Education
BYOB	Build your own block
CA	Constructive alignment
CE	Concrete experience
CAI	Computer assisted instruction
CL	Cooperative learning
CS	Computer science
DOS	Disk operating system
ECCE	Early childhood care and education
ELT	Experiential learning theory
F2F	Face to face
FORTRAN	Formula Translator
GUI	Graphical user interface
HBDI	Herrmann brain dominance instrument
IDE	Integrated development environment
ICT	Information and communication technology
ILO	Intended learning outcome
IT	Information technology
JSE	Junior secondary education
KLSI	Kolb learning style instrument
MIT	Massachusetts Institute of Technology
MCQ	Multiple choice questions
NCCE	Nigerian commission for colleges of education
NCE	Nigeria certificate in education
OOP	Object oriented programming
PC	Program comprehension

PP	Pair programming
SiPPL	Simple pedagogical programming language interpreter
SOLO	Structure of the observed learning outcome
SPED	Special needs education
SQR	Secondary research question
TA	Thematic analysis
TLAs	Teaching and learning activities
TLPF	Teaching and learning process framework
QBASIC	Quick Beginners all-purpose symbolic language
QBAT	QBASIC programming achievement test
USA	United States of America
VPE	Visual programming environment
VBASIC	Visual BASIC
WBTL	Whole brain teaching and learning
ZPD	Zone of proximal development

Background Axioms

Novices	This study refers to novices as a first year college of education students who are new to programming.
College of education	A tertiary institution in Nigeria whose primary aim is to train and prepare teachers who teach at the *primary* and *secondary* schools.
Procedural programming language	A programming language consisting of lists of series of instructions which tells a computer what to do in a step-by-step manner. The procedural programming language used in this study is QBASIC.
Visual programming environments	These are programming environments with simplified iconic interfaces that enable young people to run programs and create animations (Meerbaum-Salant, Armoni, & Ben-Ari, 2013). In this study, SCRATCH was used to support the teaching and learning of procedural programming.
Block model	It is a model of program comprehension designed for teaching programming to novices. It describes the core aspects of program understanding and also emphasises program reading instead of writing (Schulte, Clear, Taherkhani, Busjahn, & Paterson, 2010).
Learning approaches	In literature there are criticisms on the use of learning styles or approaches for designing instruction. In this study, learning approaches were used solely to plan instruction which allows for differentiation of learning where students are distributed into groups to foster social interaction. The research is not based on *how* students learn according to their styles.
Action research	Practical action research cycle was used. Two cycles were conducted between 2015/2016 and 2016/2017 academic session. The researcher acknowledges that there are different types of action research, but practical action research was considered because it guides teachers to conduct research in their own classroom with the purpose of improving own professional practice.

CHAPTER 1: INTRODUCTION AND CONTEXTUALISATION

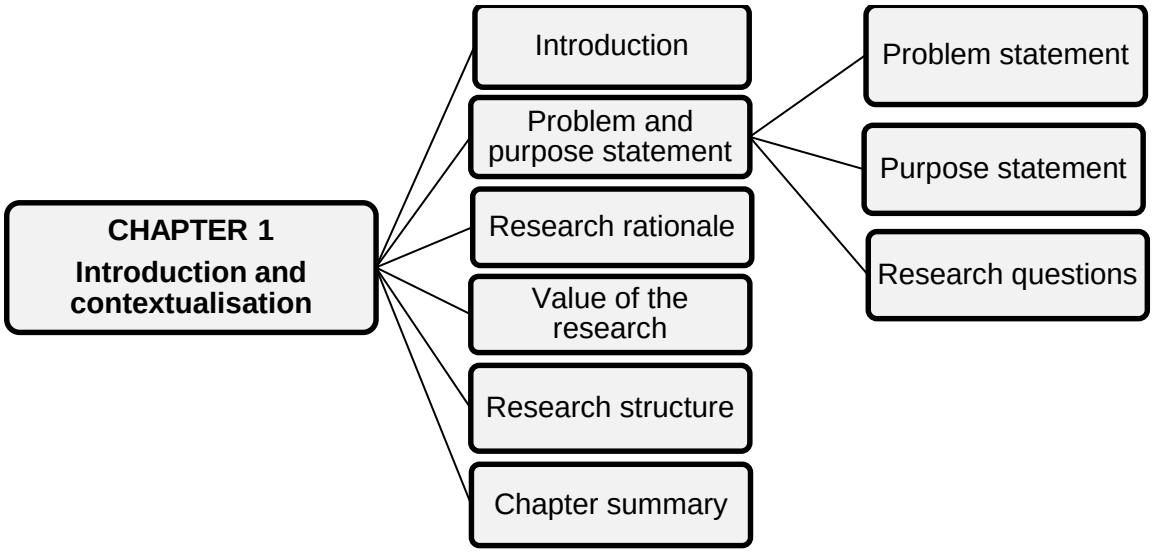


Figure 1.1: Schematic representation of Chapter 1

1.1 INTRODUCTION

The last decade has witnessed tremendous changes in Information Technology (IT). These changes have been brought about by IT developments which have affected the way people live, think and work. For these reasons, information technology skills are in high demand. The question is: Do we have highly qualified people in developing countries? In a country like Nigeria, which is the focus of this study, anecdotal evidence from the national dailies seem to suggest that there is an urgent need for well-trained computer science students, if some of the changes are to be successfully navigated (Adeyemi, 2017, January 9; Kelvin, 2016, November 13). Highly competent teachers at tertiary level, who can provide students with effective learning opportunities and technological know-how for programming, are required. Özmen and Altun (2014) indicated that an important basic skill which computer science students *must* acquire is programming. Other researchers in the field (Fessakis, Gouli, & Mavroudi, 2013; Hwang, Shadiev, Wang, & Huang, 2012b) hold the same view.

Programming involves an understanding of the activities of writing, modifying and debugging a computer program (Bergersen, 2015). Acquisition of programming skill, therefore, has been described as a vital instrument for developing problem-solving skills (Ambrosio, Costa, & Franco, 2011), higher order thinking skills (Fessakis et al., 2013) and creative thinking skills in the individual (Gao, 2011). Such skills are enhanced through programming

experience. In practice, students still develop bad programming habits (Meerbaum-Salant, Armoni, & Ben-Ari, 2011; Moreno and Robles, 2014).

Programming skill, therefore, is a student's ability to write a program from scratch and then modify and debug the said program. As a result, programming skill is a basic competence for computer science students (Fessakis et al., 2013; Hwang et al., 2012b; Özmen and Altun, 2014; Verdú et al., 2012). Programming skill develops professional and creative thinking skills in the individual (Gao, 2011). In addition, programming skill develops higher order thinking skills and, in return, produces a positive impact on a students' cognitive development (Özmen and Altun, 2014). It thus facilitates the nurturing of problem-solving skills at all levels of education (Ambrosio et al., 2011; Bergersen and Gustafsson, 2015). These highlighted skills can only be developed and honed through experience.

Programming necessitates the acquisition and thorough knowledge of various rules by the student, including semantic and syntactic knowledge (Chen and Du, 2012) as well as coding and algorithmic knowledge (Govender, 2009). Generally, most students experience difficulties in understanding some, or all, of these knowledge structures. Coding, for example, is a skill which involves the typing of rigid syntactic rules and, whilst acquiring this skill, students often make errors. Various studies (Gomes and Mendes, 2007; Govender, 2009) have affirmed that learning to program is considered hard work which is sometimes boring and often difficult to grasp. Some of the contributing factors to students experiencing difficulties in programming have been shown by studies to include: the complex nature of programming teaching (Koulouri, Lauria, & Macredie, 2015), the idiosyncratic nature and complex syntax of programming (Altadmri and Brown, 2015; Topalli and Cagiltay, 2018), gender and mathematics performance (Lau and Yuen, 2010; Sullivan and Bers, 2016; Yurdugül and Aşkar, 2013a) as well as problem solving skills (Yurdugül and Aşkar, 2013a). In addition, programming inexperience (Jegede, 2009), lack of understanding and high levels of abstraction (Ambrosio et al., 2011; Bergersen and Gustafsson, 2015; Özmen and Altun, 2014) have also been acknowledged as factors affecting students' understanding of programming. The resultant and cumulative effect of these difficulties greatly contribute to students repeatedly failing, losing interest and often dropping out of programming (Law, Lee, & Yu, 2010; Tan, Ting, & Ling, 2009).

One of the promising approaches that may address some of these identified issues is the linking of visual programming environments to support the learning of text-based programming. In addition to this, the use of educational models for planning programming teaching and determining students' mental representation of programming concepts have

also been recommended (Schulte, 2008). It is believed that if programming courses are supported within a visual programming environment which determines students' mental representation, students would gain a better understanding of programming concepts which will foster a heightened interest in programming. Thus, teachers will be able to better monitor students' progress in program understanding.

Previous studies, investigating the efficacy of visual programming for supporting text-based programming, seem to focus on middle schools and universities in *developed* countries, like the United States of America (USA) (Denner, Werner, & Ortiz, 2012; Malan and Leitner, 2007; Mladenović, Krpan, & Mladenović, 2017b; Ouahbi, Kaddari, Darhmaoui, Elachqar, & Lahmine, 2015; Wolz, Leitner, Malan, & Maloney, 2009). Studies situated in *developing* African countries pertaining to secondary schools and university support novices' understanding of programming and have attained promising results (Ajayi, Olajubu, Ninan, Akinboro, & Soriyan, 2010; Koorsse, Cilliers, & Calitz, 2015; Van Zyl, Mentz, & Havenga, 2016). However, studies which focus on the experiences and needs of pre-service teachers, are lacking.

In Nigeria, the focus of this study, there has been a dearth of research regarding pre-service teachers who should impart basic programming knowledge to learners in primary and secondary schools. To fill this gap, further studies in computer programming are necessary. The researcher's aim in this study was to ascertain ways in which to best make programming skills accessible to this group of students whilst exposing them to programming, from visual to procedural, all the while determining their mental representation of programming concepts.

1.2 PROBLEM AND PURPOSE STATEMENT

This section describes the problem and the purpose statement.

1.2.1 Problem statement

Students' learning of programming at colleges of education in Nigeria are marred by low performance rates and unsuitable programming pedagogy. The following discussion elaborates on each of these points.

Students' low performance rates

Spurred on by the pressing need for more home-grown programmers, the acquisition of computer programming skills is mandatory for all pre-service teachers studying at Nigerian colleges of education (Owolabi, Olanipekun, & Iwerima, 2014). However, pre-service

teachers in Nigeria experience procedural programming, which is used to introduce first year students to programming, as difficult to grasp. Anecdotal evidence from personal experience suggests that students find the complex syntactic and semantic structures difficult to comprehend. This is exacerbated by a lack of background knowledge in programming, a lack of practical experience as well as overcrowding of the curriculum. All these listed problems negatively impact upon students' continuation of the course. It is interesting to note that many students who perform well in other non-programming courses indicate that they find the programming course demanding (Owolabi, Adebayo, Amao-Kehinde, & Olayanju, 2015). Therefore, students are sometimes advised to re-enrol for other less taxing courses in different study fields since the programming course failure rate is so high (Akinola and Nosiru, 2014). Indeed, research conducted by programming educators and professionals in the Nigerian context show an unusually low performance rate. Owolabi et al. (2014) investigated the interactive relationship between mathematics ability and anxiety, computer and programming anxiety as well as age and gender. Using a Pearson Product Moment Correlation and multiple regression analysis, they ascertained that the effect of these factors on QBASIC programming was 20.8%. However, there was a positive significant correlation ($F = 7.869 < 0.05$) between mathematics ability and QBASIC programming achievement. If students' programming ability was as low as 20.8%, measures need to be put in place to support the learning of programming amongst first year pre-service teachers in Nigeria. Some measure of confidence would reduce anxiety and the visual programming environment might help to facilitate this process.

Programming pedagogy

Teachers' pedagogical knowledge is a crucial prerequisite for improving programming instruction in the classroom. Effective programming teaching encompasses pedagogical practices, learning theories and instructional materials to support diverse learners and access knowledge. Literature shows that Nigerian programming lecturers use the lecture method to teach (Ajayi et al., 2010) and that they hardly ever apply different teaching methods to help link abstract programming concepts to real-life situations (Ado, 2016). Even more importantly, in some Nigerian states, students attending colleges of education are exposed to theoretical aspects with little practical knowledge of programming (Isong, 2014). The result is that these students do not have the required teaching skills to address practical aspects when they finally do start working. The problem of pedagogy also relates to the assessment of programming.

Programming assessment: Another barrier to students' understanding of programming, as mentioned by Whalley, Clear, and Lister (2007), is that most programming educators lack

the frameworks necessary to design complex and challenging programming assessment tasks. The Nigerian Commission for Colleges of Education (NCCE) curriculum stipulates that students should be able to write the computer program and process data with maximum speed and accuracy (NCCE, 2012). The implication of 'writing' thereby compels lecturers to focus on writing at the expense of other comprehension skills. A new move towards the learning of programming entails: reading, tracing, explaining and understanding a given program (Schulte, 2008; Zhang, 2007). This is also why the Building Research in Australian Computing Education (BRACE)-let project emphasised the design of programming assessment based on the revised Bloom's and Structured of the Observed Learning Outcome (SOLO) taxonomy. The aim is thus to generate a theory on novice programming assessment and to make predictions regarding novices' mental representation of programming problems and performances on test items. Programming assessment involves taking into consideration the accepted set of fundamental programming skills (Lopez, Whalley, Robbins, & Lister, 2008; Whalley and Robbins, 2007). Internationally, scholars have applied the revised Bloom's and SOLO taxonomies for the design of programming assessment in their context. However, studies which address programming assessments, using the taxonomies within the Nigerian context, are still lacking and this poses a major impediment. Therefore, the careful planning of programming instruction in combination with cognitive assessments which measure students' mental representation through the explanation of relationships among concepts and processes to reveal the structure of students' knowledge, is necessary. To achieve this, Schulte (2008), educational model for improving students' mental representation of programming concepts and supporting the planning and teaching of programming lessons, guided this study.

In summary, research studies conducted by (Ouahbi et al., 2015; Rizvi and Humphries, 2012) into new visual programming environments have reported on their ability to minimise students' programming problems. Nigerian programming lecturers have used visual programming environments to support students' learning of programming at university level (Ajayi et al., 2010; Quaye and Dasuki, 2017), however, at college level, studies are sparse (Cetin, 2016). The need to support the teaching of procedural programming at college level with emerging visual programming environments exists (Dillon, Anderson, & Brown, 2012). This should be done whilst determining students' mental representation of programming concepts towards the development of programming skills. Acknowledging this need has necessitated the exploration of programming skills of first year students, from visual to procedural programming. This study will, therefore, inform the design of a new teaching and learning framework for programming teaching in Nigeria.

1.2.2 Purpose statement

The purpose of this study is to explore ways in which a visual programming environment can improve programming skills among first year college students in Nigeria. The study also investigated students' mental representation of programming concepts in accordance with the Block model. The researcher focused on students' *lived experiences,* from visual to procedural programming. This perspective facilitated the design of a teaching and learning framework for programming in Nigerian colleges of education.

1.2.3 Research Questions

The following research questions were derived from literature:

1.2.3.1 Primary research question

The proposed study was guided by the following primary research question: *How can the implementation of a visual programming environment inform the design of a new teaching and learning framework and as well support the learning of procedural programming skills in such a manner that teaching intervention planned upon students' learning approaches is positively experienced while promoting mental representations of the Block model in first year college students?*

1.2.3.2 Secondary research questions

To investigate the primary research question, the following secondary research questions (SRQ) were addressed:

SRQ1: How do first year college students in the procedural programming classroom approach learning?

SRQ2: How do the participating students experience the teaching intervention designed based on their learning approaches with a view to promote the learning of procedural programming?

SRQ3 How does the visual programming environment (VPE) support learning of procedural programming in first year college students?

SRQ4 What are the mental representations of the Block model held by first year college students in the procedural programming classroom?

1.3 RESEARCH RATIONALE

Motivation to conduct this study was shaped by the researcher's experience as a computer science student at college, from September 1997 to December 2000. At this point in time the teaching of programming was teacher-centred with little practical training for students. The researcher later served as a lecturer in the computer science department of the same

college where she taught computer science courses, including the introductory procedural programming course. This resulted in her *teaching* programming using the same method that she had been *taught* with. She believes that her unique understanding of the context and day to day involvement with first year students created an awareness and interest in students' challenges and struggles regarding introductory procedural programming. In 2012, the National Commission for Colleges of Education (NCCE), which is the third regulating body for higher education in Nigeria, reviewed its Nigeria Certificate in Education (NCE) teacher training curriculum. This revision was necessitated by public criticism that the *old* teacher training curriculum was not well enough equipped to develop specialised basic education teachers (Otuniyi, 2013). The new curriculum, therefore, makes provision for the training of quality teachers who are specialists in different areas including: early childhood care and education (ECCE), adult and non-formal education (ANFE), junior secondary education (JSE) and special needs education (SPED) (Curriculum Implementation Framework, 2012). The new curriculum encourages *competency-based* learning as opposed to *content-based* learning. The teaching of programming starts in the junior secondary school. Hence, students need to be guided by professionals to function optimally in their prospective field/s and successfully meet new challenges. The researcher became motivated to teach computer science to these students and embraced the idea of teaching programming by employing a competency-based learning approach, thus assisting her students in developing programming skills by departing from the didactic, teacher-centred method of teaching. She set out to improve her professional learning whilst employing novel approaches to benefit her students. She wanted to enable her students to program in a holistic way and socially construct meaning/s within the learning environment.

1.4 RESEARCH DESIGN

This study follows an interpretive paradigm. The methodological choice for data collection was a multi-method qualitative approach in which hermeneutic phenomenology was used as a method of inquiry. Within the qualitative research design, two cycles of practical action research strategy, which spanned two semesters between the 2015/2016 and 2016/2017 academic sessions, were used. The strategy was to study first year college students' programming skills, from visual to procedural programming. Practical action research guided the researcher in reviewing her own teaching. Multiple sets of data were collected via classroom observations, interviews, artefacts, student reflective journals and the researcher's journal.

1.5 SIGNIFICANCE OF THE RESEARCH

This study aims to address gaps in the literature by providing a foundation upon which other literature will be added to ameliorate the teaching and learning of programming, both nationally, and internationally. Methodologically, classroom teachers, programming educators and professionals in the field of computer science can adapt the practical action research strategy, as used in the design of this study, to explore and develop their students' programming skills. They can also consider investigating students' understanding of programming using hermeneutic phenomenology which serves to provide philosophical meaning to students' programs. Practically, classroom teachers, programming educators and professionals might consider implementing the teaching and learning process framework for programming in their classrooms, and for further research. The results of their research can be shared on local and international forums to enhance the parity in teachers' research and teaching. In addition, curriculum developers may also consider the inclusion of visual programming environments into secondary and tertiary curriculums. The result of this study will be communicated to the management of the college through a position paper, thus providing institutional support for the implementation of the programming framework for the teaching of programming at the college.

1.6 RESEARCH STRUCTURE

A synopsis of the thesis, divided into seven chapters, is forthwith provided. The well-articulated research report presents a coherent whole in response to the stated research questions.

Chapter 1: Introduction and contextualisation

This chapter presents an introduction to the study by referring to issues which impact on the learning of programming, both internationally and nationally. The consequent problem and purpose statement itemise and explain problems relating to the teaching and learning of programming in Nigeria. The section further discusses the purpose of the study as well as the research questions which govern it. The rationale for conducting the study, as well as a brief explanation as to the research design, is presented. The chapter concludes with an explanation as to the significance of the research and research structure.

Chapter 2: Programming and program comprehension

Chapter 2 discusses literature on programming education and focuses on issues relating to expert and novice programmers, programming paradigms and procedural and visual programming. Following this, literature regarding teaching programming in Nigeria is reviewed with a special focus on colleges of education. The chapter concludes with a

discussion on the program comprehension process and program comprehension models as well as a summary.

Chapter 3: Perspectives on learning and learning theory

Chapter 3 presents a continuation of the literature review. The discussion centres on perspectives about learning and learning theories. Theories of learning, relevant to the study, including cognitive and social constructivism, are discussed. Other areas of learning, including learning styles, learning taxonomy and assessment are also dealt with. The chapter identifies gaps in the literature and reflects on *how* the study wished to address said gaps. A synthesis of the two chapters produces a teaching and learning process framework for the teaching of programming which is modified further at the end of each of the action research cycles. The chapter is concluded with a summary.

Chapter 4: Methodology and methods

Chapter 4 of the study presents the methodology. This is further divided into three sections viz-a-viz: research design process, data collection process and data analysis process. The research design process focuses on the philosophy, research approach, strategies, time horizon as well as the techniques and procedures employed in the study. Data collection processes discus the different methods used for data collection, while the data analysis centres on hermeneutics and thematic analysis of data. The ethical guidelines and quality criteria adhered to in the study are explained and the chapter concludes with a summary.

Chapter 5: Results and findings - Action research cycle 1

Chapter 5 presents the results and data analysis of the first action research cycle. The results are comprised of data collected from: surveys, observations, interviews, artefacts and documents. The findings section discusses the data analysis and generated themes. The chapter concludes with a meta reflection and summary.

Chapter 6: Results and findings - Action research cycle 2

Chapter 6 presents the results and findings of the second action research cycle. The qualitative data collected from the surveys, interviews, observations, artefacts and documents are presented. The findings section focuses on themes generated from the data analysis. The chapter concludes with expert's reflection on the TLPF, meta reflection and summary.

Chapter 7: Conclusion and recommendation

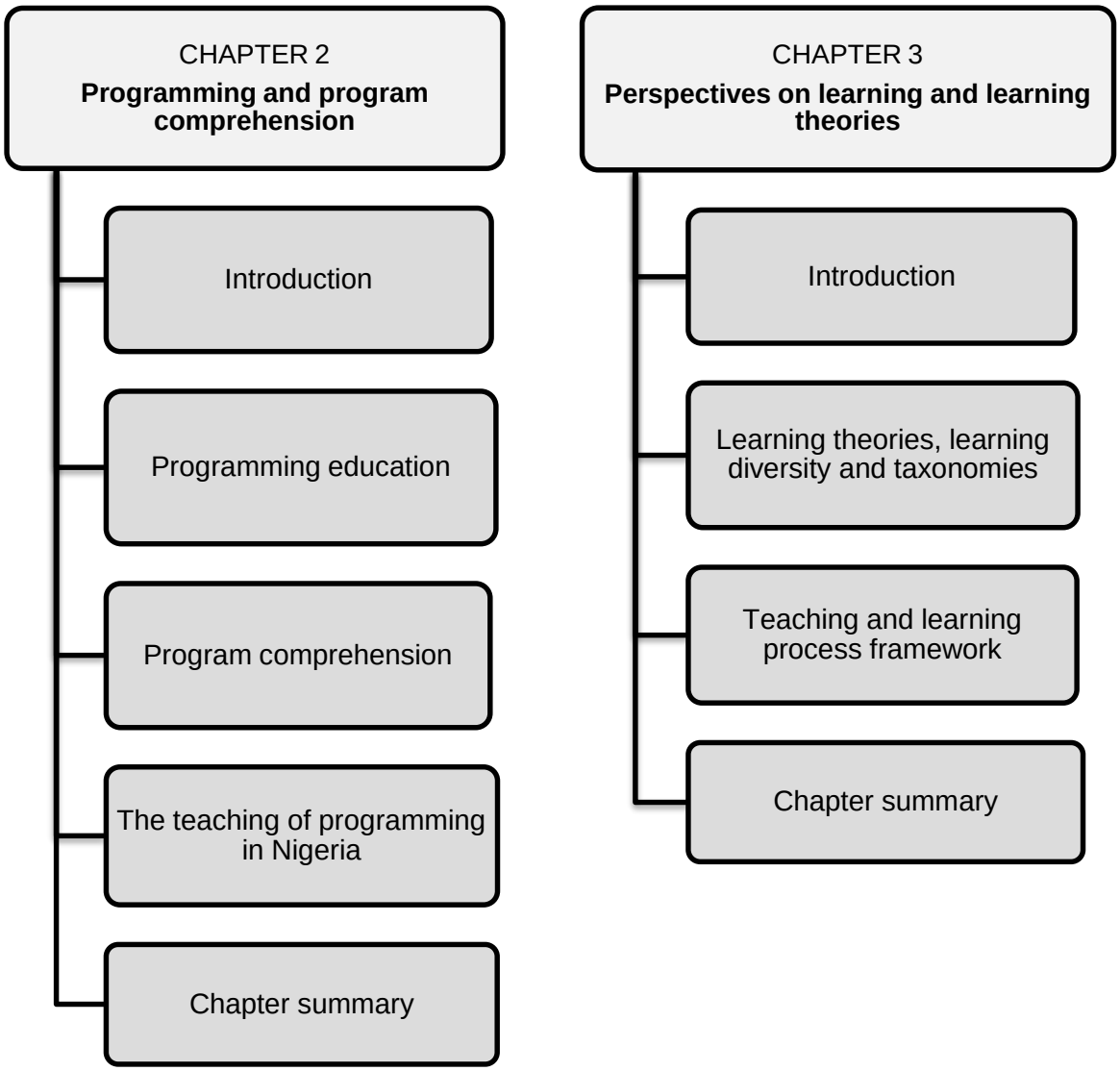
Chapter 7 presents the literature control which supports findings from literature while the conclusion details *how* the research questions were answered in relation to the conceptual framework. Finally, the research concludes with study contribution, study limitation, research and policy recommendations.

1.7 CHAPTER SUMMARY

In summary, Chapter 1 introduced and contextualised the study. This was done in an effort to focus the research, as detailed by the research questions. Research design (*q.v.* section 1.3) provided a glimpse into *how* the study was designed. Section 1.5 exemplified the theoretical, methodological and practical value of the research. The final section, which presented the research structure, offered a synopsis of each chapter in the thesis.

LITERATURE REVIEW CHAPTERS

The literature review is divided into two chapters. The first chapter, (*q.v.* Chapter 2) discusses programming and program comprehension. The second chapter of the literature review (*q.v.* Chapter 3), describes perspectives on learning and learning theories. The literature map below illustrates the concepts discussed in each chapter.



Schematic representation of literature review chapters

CHAPTER 2: PROGRAMMING AND PROGRAM COMPREHENSION

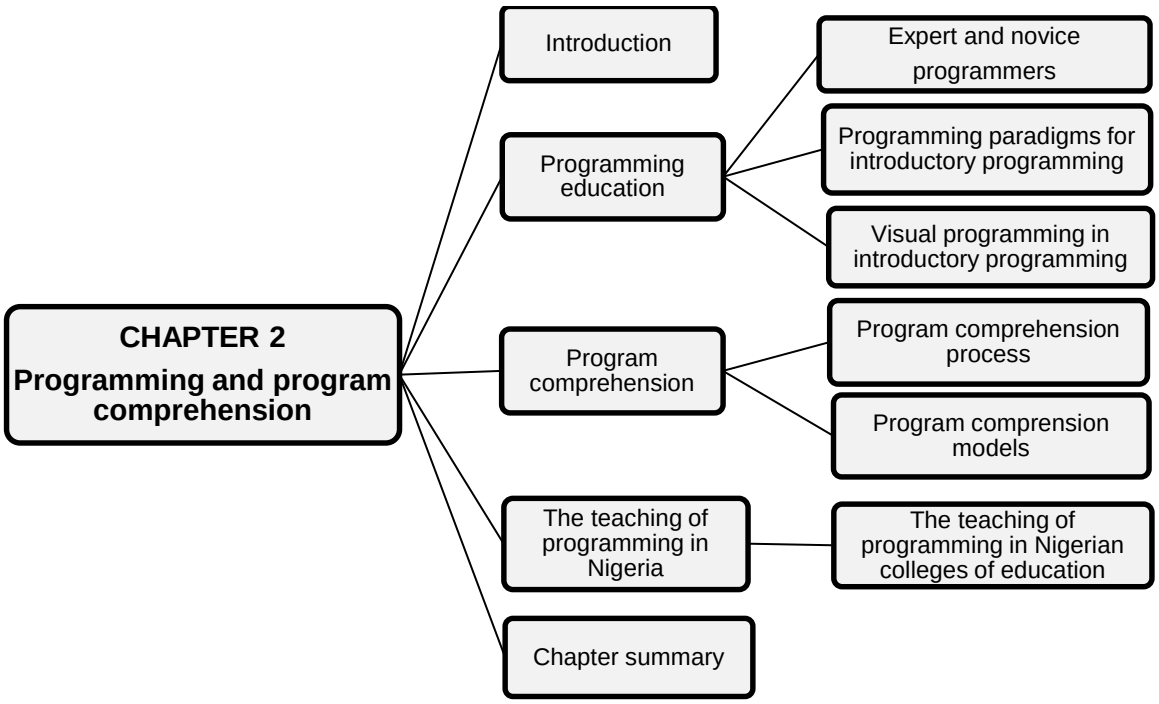


Figure 2.1: Schematic representation of Chapter 2

2.1 INTRODUCTION

This chapter discusses programming and program comprehension. It commences with an explanation of programming education whilst detailing the ways in which programming education curricular is offered in the Nigerian context. The chapter continues with a description of *novice* and *expert* programmers as well as the differing ways in which they comprehend a program. Programming paradigm follows, with an explanation as to the types of programming languages. Further discussions on procedural and visual programming languages for the teaching of introductory programming are detailed. This section also discusses the difficulties experienced by novices in procedural programming and the strength of visual programming environments in alleviating novices' difficulties. Further review focuses on program comprehension with explanations regarding its processes and models of comprehension in novice and expert programmers. A further explanation of program comprehension details how novices develop a mental representation for program understanding. The chapter continues with an explanation of programming teaching in the Nigerian context, highlighting issues which relate to programming pedagogy and programming curriculum. The chapter concludes with a summary of all the sections.

2.2 PROGRAMMING EDUCATION

The society we live in is permeated by technology. People are increasingly using technological devices and these devices are becoming more versatile. In this era of technological advancement, computer science literacy refers to individuals' ability to become technologically literate in this competitive market (Ezziane, 2007). In promoting technological literacy, programming, which is a core aspect of computer science, is important (Lau and Yuen, 2011; Vujošević-Janičić and Tošić, 2008). Since it fosters technological literacy (Lau and Yuen, 2011), programming should be used to prepare students for the workplace. The study of programming involves two different perspectives: software engineering and psychological/programming education (Robins, Rountree, & Rountree, 2003). *Software engineering* focuses on expert programmers who design systems in teams, while *programming education* deals with the teaching of programming to novices. It also requires the learning of some software engineering practices. This study is based on the *psychological/educational perspective*.

From an educational perspective, the major aim of teaching programming to novices is to help them develop *fundamental programming skills* (Gao, 2011) and *problem-solving strategies* (Gomes and Mendes, 2007). However, programming involves complex cognitive skills which necessitates that a programmer build an abstract representation of a program structure (Wiedenbeck, Labelle, & Kain, 2004; Yurdugül and Aşkar, 2013a). Novices' first programming course is during their first year at the tertiary institution. The general name for this first programming course is *introductory programming*, with varying course codes in different contexts. Over the years, research into introductory programming has shown that novices experience challenges solving programming problems after an introductory programming course (Lister et al., 2010). One of the reasons pointed out in literature is novices' inability to apply abstract concepts during problem solving (Gomes and Mendes, 2007). A novice needs to attain three different interconnected knowledge *types* in programming to enable him/her to successfully abstract programming concepts and solve problems. These knowledge types are: (1) *syntactic knowledge* which involves gathering basic facts and rules governing the use of a programming language; (2) *conceptual knowledge* which involves the understanding of programming language structures, the building of a mental representation, the execution processes of a program and developing the ability to solve programming problems by effectively applying conceptual knowledge and syntactic knowledge to build (3) *strategic knowledge* (Lye and Koh, 2014; McGill and Volet, 1997). It is important to provide background information and characteristics of the novice programmer, as well as ways in which they differ from expert programmers, to ascertain possible reasons for their inability to solve programming concepts.

2.2.1 Expert and novice programmers

The major aim of teaching programming to novices is to develop their programming skills. Developing programming skills, however, presupposes the locating of important parts of the program and appropriately forming relationships between them. These fledgling skills are nurtured in the novice programmer and they are honed to maturity when this programmer finally becomes an expert. Building expertise involves having a rich knowledge base of the domain which comprises the facts, concepts and principles inherent to the domain (Schunk, 2014). When comparing *expert* and *novice* programmers, literature explains that novices do not possess many of the abilities ascribed to experts (Brooks, 1990; Jimoyiannis, 2011; Robins et al., 2003; Winslow, 1996). The differences between novice and expert programmers hinge on three points: *skill acquisition*, *abstract representation of knowledge* and *knowledge organisation.*

In *skill acquisition*, experts employ a *working forward* strategy to identify the problem format and to then obtain an approach which matches it. These programmers are good at identifying key patterns of schemas in the given problem and relating them to background knowledge by generating a smaller number of potential solutions (Robins et al., 2003; Schunk, 2014). As observed by McCracken et al. (2001), from a study conducted at four universities, novices cannot make a program, run because they lack the necessary skills. Experts form *abstract representations* of the code while novices form concrete representations of the code (Jimoyiannis, 2011). Supporting the statement, Eckerdal et al. (2006) argue that novices have weak mental models of programming constructs which affect how they program. Therefore, students form their own mental representation of the program. They also lack the skills necessary to abstract program parts and comprehend the relationship between them in order to form a meaningful whole (Robins et al., 2003).

Schunk (2014) presented a summary of experts' characteristics: they *organise knowledge* hierarchically using a deep structure, they *monitor* their performance while working and they *weigh* the value of the strategy used, they spend time *planning* and *analysing* until a strategy to solving the problem has been found. They also possess a large body of procedural knowledge, good information processing skills and organisation of skills in diverse domains (Toker and Moseley, 2013). In contrast, novices solve problems in a piecemeal fashion because of poor organisation in memory and use trial and error to work backwards from the problem given (Schunk, 2014). They organise knowledge using a surface structure, use general knowledge instead of specific knowledge to solve problems, fail to plan and lack

detailed mental models which lead to them approaching programming problems on a line by line basis, instead of building meaningful chunks (Jimoyiannis, 2011; Robins et al., 2003).

Becoming an expert programmer is a process. It takes more or less ten years for a novice to evolve into an expert programmer (Winslow, 1996). The process, as identified by Winslow, can be broken down into the stages of: (1) novice, (2) advanced beginner, (3) competent, (4) proficient and (5) expert. Given this stratification, it is worthwhile to state that this thesis is based on novices (first year college students) and *not* experts. It is thus clear that these students will exhibit the characteristics of novices, as explained before. The next section discuses programming paradigms for teaching introductory programming, as well as novices' difficulties in procedural programming which is the focus of this study.

2.2.2 Programming paradigms for introductory programming

In teaching programming to novices, the choice of a first programming language, and its related programming paradigm, is important (Vujošević-Janičić and Tošić, 2008). A programming paradigm is complex, as it includes aspects of programming languages, a set of programming styles and the methods used by a language to describe processes for solving problems (Hofstedt, 2011; Stolin and Hazzan, 2007). This means that programming relies solely on the language itself (Yeh, 2009). The software community has profiled many programming languages, which are classified based on varying programming paradigms. Of all the numerous programming paradigms, four are fundamental to the teaching of introductory programming namely: (1) procedural (imperative), (2) functional, (3) logic and (4) object-oriented programming (Stolin and Hazzan, 2007; Vujošević-Janičić and Tošić, 2008). Each of these paradigms has their own special way of thinking, with supporting programming languages.

Procedural (imperative) is a programming style that provides varying commands for the structuring and manipulation of codes (Vujošević-Janičić and Tošić, 2008). It allows the programmer to state the computations that change the program code using procedures. The data and procedure (sequences of statements) of this paradigm are separate, with the instruction manipulating input data to produce outputs (White and Sivitanides, 2005). The advantage of a procedural approach to programming is that it is straightforward (Aleksic and Ivanovic, 2016). Typical examples used for teaching introductory programming are PASCAL, QBASIC and C language.

A functional programming paradigm is based on abstract and mathematical functions which allow a programmer to reflect on a problem at a higher level of abstraction (Louden and Lambert, 2011). Common examples are Lisp, Scheme and Haskell programming language. *A logic programming paradigm* is based on a logical declaration to the solution of a problem using axioms, rules and a goal statement (Vujošević-Janičić and Tošić, 2008). A common example is PROLOG.

Object-oriented programming (OOP) paradigm focuses on objects and their attributes, performing an operation between objects to produce data as result (Mössenböck, 2012; Vujošević-Janičić and Tošić, 2008).Typical examples include Java, C++, C# and Python.

There has been a paradigm shift from procedural to object-oriented programming (Sajaniemi and Kuittinen, 2008), and therefore OOP has been at the centre of programming classroom teaching for the past two decades (Eid and Millham, 2012). The possible reason for the popularity of OOP languages is their real-life constructs (Robins et al., 2003) and program development which is generally used in industries (Aleksic and Ivanovic, 2016). Sajaniemi and Kuittinen (2008) noted that this shift has not been motivated by any research in computer science or psychology of programming and may not be the right approach for introducing programming to students. They added that learning programming should be based on developing skills and *not* on a programming paradigm. In addition, the identification of objects is not easy and bringing together the program and problem domain is quite difficult. This may explain why OOP is difficult to learn for novice programmers. Nevertheless, support for the teaching of OOP exists because it presents an easy way of creating a solution to real-world problems (Burton and Bruhn, 2003; Wiedenbeck, 1999). Other programming paradigms are script programming, visual programming and concurrent programming (Vujošević-Janičić and Tošić, 2008).

Teaching introductory programming is imperative as it lays the foundation for learning other programming languages. Numerous programming languages have rapidly developed over the years in numerous different countries. Consequently, the choice as to *which* programming language to use in teaching introductory programming has sometimes been controversial (Aleksic and Ivanovic, 2016). A research study conducted by researchers on the state of 1 019 programming subjects in 143 universities based in 35 European countries revealed that procedural programming languages are mainly used for teaching introductory programming, even though object-oriented programming is used by most programmers. This was confirmed by an earlier report by Mason, Cooper, and de Raadt (2012) based on a 2010 survey on 28 universities in Australia. In contrast, a study done by Bennedsen and

Caspersen (2007) on introductory programming notes the distribution of paradigms used for teaching introductory programming by different countries around the world as: object-oriented (49%), procedural (25%), functional (9%) and other (17%). Procedural and object-oriented programming paradigms are thus mostly used for the teaching of introductory programming to novices. Despite their general acceptability for teaching introductory programming, novices still have difficulties in learning programming (Milne and Rowe, 2002; Özmen and Altun, 2014). This study focuses on the procedural paradigm and the next section will discuss difficulties experienced by novices in procedural programming.

2.2.2.1 Procedural programming and difficulties experienced by novices

Information Technology (IT), as a discipline, has prompted many researchers to study the best way/s in which to teach programming fundamentals to enhance greater understanding and retention rates (Burton and Bruhn, 2003). Procedural programming is believed to aid students in comprehending the fundamental building blocks of programming (Godbole, 2014). Procedural programming separates data from procedures, with the *instructions* manipulating data and the *procedures* (sequences of statements) converting the input into output (White and Sivitanides, 2005). Procedural programming involves the breaking down of programming tasks into a collection of variables, data structures and subroutines. Never the less, learning procedural programming involves the acquisition of *syntactic* and *semantic* knowledge of a particular programming language. The acquisition of these knowledge structures are essential to the creation of programs in a language but often prove difficult for novices to comprehend. These difficulties, as experienced by novices in procedural programming, are discussed in the next section.

- *Syntactic and semantic issues*

The syntax of a programming language contains a set of unnatural rules which are not familiar to novices. Syntax is cognitive and of not much value to the program without a high-level construct in programming. Since only a syntactically correct program can be executed by the computer, it is possible that novices neglect the creation of an algorithmically correct program by focusing on the syntactically correct program (Grandell, Peltomäki, & Salakoski, 2005; Vasilopoulos, 2014). The semantics of a programming language are concerned with deriving meaning from the use of programming language constructs. The incorrect use of the semantics of a programming language results in the program giving an *error*. An example of this is when a programmer misconceives a program command and uses “IF-THEN” instead of “DO WHILE”. This misconception may be due to insufficient knowledge and/or a problem which relates to natural human interaction (Guibert, Girard, & Guittet,

2004). However, simply *having* knowledge of the syntax and the semantics does not necessarily guarantee that one has developed a deep understanding of programming skills (Vasilopoulos, 2014). The language of a program is based on the operations which the computer can perform, and novices do not generally have knowledge pertaining to the programming language domain. They interpret the commands of a programming language in terms of the rules governing their experienced and familiar interaction. They are not aware of the low-level processes used in processing data and executing commands (Ben-Ari, 2001). Thus, they expect computers to respond to them as humans do. Semantic errors can be frustrating to novices since a lack of understanding, or the incorrect use of concepts, causes the program to stop running. For example, the *word* "while" in the English language is semantically different in meaning to the *command* "while" in procedural programming languages (Vasilopoulos, 2014). Solving everyday real-world problems through the creation of programs entails an understanding of both the semantic and syntactic knowledge of a typical programming course. Previous research studies (Dagdilelis, Satratzemi, & Evangelidis, 2004; George, 2000; Kinnunen and Malmi, 2008; Ma, Ferguson, Roper, Ross, & Wood, 2008; Milne and Rowe, 2002) reported that the semantic concepts which novices find difficult in procedural programming are: variables, input/output statements, assignment statements, iteration statements, conditional statements, arrays, recursion and procedures. Therefore, a novice programmer should not only have knowledge of a programming language's syntax and semantics, but must also know how to use these constructs to solve problems programmatically (Vasilopoulos, 2014).

- *The notional machine's mental model*

A mental model is an internal representation that describes an aspect of an individual's environment (Sorva, 2013). Programming involves developing many mental models (Mow, 2008). Firstly, programs are written with the aim of solving a real-life problem, and novices need to construct a mental model for devising a solution and writing a correct program that solves this problem. Secondly, novices' construction of a machine's mental model is necessary when trying to program. A notional machine, as conceived by du Boulay, O'Shea, and Monk (1981, p. 237) is *"an idealized, conceptual computer whose properties are implied by the constructs in the programming language employed".* The property of the notional machine is embedded in the programming language to be learnt, and the mental model of that machine can only be formed through the program code (Sorva, 2013). Thirdly, the novice programmer needs to form a mental model which details how the execution of the current program will take place (Mow, 2008). These varying mental models for problem solving in programming have been researched by different authors. An earlier study, conducted by Spohrer and Soloway (1986), on novices' learning of the PASCAL

programming language reveals that they held misconceptions regarding *natural* language and *programming* language. Jimoyiannis (2011) also found that students lack the mental model of the internal operation of a variable and assignment statement during program execution.

In another study, Vrachnos and Jimoyiannis (2017) found that students held incorrect representations of the array concept of small sets of code in the PASCAL program. They linked the result to students' earlier misconceptions regarding variables. Their findings confirmed the discussion in literature that many programming languages resemble the natural languages common to the novices. They thus tend to form an unsuccessful representation of the notional machine which results in misconceptions (Sorva, 2013). To support novices' development of effective mental models of the notional machine, scaffolding, coaching and good modelling are suggested in the literature (Mow, 2008).

- *Debugging skills*

Another difficulty novices face when programming is debugging, with research studies into novice debugging dating back to the 1980s. *Debugging* is an activity that occurs after running the program in order to find out *where* the error lies and *how* to fix it (McCauley et al., 2008). For novice programmers, debugging is a difficult task (Fitzgerald et al., 2008; Lin et al., 2016). Locating errors is viewed as the third most difficult skill to be acquired in programming, in accordance with Lahtinen, Ala-Mutka, and Järvinen (2005). Novices, therefore, lack the skills needed to correctly debug a program (McCauley et al., 2008; Qian and Lehman, 2017). Novices' lack of debugging skills can be related to their fragile knowledge of programming (Perkins and Martin, 1986). For example, *bugs* arise in a novice program due to misconceptions that programming instruction is related to natural language (Pea, 1986). Novices also *debug* a program on a line by line basis without conceiving a holistic view of the program (Kolikant and Mussai, 2008). They therefore devote most of their time and effort to correcting errors during programming (Vasilopoulos, 2014). They do not take time to examine the suggestions for program error origin and thus select incorrect hypotheses. Though debugging is an important programming skill, textbooks afford it little focus (McCauley et al., 2008).

- Other novice difficulties in programming relate to: error messages, programming strategies and plans (Ismail, Ngah, & Umar, 2010; Mow, 2008); recursion (Katai, 2011); problem-solving (Falkner and Palmer, 2009); arrays, loops, conditional statements and other concepts (Cetin, 2016; Ginat, 2004). To help novices understand programming constructs, Kelleher and Pausch (2005) have suggested that the *syntax* of programming could be replaced with *graphical elements* to symbolise constructs such as variables, control options

and commands. Consequently, there has been a move towards the use of visual programming to support the teaching of introductory programming courses to support novices' learning of programming (Gurudatt and Sathyaraj, 2014; Hari, Bhushan, Venkataraman, & Geeta, 2013).

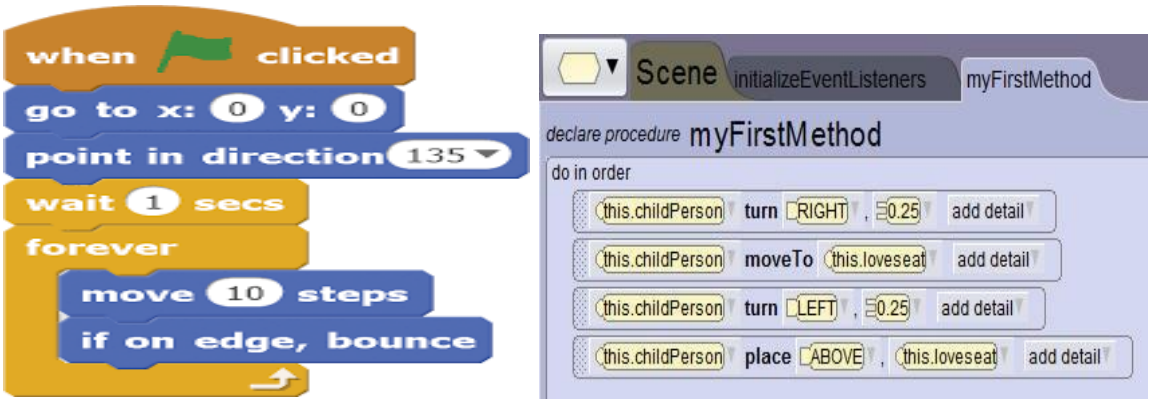
2.2.3 Visual programming languages in introductory programming

Visual programming environment, or block-based programming, supports the use of a graphical user interface (GUI) with each programming construct displayed using graphical objects (Aleksic and Ivanovic, 2016). Many arguments have been put forward toward the potential benefits of visual programming languages. One possible benefit of visual programming is the graphical dimension which introduces a fun and motivating component which is absent in its procedural, text-based counterpart (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010). Lye and Koh (2014) affirmed that novices would find *visual* programming easier to use than *text-based* programming because of the former's ability to support forward and backward reasoning, activate memory and present a visual representation of the control and data flow in a program. The visual programing environment (VPE) assists with problem-solving through the graphical presentation of problems. Novices can construct and identify important program features and finally produce their planned solutions (Kelleher and Pausch, 2005; Lye and Koh, 2014). This warrants that novices develop problem-solving strategies, such as planning and designing solutions, before they can successfully solve programming problems and develop skills (Fee and Holland-Minkley, 2010; Hazzan, Lapidot, & Ragonis, 2015). The cognitive load found in text-based programming is also reduced by chunking codes into smaller units (Cetin, 2016), thus helping students to focus on *codes* instead of *syntax* (Bau, Gray, Kelleher, Sheldon, & Turbak, 2017). Visual programming does not return error messages, unlike text-based programming.

Many programming teachers are attracted to the simplicity of VPE in teaching and learning programming as its readily available and promotes interactive engagement and dynamic outcomes (Fincher, Cooper, Kölling, & Maloney, 2010). The strength of visual programming environments may reduce difficulties encountered by novices in procedural and other text-based programming, thus allowing them to focus *less* on syntax and *more* on the sagacity and structure of a program (Kelleher and Pausch, 2005).

A series of visual programming environments (see Figure 2.2) has been developed to support novice programming (Chao, 2016). *Alice* is a 3-dimensional animation with a drag-

and-drop environment which supports the creation of stories and prevents novices from making syntax errors (Fincher et al., 2010; Kelleher and Pausch, 2005).



Scratch Alice

Figure 2.2: Samples of visual programming languages

Scratch was developed for young people and utilises a colourful command block structure which enables users to construct programs through *dragging* and *dropping* graphical objects to form sprites (Maloney et al., 2010). Other visual programming environments include: LightBot which was proposed by (Gouws, Bradshaw, & Wentworth, 2013), Kodu (MacLaurin, 2011), Raptor (Carlisle, 2009), Snap! (a variation of Scratch), amongst others. Section 2.2.3.1 further explores literature on Scratch programming.

2.2.3.1 Scratch programming

This is a visual programming environment which possesses both a social computing environment and a highly supportive rich programming interface (Ouahbi et al., 2015; Wolz et al., 2009). Scratch was developed by the Massachusetts Institute of Technology (MIT) laboratory's lifelong kindergarten group to boost the technological skills of post school youths in poor communities (Malan and Leitner, 2007; Resnick et al., 2009; Su, Yang, Hwang, Huang, & Tern, 2014). Scratch is not the first visual programming environment, but its strength provides *wider walls* where students can express themselves programmatically over other environments and this has made it gain the steady interests of researchers (Su et al., 2014). Scratch was developed with the notion that it would lower barriers to learning programming by empowering novices to master programming constructs and logic before learning real programming. Scratch also has the potential of creating interactive stories, cartoons, games, musical compositions, mathematical models and numerical simulations that could be combined in a project. Figure 2.3 presents the Scratch environment.

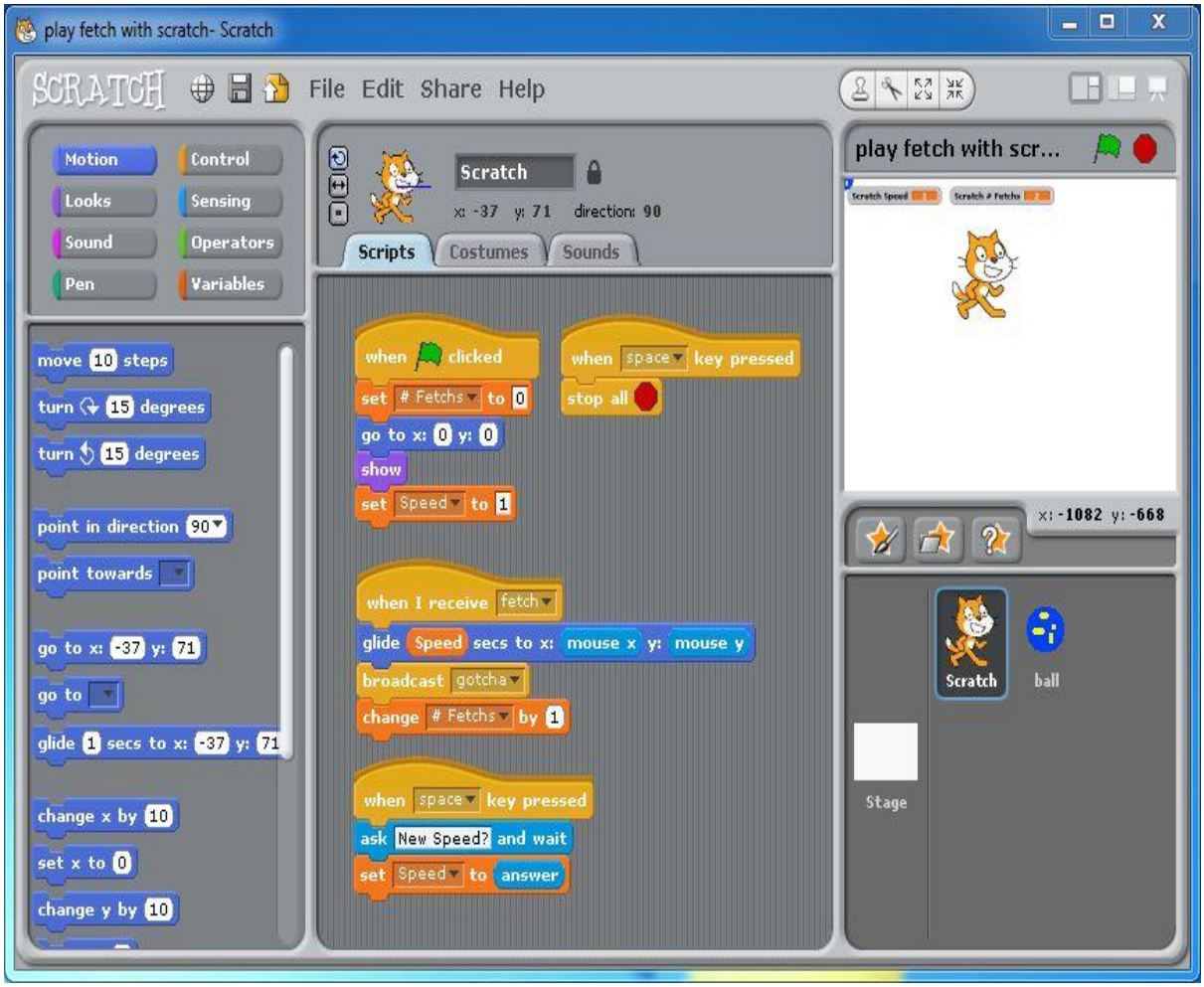


Figure 2.3: The Scratch block palette, scripting area and the stage

Harvey and Mönig (2010) state that Scratch typifies an object-based language where sets of animated objects are designed by choosing shapes from the library, or an imported file. Maloney et al. (2010) established that even though Scratch is an object-based language, it is not an object-oriented language, because it does not support the inherent characteristics of object-oriented programming languages. Thus, Scratch has some of the *features* of object-oriented programming. The Scratch environment has a block palette, script area, selection area, a stage and sprite. Programs in Scratch are called *scripts* which are created by dragging and dropping blocks on the stage. These blocks represent program components such as an operator, expressions, conditions, loops and variables or a function which snap together to form a script. The behaviour of a sprite gives visual feedback when execution takes place (Fincher et al., 2010). However, Scratch does have some limitations including a lack of procedure and weak support for data structure (Harvey and Mönig, 2010; Maloney et al., 2010). The authors added that these two limitations were not an oversight, but an intentional choice made by the designer to keep the Scratch language uncluttered for novice learning. Therefore, Mladenović, Krpan, and Mladenović (2016) argue that Scratch can still

be used to introduce introductory programming to novices (from an elementary to university level), since it is less abstract and works well without the limitations. Nevertheless, these identified limitations in Scratch are important constructs in computer science which become necessary when introducing programming to advanced students. A variant of Scratch, Build Your Own Block (BYOB), was developed by Harvey and Mönig (2010) to incorporate these features.

Programming requires that students develop computational thinking skills needed to solve real-world problems (Grover and Pea, 2013). Computational thinking is the thought process involved in articulating problems, along with their corresponding solutions, and then presenting them in a form comprehensible to a computer (Wing, 2006). This means a programmer must be able to think like a computer scientist in order to solve programming problems. Scratch has been used to teach computational thinking skills to students. Brennan and Resnick (2012) and Chao (2016) discussed frameworks for teaching computational thinking to students using Scratch programming. These frameworks are *computational thinking concepts* which focus on programming concepts such as loops and sequences which programmers explore during Scratch programming; *computational thinking practices* which refer to the practices programmers develop during programming and *computation thinking perspectives* relating to programmers' perceptions about themselves *and* the program being developed.

Sáez-López, Román-González, and Vázquez-Cano (2016) conducted a two year case study where Scratch programming was integrated into the elementary school curriculum in Spain. Programming lessons were taught using an active learning strategy. A quazi-experimental study with 107 students revealed that the *experimental* group (Scratch) showed a better understanding of programming concepts than the *control* group. Students in the Scratch programming group also reported higher levels of motivation, commitment, fun and enthusiasm. The results of this study support further studies by Weintrop and Wilensky (2017) and Mladenović et al. (2017b) who noted that students showed higher levels of interest and displayed positive attitudes towards learning programming in the future.

Mladenović, Boljat, and Žanko (2017a) conducted another study in which they compared misconceptions held by 207 K-12 students about loops in Scratch, Logo and Python. Findings revealed that students' misconceptions regarding loops were less prevalent in Scratch than in Logo and Python. The authors concluded that Scratch programming had the potential to facilitate the learning of programming in students.

In the context of tertiary education, Cetin (2016) introduced 56 pre-service teachers to computing while exploring the utilisation of Scratch towards learning programming concepts through a mixed methods approach. Findings indicated that the experimental group, which comprised of pre-service teachers who had been exposed to Scratch, displayed a better understanding of programming concepts than the control group, which had been exposed to C language. He reported that students' higher performance could be attributed to Scratch's potential to reduce syntax related cognitive load and encourage student engagement. The study further revealed that there were no significant differences in attitude toward programming between the experimental and control group pre-service teachers.

We can thus deduce that Scratch programming is effective in introducing programming concepts to novices.

2.2.3.2 Transitioning from Scratch to Text-based programming

In recent times, introductory programming courses have been taught by, firstly, exposing students to visual programming and then moving to text-based programming. Transitioning from visual to text-based programming is important if students are to become fluent programmers in the future (Shapiro and Ahrens, 2016). At Harvard, course CS50 is taught with a progression from Scratch to C language. At Berkeley, CS10 is taught from Snap! to Python (Bau et al., 2017) while others move from Scratch to Java. Malan and Leitner (2007) were the first researchers to perform a trial experiment with Scratch as budding scientists used it as a herald to Java programming. They conducted the study at Harvard College summer school with the intention of improving novice programmers' experiences. They discovered that 76% of the students rated Scratch as a *positive influence* while 16% rated *no influence*. The positive result attained by Malan and Leitner's 2007 study influenced other researchers to conduct further studies on novices' use of Scratch as a precursor to programming languages at elementary and high school level as well as colleges.

Lewis (2010) compared the learning outcomes and attitudes of Grade 6 students in terms of their understanding of loops and conditional in Scratch and Logo. Students were first exposed to Scratch before text-based Logo. Findings showed that students who learned Scratch displayed greater competence in interpreting loops and conditionals and held better attitudes toward programming. The findings supports a study conducted by Mladenovic, Rosic, and Mladenovic (2016). Meerbaum-Salant et al. (2013) conducted a mixed method research on novices who used Scratch to learn science concepts. Subjects were drawn from two classes and were aged 14 and 15 years. The groups completed pre-tests, interim and

post-tests. The revised Bloom's and SOLO taxonomies were used in designing the tests. Using descriptive statistics, students' internalisation of concepts in Scratch and CS were evaluated. The T-test results showed that student achievement in the two classes did not differ significantly. Field notes, as well as student and teacher interviews were analysed using content analysis.

Subsequently, Ouahbi et al. (2015) conducted an experiment on high school students in the north eastern part of Morocco. They used Scratch to facilitate the learning of programming concepts through the creation of games. A sample of 69 students were grouped into three. Group A was taught with Scratch while groups B and C were taught with PASCAL. Questionnaires were distributed *before* and *after* the teaching to test students' programming levels and habits, respectively. The result showed that 80% of the students found Scratch interesting and displayed high levels of motivation to learn while programming with PASCAL (73.3%) was considered as boring. This study also corroborates the findings of Sáez-López et al. (2016).

Armoni, Meerbaum-Salant, and Ben-Ari (2015) built forth on the study they conducted in 2010 in Meerbaum-Salant et al. (2013) by studying the transition from CS with Scratch to a C++/Java programming. The study adopted a mixed method approach with a sample of 120 Grade 10 learners. In terms of variables and conditional execution concepts, quantitative findings reported no significant difference between the experimental and control groups. There was, however, a significant difference in repeated execution at all levels for the experimental group. The final test revealed that the experimental group performed better at relational creating. Grounded analysis of qualitative data revealed that there was an increase in enrollment and efficiency with higher levels of motivation and self-efficacy on the part of the of students.

A recent study researched programming patterns of students in Scratch. Swidan, Serebrenik, and Hermans (2017) investigated the naming pattern of variables and procedures among Scratch programmers. Using 250 000 projects obtained from the Scratch online website in 2016, their findings show that there is a proliferation regarding the use of longer identifier and procedure names among Scratch programmers. Their study also supports the notion that students may find naming of variables difficult as they transition to text-based programming.

Another study, conducted by Weintrop and Wilensky (2015), investigated the perceptions held by high school students regarding block-based (Snap!) and text-based (Java)

programming. Snap! was used as a herald to Java programming. Findings from interviews revealed that students reported that factors such as the simplicity of blocks and the drag and drop composition contributed to the perceived *easiness* of block-based programming. Students identified: lack of authenticity, less powerful technique, long-winded blocks and slower authoring as drawbacks to the use of block-based compared to text-based programming. This study confirmed the findings of Monig, Ohshima, and Maloney (2015).

Evidence gained from research in different contexts, therefore, encourages the use of Scratch *in combination with* teaching text-based programming and not to replace it (Dorling & White, 2015). The essence of teaching programming through block-based programming is that students will be able to link block-based and text-based constructs. Unfortunately, Krpan, Mladenović, and Zaharija (2017) highlighted that a mediated transfer is not always common. Since this study focuses on procedural programming (QBASIC), a discussion as to the concepts that *can* be taught in either Scratch or procedural programming will be useful (O'Leary and O'Leary, 2013; Van Zyl et al., 2016; Wassermann et al., 2011). Table 2.1 provides a summary based on discussions in the previous section.

Table 2.1: Comparison of concepts in Scratch and QBASIC programming

Concept	Scratch	QBASIC
Interface	- Uses a graphical user interface (GUI) consisting of a block palette, scripts area and the stage.	- Use an integrated development environment (IDE) using a DOS program.
Variables and data types	- Variable must be given a name. - No rules about variable naming convention. - Shapes indicate data types which are Number, String and Boolean. - Makes decision about the type of data a variable holds. - Visual representation of variable content through the monitor. - Data types need not be distinguished.	- Variable must be given a name. - Variable naming convention must be followed. - Knowledge of different data types with their specific values is required. - Type of data type a variable will store must be declared. - Variable contents are not visible. - Data types need to be distinguished.
Syntax	- No knowledge of syntax required. Has blocks of code which fit together.	- Knowledge of syntax is required.
Program execution	- Click on green flag. - Programs will always execute.	- Press F5, and program is translated to machine code by the compiler. - Program will only execute when it is free of syntax error.
Problem solving	- Algorithm design is necessary. - Set up Sprite and create scripts. - Test the program.	- Algorithm design is necessary. - Set up interface and type code. - Correct syntax error, test the program and vice versa.
Testing and debugging	- Errors (logical and execution) can occur during program execution.	- Errors (syntax, logical and data) can occur during program execution.

	- Program can be tested step by step to resolve logical errors.	- An built-in debugging facility allows program to be executed step by step.
Decision and looping structures	- Visually represented code blocks and structures can be chosen on purpose. - Knowledge of Boolean usage is required. - Knowledge and usage of conditional and unconditional loops is necessary.	- Knowledge and usage of varieties of looping and decision structures is required.
Data structure	- Data structure is named as lists. - Values in the lists are named as items. - Item number is called index. - Knowledge and usage of program blocks for list is required.	- Data structure is named as array. - Values in the array are named as elements. - Element number is called dimension. - Array must be defined before it can appear within an executable statement.
Operators	- Logical operators are NOT, OR and AND. - Recognises two relational operators (< and >).	- Logical operators are NOT, OR and AND. - Recognises six relational operators.
Program implementation procedures	- Import sprites into the stage area. - Link scripts to the sprites by dragging code blocks from the blocks palette.	- Declare code for input, processing and output by typing the code. - Link events by typing code.

2.3 PROGRAM COMPREHENSION

The study of program comprehension is interdisciplinary in nature. Its theory builds upon text comprehension theory in psychology, philosophy, cognitive information processing and computer science (Schulte, 2008). As such, program comprehension is a fundamental aspect of program understanding which, in turn, is a core aspect of software engineering. In addition, program comprehension involves a wide range of activities such as end-user customisations to existing software, software maintenance and introductory programming courses (Dasgupta, 2010). This section reviews the key elements of the program comprehension process and the different program comprehension models.

2.3.1 Program comprehension process

The program comprehension process has been proposed and has have common elements which are: external representation, cognitive structure and assimilation process. Chen and Du (2012), as well as Schulte et al. (2010), explicitly define these elements which are represented in Figure 2.4.

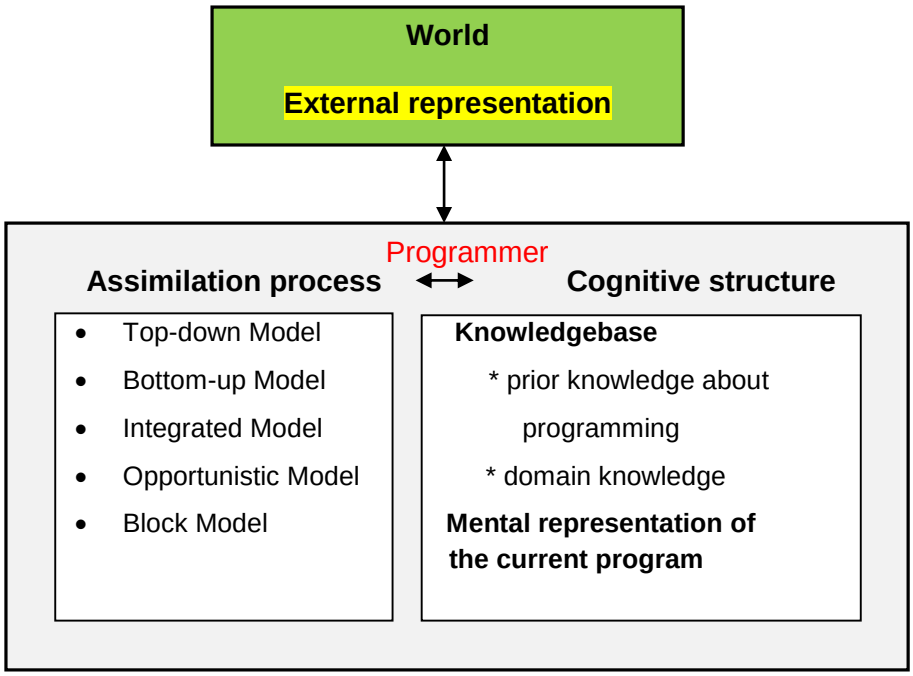


Figure 2.4: Major elements of program comprehension models

Source: Schulte et al. (Schulte, 2008, p. 2)

External representations are any "external" materials and data associated with the program being studied. These do not include the internal knowledge of the programmer but helps the programmer during comprehension of the code. The *assimilation process* concerns the programmer's method for extracting information from the program text to form a mental representation of the code. The empirically tested assimilation processes in program comprehension (PC) can be grouped into five models namely: top-down, bottom-up, integrated, opportunistic and the educational block model. Next, the *cognitive structure* is the internal knowledge of the programmer. It comprises the knowledge base of programming, the program's application domain as well as the programmer's mental representation of the program. The *knowledge base* consists of the programmer's general, language-related and domain knowledge, while the internal knowledge which the programmer built during comprehension using the knowledge base is called the *mental representation*. Frankish and Ramsey (2012) stressed that human beings make use of both unconscious and conscious mental representations to build their cognition. The concept of mental representation is further explained in section 2.4.1.1

2.4.1.1 Mental representations and students' success in programming

The study of mental representation is rooted in a cognitive science discipline, the aim of which is to study the relatedness of cognitive process to behaviour and how they are computationally realised and biologically implemented (Held, Vosgerau, & Knauff, 2006). In literature, cognitive science researchers commonly use mental representation interchangeably with "mental model". The theory of the mental model describes the cognitive interaction of individuals with the environment (Lau and Yuen, 2010). To add, the theory of the mental model can be traced back to the work of Craik (1943) who proposed that humans use mental models of external reality to understand, explain and make judgmental conclusions about the environment. Forty years later, Johnson-Laird (1985), another influential proponent in this field, offered an explicit discussion on mental models. Manktelow and Chung (2004), reporting Johnson-Laird, stressed that mental representation has three representations namely: (1) propositional representation, (2) mental imagery and (3) mental models. The *propositional representations* are language-like representations which are explicit, discrete, abstract entities that capture the real content of the mind, irrespective of the uniqueness in which that information was encountered (Eysenck and Keane, 2000; Paivio, 1990). *Mental imagery* is two-dimensional visual icons basically of an object or scene or model. In addition to this, Lau and Yuen (2010, p. 276) defines a *mental model* as "a cognitive construction of the internal (mental) representation of objects of external reality; which enables individuals to understand, explain, predict and make inferences about decisions".

The importance of mental representation in the learning of computer programming cannot be overemphasised. Programming entails developing abstract representations of a program process and giving a logical explanation of its structure (Wiedenbeck et al., 2004). Research studies on cognitive science, computer programming and other disciplines have established that for learners to understand and process complex tasks or systems, they need to be exposed to a learning that is facilitated by building appropriate mental models or an elaboration of their incomplete models (George, 2000). This means that if novices have only an understanding of the syntax of a programming language, they may not be able to solve complex programming tasks if they do not have an adequate mental model of the problem at hand. Building effective mental models are of great importance in comprehending programming concepts.

A programmers' mental model, therefore, includes knowledge about programming, problem-solving, structure and the function of a program, debugging and the syntax and semantics

of a specific language. To support this further, McGill and Volet (1997), Yurdugül and Aşkar (2013b), and Bayman and Mayer (1988), classified the kinds of programming knowledge necessary to understand the complex processes of programming that would enable a student to build a viable mental model. Firstly, conceptual knowledge *is* knowledge about understanding programming constructs and principles, development of mental models and understanding the actions of a program. Secondly, syntactic knowledge involves features of the programming language and its rules. Thirdly, strategic knowledge uses both conceptual and syntactic ideas toward solving programming problems. The success of a student in comprehending programs and solving programming problems is thus based on these three important knowledge structures.

Indeed, mental models/mental representation have been studied by researchers (Corritore and Wiedenbeck, 2001; Littman, Pinto, Letovsky, & Soloway, 1986; Nanja and Cook, 1987; Pennington, 1987; Soloway and Ehrlich, 1984; Wiedenbeck, 1999) which have established its important role in program comprehension and the comprehension of related tasks. In the last decade, some other studies have researched novices' mental representation in programming. Dehnadi and Bornat (2006), for example, observed the mental models of 60 students regarding assignment instructions in Java programming. A test was administered before *and* after students had been taught. The result of the correlation analysis showed that 44% of the students were consistent with a single mental model, 39% were inconsistent with several mental models while 8% did not answer any questions. The authors, therefore, established that success in the first stage of an introductory programming course is directly influenced by *how* students solve basic problems before their actual programming experience.

In the light of the above, Ma, Ferguson, Roper, and Wood (2007) built upon Dehnadi and Bornat (2006) work. They investigated the mental models of university first year students regarding value and reference assignment concepts in Java using Blue J. Both open-ended and multiple-choice questionnaires were used to obtain their mental models. Students were classified as *consistently appropriate*, *consistently inappropriate* and *inconsistent* based on the viability of their mental models. Findings showed that the consistently appropriate group performed better than the two other groups during formative assessment and a final examination. Students also held a viable mental model of programming concepts. These two discussed studies helped students in building mental models in Java programming. The positive outcomes might be linked to the influence of previous knowledge gained by the students in procedural programming during high school. The present research, however, is based on procedural programming.

Other studies investigated novices' mental models by mapping programming responses to SOLO. Jimoyiannis (2011) conducted a survey study on secondary school students in Greek. Students attended a DAPE course on the algorithm and introductory programming according to the Greek curriculum for two hours a week. They then completed a questionnaire based on six programming tasks, six weeks after the course was taken. Students' responses were mapped to the SOLO taxonomy and results indicated that more than half of the students manifested pre-structural, unistructural, and multi-structural responses. Also, students had built mathematical-like mental models of the variable and assignment programming concepts.

Cetin (2015) conducted a mixed methods study involving 63 mechanical engineering students who learnt C programming language in the second year of their study. Using APOS (action, process, object and schema), students' understanding of loops and nested loops were determined. Findings showed that the APOS framework was useful in examining students' understanding of mental structures in programming. This framework proved useful in that it helped students to develop writing and reading skills in programming. It did not classify student assessment tasks based on SOLO or the Bloom's taxonomies, instead, an open ended questionnaire was used. The idea of using SOLO taxonomy to classify assessment is a noticeable development over the other studies.

2.3.2 Program comprehension models (assimilation process)

This is the assimilation process required to understand a program text (q.v. Figure 2.4). Assimilation processes are referred to as models of comprehension. The models that will be explained are the top-down, bottom-up, opportunistic and block model.

2.3.2.1 Top-down comprehension model

Brooks (1983) and Soloway and Ehrlich (1984) are some of the researchers who presented a top-down model to program understanding. The top-down comprehension model assumes that comprehension is based on *first* applying knowledge about the problem domain and then mapping this knowledge to the microstructure of the code (Schulte et al., 2010; Zhang, 2007). Brooks (1983) posited that the top-down model makes use of a general hypothesis about the program which is refined by formulating an opportunistic sub-hypothesis that is driven by beacons in the code. When such beacons are found, it is possible that the programmer draws a final conclusion that the operation is present, else the programmer revises or rejects the hypothesis after a careful study of the program. Thus, comprehension is reached through a series of hypothesis refinement (Corritore and Wiedenbeck, 2001).

Unfortunately, Brooks did not test his theory, but researchers such as Wiedenbeck (1999) and Gellenbeck and Cook (1991) have tested Brooks' theory using code and naming beacons.

2.3.2.2 Bottom-up comprehension model

Pennington (1987) improved Van Dijk and Kintsch's model for text comprehension by adapting it to program comprehension. In this model, the assimilation process requires that the programmer first read code on a line by line basis and then chunk or group the code into higher level abstractions. The "chunking" or "grouping" is repeated several times until a high-level comprehension of the program is formed (Chen and Du, 2012; Schulte et al., 2010). In this respect, Pennington (1987) suggests two models. She proposed and distinguished between a program model representation and a domain model. She noted that when programmers are unfamiliar with the source code, they build an elementary mental representation called the *program model*. The program model comes first during comprehension and it represents the text base and other elementary operations of the program. Another model built from the bottom-up is called the *domain model.* This includes a mental model of the flow of data within the program, as well as goals of the program. According to Pennington's bottom-up model, the program model appears *before* the domain model.

Pennington carried out two experiments to test her model. In the first experiment, professional programmers used the recall and recognition tasks when exposed to 15 pieces of code. Findings revealed that the programmers better answered questions on control flow than function questions. In the second study, high and bottom level comprehenders were exposed to a 200-line FORTRAN program which they studied for 45 minutes prior to a modification task. They think aloud as they read the program and solved 20 comprehension questions for 30 minutes. Findings show that only programmers with high levels of comprehension could cross-reference between the program and domain model. These findings supported her hypothesis that the program model appears *before* the domain model.

2.3.2.3 Opportunistic comprehension model

The opportunistic model does not support the supremacy of the top-down or bottom-up models. This model affirms that programmers are opportunistic when they exploit both the bottom-up and top-down, based on their availability, during comprehension (Letovsky, 1987;

Von Mayrhauser and Vans, 1995, 1996; von Mayrhauser and Vans, 1997). Proponents of this model state that, on the one hand, programmers use the bottom-up approach for comprehension of a new code when in an unfamiliar domain. However, they switch to a top-down approach when exposed to a program with many lines of code. In addition, a programmer with vast domain knowledge uses the top-down approach more frequently than a programmer with less domain knowledge and would thus spend more time on that stage. The proponents therefore proposed the top-down model, program model and situation model as the models of program comprehension, while stressing the importance of the knowledge base.

Subsequent empirical work (Burkhardt, Détienne, & Wiedenbeck, 2002; Corritore and Wiedenbeck, 2001) has been done using the opportunistic model. The researchers studied the comprehensions differences in C and C++ in experts. They reported that the pattern of comprehension viewed in C++ experts changed over time when compared to the C experts. Subsequent work carried out by Corritore and Wiedenbeck (2001) on novices revealed that their mental representation of the program text was concrete while advanced novices formed their representations abstractly. Although the previously discussed models offer useful guidance as to the study of program comprehension, they do not fully support any one strategy for program comprehension, rather *any* of the three models discussed are used by programmers as the need arises.

2.3.2.4 Block model

This research study used the Block model which is based on both the text and program comprehension theory (Schulte et al., 2010). This model is informed by the bottom-up model of Pennington (see section 2.3.2.2). This model was designed by Schulte (2008). The underlying reasons for building this model was based on his observation that: a) program comprehension is rarely used in the educational setting and prior work mostly focused on professional programmers instead of novices or students, b) there is little connection between writing and reading ability, c) computer science educators strongly focus on the construction of introductory programming with the goal of making students write programs which might lead them away from considerations of reading and d) when questioned, students only answer based on their experience of difficulties. The model was consequently designed with the aim of teaching introductory programming and describing the core aspects of program understanding. The Block model is represented in Table 2.2 and adapted for use in this study.

Table 2.2: The Block model

Macrostructure	Understanding the overall structure of the program text.	Understanding the algorithm of the program.	Understanding the goal/purpose of the program (in its contexts).
Relations	References between blocks, e.g. method calls, object creation, accessing data and so on.	Sequence of method calls.	Understanding how subgoals related to goals, how function is achieved by subfunctions.
Blocks	Regions of Interest (ROI) as a sequence of statements.	Function of a block may be a subgoal.	
Atoms	Language elements.	Operation of a statement.	The function of a statement. Goal only understandable in context.
	Text surface	**Program execution** (data flow and control flow)	**Functions** (as means or as purpose), goals of the program.
Duality	***Structure***		***Function***

Block model emphasises the *reading* of programs, instead of *writing*, and describes the core aspect of program understanding. The column of the blocks represents a dimension of program comprehension while the row depicts levels of comprehension. The model's comprehension follows a bottom-up approach. Comprehension starts by reading the program (text) using the bottom-up process. New information is sequentially included to the reader's internal mental model from "atom" to "blocks" to the "relations" between blocks towards recognising the "macrostructure" which constitutes a holistic aspect of the block. The benefit of *this* model to the teaching of computer science education over the other models is that the other models are quite complex, and no accurate description of comprehension sequences exists. In the Block model, different sets of blocks can be arranged in different order. It is simple in the sense that it helps reduce the variety of known types to the information needed in the mental models. For understanding to occur, appropriate information must be extracted based on inferences. Knowledge types needed for understanding the program text are: the text surface, program execution and function (see Table 2.2 above). *Text surface* represents external representation or actual code the programmer or reader reads for understanding the program text. The text surface is supplemented by the *program execution* which comprises the control and data flow of the program text. Both the text surface *and* program execution then form the mental representation of the *structure*. *The function* relates to the goals of the program which are constructed through inferences and additional domain knowledge.

From the previous description of different PC models, it might be difficult for novices to build mental representations since they do not possess a rich knowledge base that can be used to link the new program text. In addition, it takes about 10 years for a novice to become an expert (Robins et al., 2003). Nevertheless, insightful computer science (CS) educators suggest that novices can obtain a basic level of program comprehension in the space of a year of consolidation through CS1 and CS2 (Schulte, et al 2010). It is speculated that the educational Block model of Schulte might prove interesting in solving the deficiencies inherent to previous models. Up to date, only a few researchers have tested the Block model.

Schulte (2008) tested his model by conducting a qualitative study (comparative and observational) on third year university pre-service teachers. Students were divided into two groups (control and experimental). They had to plan and analyse lessons on Bubble sort algorithm and were then asked to develop a concept for the topic. The findings revealed that only the *control group* identified the possibility of teachers neglecting motivation and the teaching of topics in a wider context if too much focus is placed on the Block model. For further research, Schulte stated that it is possible to research whether the Block model could be used for skill description and identifying competency levels of learners in conjunction with the SOLO taxonomy since his description resembles taxonomic levels. He warned that the SOLO taxonomy should not, however, be confused with possible learning sequences.

Whalley and Kasto (2013) added to the existing literature and tested Schulte's Block model, evaluating its effectiveness for measuring code comprehension exam questions in a first semester programming examination alongside the SOLO and Bloom's taxonomies. They ascertained that the Block model had some of the inherent problems found in the two taxonomies, for example the categorisation of competence verbs. They, therefore, concluded that the Block model still provided a better way of describing novice programming code comprehension tasks than previous models.

The next session discusses the teaching of programming in Nigeria. Literature relating to past research works on procedural (QBASIC) programming in colleges of education in Nigeria will also be reviewed. This section is included to identify gaps and situate the current study within the context.

2.4 THE TEACHING OF PROGRAMMING IN NIGERIA

The role which information and communication technology (ICT) plays in achieving national technological development goals in the 21st century cannot be overemphasised (Agbetuyi and Oluwatayo, 2012). Worldwide the education domain is being influenced by these developments, and Nigeria is no exception. Computer education, as well as its integration into the Nigerian educational system, is emphasised in the Nigeria National Computer Policy (Federal Ministry of Education, 1998). The first objective of this policy was to educate the people of Nigeria to appreciate the impact of information and communication technology (ICT) on society, thus highlighting ICT's effective utilisation and the underlying technological know-how which creates, supervises and conveys information. The second objective was ensuring that Nigerians become familiar with the operation and programming of computers and the building up of software packages. This process included a fostering of understanding as to the formation and operation of computers as well as their history (which the research study focused upon) as well as heightening an awareness as to the value, societal and psychological impact of the computer. In a bid to achieve these objectives, and compete with other developed countries, the Nigerian government introduced computer science disciplines into schools, from primary to tertiary level (Federal Ministry of Education, 2004). An essential component of computer science is "programming", which every student studying computer science must learn and pass because it is a key requirement for graduation. Programming is introduced to students from the secondary (grade 7 to 9) to tertiary levels of education in Nigeria. Of importance to *this* study is the teaching of programming in all tertiary institutions, and specifically colleges of education. The following section therefore discusses the programming curriculum at tertiary institutions in Nigeria.

2.4.1 The teaching of procedural programming in Nigerian colleges of education

Colleges of education in Nigeria are established with the purpose of training and preparing students to become *primary* and *junior secondary* teachers (Aina, 2015; Iji, 2005). This category of student is referred to as *pre-service teachers* (Ololube, 2007). The adequate preparation of pre-service teachers studying computer science and programming related courses is sine-qua-non to technology innovation at all tiers of education in Nigeria. To adequately prepare pre-service teachers, certain structures were put into place for teaching programming. The National Commission for Colleges of Education (NCCE, 2008, 2012), the organisation in charge of the Nigeria Certificate in Education (NCE), and its curriculum developers, designed curricula for the teaching of computer science and programming courses in Nigeria. The curriculum is meant to prepare pre-service teachers in all the 85 federal, state and private colleges of education in Nigeria. Table 2.3 supplies a breakdown

of the curriculum stipulated programming courses offered at all the colleges of education in Nigeria. Other computer science courses are not included in the table since they do not form part of this study. All courses, however, complement each other and aim to educate the individual student to become a graduate and professional teacher in the field of study.

Table 2.3: Programming curriculum in Nigerian colleges of education

Course code and level	Course title	Unit	Status
CSC 112 (100 level 1st semester)	QBASIC Programming Language	2	C
CSC 216 (200 level 1st semester)	C programming language	2	C
CSC 221 (200 level 2nd semester)	PASCAL programming Language	2	C
CSC 321 (300 level 2nd semester)	Advance Level programming language (JAVA, C++, VBASIC, VCOBOL	2	C

Source: NCCE (2012)

CSC: computer science, C: compulsory, Unit: the status of each programming course

The first three programming courses which are introduced in the first and second year (100 and 200 levels) are procedural, with QBASIC programming being the introductory programming course. Object-oriented programming with any of JAVA, C++, VBASIC and VCOBOL is introduced during the last semester of the three year program. The objectives of the three year program, as highlighted in the NCCE (2012) curriculum, state that students must be able to:

- Teach computer studies at primary and secondary levels.
- Write computer programs and process data with maximum speed and accuracy.
- Demonstrate a high level of competence in preparation for further studies in computer science education.
- Motivate pupils' interest in the study of computers by appropriately using information and communication technology (ICT) teaching and learning strategies.
- Apply the use of the computer as an aid in daily life activities.

Pedagogic strategies for the teaching of programming at colleges of education, as stipulated by the NCCE (2012) curriculum, are based on the selection of one of the methods of: discussion, lecture, practical demonstration, tutorials, supervised projects, students' guided practice, problem solving/enquiry, excursion to computer firms, seminar and using

computer-assisted instruction (CAI) by exposing students to five hours of practical per week. There is, therefore, a call for the effective application of the different pedagogic strategies towards the development of programming skills in students (NCCE, 2012). The programming assessment comprises a mid-semester test (15%), laboratory assignments (25%) and summative examination (60%) (NCCE, 2008, 2012). One can argue that the overall knowledge gained by these students (pre-service teachers) will provide them with the content knowledge needed to teach programming which will, in the long run, help in achieving the national goals of technological development.

Surprisingly, many students still fail programming, one of the basic skills in computer science (Jegede, 2009). In support of this statement, Olelewe and Agomuo (2016) and Olelewe (2016) stressed that the failure rate and lack of students' interest in learning programming (QBASIC) at colleges has had a negative effect on the achievement of the goals of teaching and learning computer programming. In effect, if pre-service teachers are unable to grasp programming skills, it would be very difficult for them to teach said skills to young learners at primary and secondary school levels which, in the long run, will most certainly affect achievement of the goals. Therefore, the researcher posits that studying *how* pre-service teachers learn to program from their first year at college is an important step in ascertaining the areas of the computer science curriculum where pre-service teachers can be prepared to acquire the necessary programming skills needed to contribute to the development of the Nigerian economy. This is in support of Olelewe and Fakorede (2009) who stressed that the preparation of first year programming students to learn QBASIC is vital in the development process as it helps in achieving curriculum goals and provides manpower needed to sustain the Nigerian economy. Since the overall outcomes of an educational curriculum is a determinant factor for measuring the success, or failure, of an educational system (Olelewe and Agomuo, 2016), research into first year college students' programming (with special emphasis on QBASIC programming) is necessary.

In developed countries extensive research studies have been done on procedural programming languages such as QBASIC, COBOL and PASCAL with respect to first year students. Recent studies in these counties have indicated a move towards object-oriented programming. In Nigeria, limited research has been done on first year students in colleges of education. Olelewe and Agomuo (2016) studied the effectiveness of the B-learning model and face to face (F2F) learning environments on student achievement in QBASIC programming. They used a quasi-experimental design using two groups, which comprised of 148 first year college of education students from Enugu state, Nigeria. The QBASIC programming achievement test (QBPAT) was used to test students' achievements in the

two groups, and the result showed the B-learning model to be more effective in improving students' achievement than the traditional face to face method.

Owolabi et al. (2014) studied mathematics ability and achievement, programming anxiety, age and gender as determinants to first year students' achievement in QBasic programming. Using a Pearson Product Moment Correlation and Multiple regression analysis, they discovered a positive and significant correlation between mathematics and QBASIC programming achievement. They suggested that programming teachers should encourage first year students to improve their mathematics ability which will then improve their performances in QBASIC programming. An earlier study by Adenubi, Lateef, and Bukonla (2007) developed a user-friendly graphical user interface (GUI) software called Simple Pedagogical Programming Language interpreter (SiPPL) for the teaching of programming at junior secondary schools. Its design involves the use of Java programming, based on the NetBeans-IDE, an integrated development software. The core idea was to develop a programming software that would bolster young learners' interest in programming. However, no report detailing the effectiveness of the SiPPL for introducing QBASIC programming was found. Table 2.4 presents research studies on QBASIC programming in Nigerian colleges of education. Due to the deficiencies of past studies in Nigerian colleges of education, the research process will use *Scratch programming as a precursor to the learning of procedural programming* in a Nigerian college of education and explain its impact.

Table 2.4: Studies on procedural programming in QBASIC programming in Nigeria

Author	Setting	Methodology	Intervention	VPE
Olelewe and Agomuo (2016)	College	Quantitative	B-learning and F2F	Nil
Owolabi et al. (2014)	College	Quantitative	No intervention	Nil
Adenubi et al. (2007)	Secondary school	Nil	Not tested	SIPPL

2.5 CHAPTER SUMMARY

This chapter reviewed the literature on programming and program comprehension. The chapter commenced with a discussion on programming education (*q.v.* section 2.2), which unfolded with explanations regarding the differences between *expert* and *novice* programmers (*q.v*. 2.2.1), programming paradigms for introducing programming (*q.v*. 2.2.2) and the difficulties experienced by novices in procedural programming (*q.v*. 2.2.2.1). Following this, section 2.2.3 discussed the use of a visual programming environment for introducing programming to novices with emphasis on Scratch programming (*q.v.* section 2.2.3.1). The last section (*q.v.* section 2.4) discussed how programming teaching is done in Nigeria (*q.v.* section 2.4) while focusing on colleges of education and pinpointing the deficiencies found in literatures on QBASIC programming in Nigerian colleges of education (*q.v.* section 2.4.1).

CHAPTER 3: PERSPECTIVES ON LEARNING AND LEARNING THEORIES

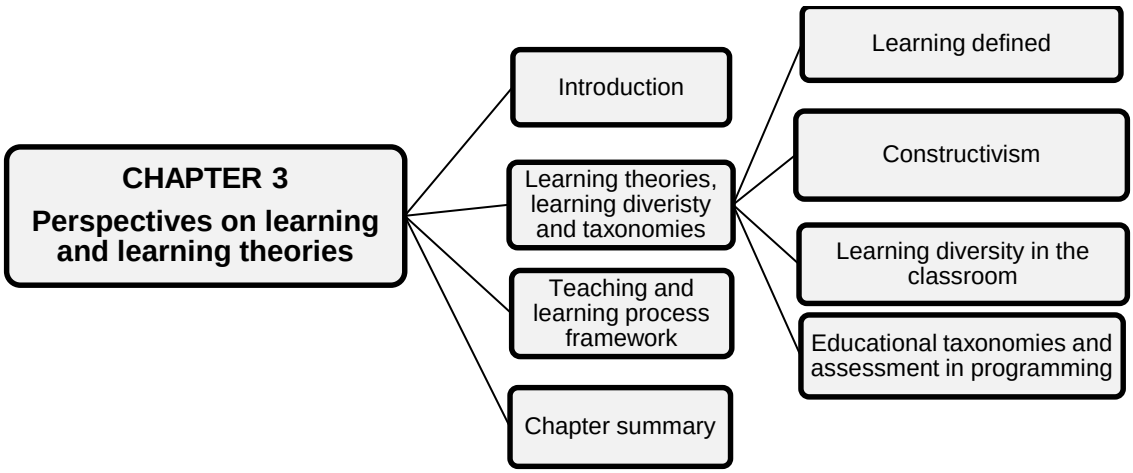


Figure 3.1: Schematic representation of Chapter 3

3.1 INTRODUCTION

Learning, as a phenomenon, and *how* to influence it has been widely discussed. This chapter focuses on those learning perspectives and learning theories needed for the teaching of programming. Learning and learning theories which support the research study, including Kolb and Herrmann's Brain Dominance Instrument (HBDI), are discussed (see section 3.2). In addition, educational taxonomies for teaching, learning and assessment as well as the use of constructive alignment in developing courses for teaching and learning are touched upon. In addition, empirical studies reporting on their use in programming are also reviewed. Section 3.3 discusses the conceptual framework, based on the salient points from the reviewed literature, designed for the study. The chapter concludes with a summary (section 3.4).

3.2 LEARNING THEORIES, LEARNING DIVERSITY AND TAXONOMIES

This section describes learning, as conceived by different learning theorists (see section 3.2.1). A more detailed discussion in section 3.2.2 focuses on constructivism, the learning theory guiding this research study. In section 3.2.3, learning diversity is further explained as it influences instructional design. The last section describes how to plan learning based on educational taxonomies (section 3.2.4).

3.2.1 Learning defined

People generally view learning as important, but many different views exist regarding the subject. Learning theorists, researchers and practitioners have offered no agreed

consensus on precisely what learning is (Illeris, 2009; Pritchard, 2013; Schunk, 2014). Illeris (2007, p. 3) defines learning as “any process in living organisms that leads to permanent capacity change and which is not solely biological maturation or ageing” whilst Schunk (2014, p. 3) defines learning as “an enduring change in behaviour, or incapacity to behave in a given fashion, which results from practice or other forms of experience”. Ormrod (1999) defines learning as *a change in performance through conditions of activity, practice and experience*. Even though theorists do not hold to one agreed definition of learning, similarities, pertaining to changes in behaviour resulting from experience, are still found.

Jonassen (2009) noted that it is not possible to explain the way learning should occur based on a single theory. There are many different theories as to the nature of learning in the field of psychology and each of these has contributed valuable insight into the way teaching and learning should occur in the classroom. The three broad learning theories are behaviourism, cognitivism and constructivism. According to *behaviourism*, learning is achieved when there is a permanent change in behaviour as a response to an external stimulus (De Corte, 2010). This definition excludes learning as the formation of mental operations. *Cognitivism* is a shift from behaviourism and deals with the study of internal mental processes which underpin behind the knowledge structures acquired by an individual (De Corte, 2010; Jonassen, 1991; Pritchard, 2013). In cognitivism, learners are viewed as information processors. In an effort to unknot the internal mental processes and knowledge structures of learners, cognitive learning theorists improved upon the work of behaviourists, thereby affirming that a learner should be an *active* constructor of information and not a *passive* recipients (De Corte, 2010). Therefore, *constructivism* views learning as the active construction of knowledge through interaction with the environment towards the restructuring of internal mental processes (De Corte, 2010; Pritchard, 2013; Schunk, 2014). In constructivism, the individual’s prior experience becomes important to understanding the meaning of the material (Amineh and Asl, 2015). The constructivist learning theory guides this study. The researcher chose this theory because she holds the view that learning should be socially constructed among learners. Further discussions on constructivism follow.

3.2.2 Constructivism

Constructivism, as a learning theory, is not a new phenomenon (Sigmund and Thomas, 2009) and it means different things to different people. As described by Von Glasersfeld (1998, p. 123) the first constructivist philosopher, Giambattista Vico, stated that “epistemic agents can know nothing, but the cognitive structures they themselves have put together ‘to know’ means to know how to make”. The more recent rebirth of constructivism can be

credited to the work of theorists like Jean Piaget, Lev Vygotsky, Jerome Brunner, John Dewey and Von Glasersfeld (Sigmund and Thomas, 2009). A common understanding gained from these theorists is that constructivism places the learner at the centre of the teaching and learning process, thus involving him/her in the mental construction of knowledge based on experience, as well as building mental representations of experiences as knowledge progresses. There are two strands of constructivism prevalent in education: *cognitive* constructivism and *social* constructivism. Figure 3.2 gives a schematic representation of the types of constructivism, along with their corresponding concepts. The figure is unpacked and discussed in detail in the following subsections.

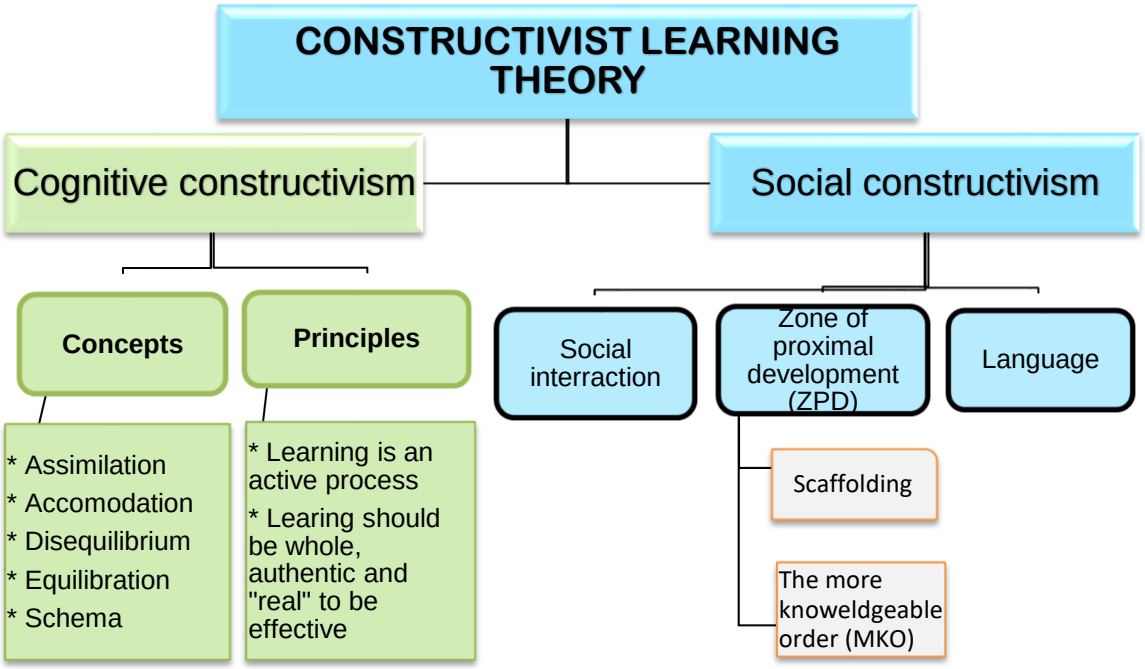


Figure 3.2: Synthesis of constructivist learning theory

3.2.2.1 Cognitive constructivism

In accordance with the Piagetian theory of *cognitive constructivism,* learning is seen as an active mental construction of information (Pritchard, 2013). Piaget holds forth that two principles, adaptation and organisation, guide the learner's intellectual growth (Piaget, 1964). He noted that an individual learner must adjust to both physical and mental stimuli in order to survive in an environment. Pritchard (2013) and Schunk (2014) described Piaget's assimilation and accommodation as the two basic processes which guide adjustment to environmental influences. They noted that the process, whereby new knowledge is added into existing mental structures, is *assimilation* and that the altering of mental structures to cope with a new experience that has contradicted the mental model is *accommodation.* In a

situation where the external reality does not match the individual's mental stimuli, *internal conflict* or *disequilibria* exists. The stable stage where there is no longer any conflict between new and existing knowledge is *equilibration*. Therefore, for learning to take place, equilibration is needed to bring about a balance between assimilation and accommodation. Within the constructivist theory, different interpretations exist regarding the basic ideas of knowledge construction and understanding schema. *Schema* is a cognitive structure that helps in the meaningful organisation of large amounts of information received from the environment (McVee, Dunsmore, & Gavelek, 2005). Since learning is based on a student's prior knowledge, this schema is the previous knowledge (Bhattacharjee, 2015) that assists the student with gaining reference, understanding or prediction. Schema activation is frequently used in classroom situations (Pritchard, 2013). The interrelationship between the Piagetian concepts of cognitive constructivism, as discussed above, is illustrated in Figure 3.3.

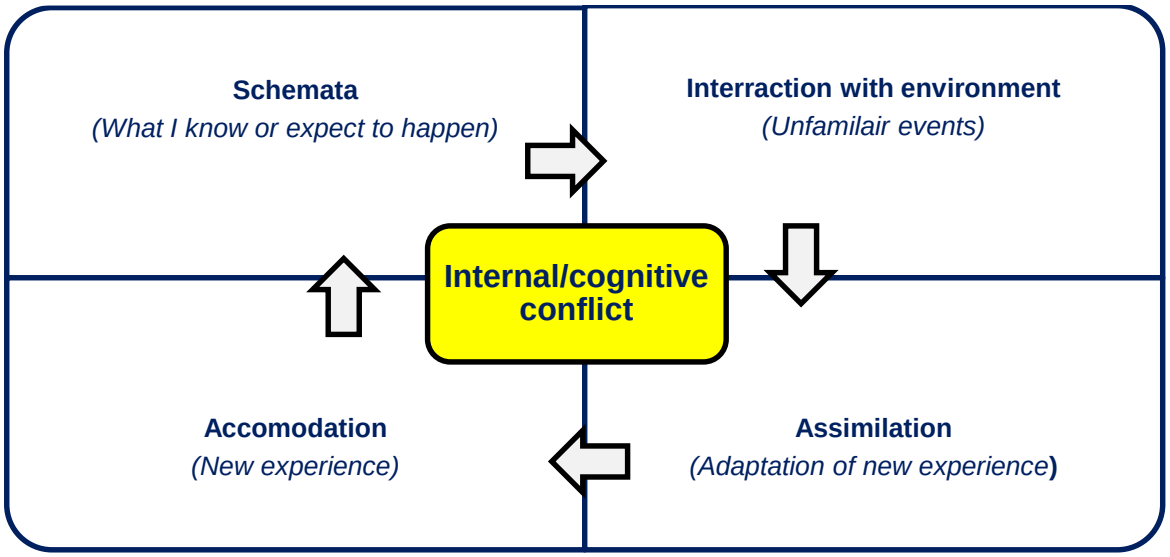


Figure 3.3: Graphical representation of Piaget's theory of cognitive constructivism

3.2.2.2 Social Constructivism

Social constructivism is another strand of constructivism proposed by Lev Vygotsky (1962/1986). He theorised that the social environment facilitates development and learning (Schunk, 2014). He stressed that processes of interpersonal interaction, as well as cultural-historical and individual factors, are crucial to human development. This contradicts Piaget's belief that learning is mainly an internal and individual cognitive process. The underlying assumptions of Vygotsky's theory (Vygotsky, 1978) is that: *social interactions* are essential to the construction of knowledge between individuals. *Self-regulation* develops as individuals internalise events which emerge during mental operations informed by social

interactions; *human development* thus happens through the cultural transmission of tools. *Language* is a vital tool and interactions in the *zone of proximal development (ZPD)* promote cognitive development.

Constructivist theories can be further categorised into three unique perspectives: exogenous, endogenous and dialectical constructivism (Schunk, 2014). Each of these has implications for instructional design (ibid). Vygotsky's perspectives about learning are dialectical because they emphasise interaction between individuals and their immediate environment (Schunk, 2014). Learning designed, based on constructivism, stresses meaningful learning (Gasaymeh, AlJa'afreh, Al-Dmour, & Alrub, 2016). Therefore, learning informed by constructivism portrays the learner as an active constructor of meaning. This concept has become popular in education reform, but it has not been embraced by all teachers (Schunk, 2014). The principles of constructivism pertaining to learning, as explained by different authors, are presented in Table 3.1.

Table 3.1: Implications of constructivism for learning

Ideas in constructivist learning	Meaning	References
Learning is active	Learning is the active construction of knowledge and learners construct their own meaning of knowledge and skills. Therefore, meaningful learning is the active creation of knowledge from personal experience.	Fosnot and Perry (2013); Ormond (1999)
Learning develops	Learning is development and requires that learners invent, do self-organization, raise their own plausible questions, test and validate hypotheses.	Fosnot and Perry (2013)
Learning in memory is remembered through construction	The theory has not dealt unequivocally with memory, but contends that learners remember information based on their meaningful constructions of learning.	Schunk (2008)
Situated learning	Learning is situated and hence teaching must be situated in that context.	Fosnot and Perry (2013)
Learners own their error	Learners' errors are their own conceptions and thus should not be avoided by the teacher.	Fosnot and Perry (2013)
Reflection through interaction	Reflections that are based on abstractions are the driving force of learning and interaction within a social community improves further thinking. The learning environment must be structured to promote such interactions.	Fosnot and Perry (2013); Amineh and Asl (2015)

Scaffolding	Learning is well structured by creating a supportive environment where learners construct their understandings and teachers give scaffolding that will assist learners to maximize their ZPD.	Bhattacharjee (2015)
Differentiation of tasks	Multidimensional classrooms encourage greater diversity in learners' performances. Differentiation of tasks, grouping patterns, student autonomy and salience of performance evaluations should be infused in the classroom.	Taylor and MacKenney (2008)
Assessment is authentic	Assessment is not done at the end of the teaching exercise, but incessantly during the teaching process. Assessment should also be authentic and must foster deep structure learning.	Amineh and Asl (2015)

3.2.2.3 Constructionism

The word *constructionism* originated from Piaget's constructivism. (Papert and Harel, 1991, p. 6) defined constructionism as "learning by making". Papert asserted that constructio*n*ism ('n' in italics) with its "N" idea shares a worldview with the "V" idea in constructi*v*ism ('v' in italics) as it refers to "building knowledge structures regardless of the circumstances of learning" (Papert and Harel, 1991, p. 1). He further noted that this form of learning happens in a context where the learner is consciously engaged in the construction and modification of digital artifacts. Constructivism and constructionism share similar views about learning. However, there is a distinction. While Piaget's constructivism is interested in the learner's learning at each stage of development and his/her own construction of learning, Papert's constructionism focuses on the art of learning through making, or designing, a sharable object (Girvan, Tangney, & Savage, 2013; Kafai and Resnick, 1996). To this end, Papert's constructionism emphasises learners' engagement in a conversation brought about by the production of an artefact comparable to the way in which conversations bring about self-directed learning and the construction of fresh knowledge (Ackermann, 2001). Papert's constructionism not only focuses on the construction of knowledge, but also on a way of learning whereby a student builds an object, makes changes to the object by moving back and forth towards deconstruction and reconstruction of knowledge (Kafai and Resnick, 1996; Kynigos, 2015). In Papert's view, presenting our ideas or feelings is paramount to learning (Ackerman, 2001).

Different tools have been applied in both formal and informal learning environments to represent the idea of Papert's constructionism. These tools, including Lego Mindstorms (Resnick, Ocko, & Papert, 1988) and Scratch programming (Resnick et al., 2009), have

been designed to aid students' understanding of programming problems through self-construction (Li, Cheng, & Liu, 2013). Constructionism is relevant to this study because Scratch programming requires students to design programming blocks, both individually and in groups, and to collaboratively share ideas.

3.2.2.4 Constructivist teaching strategies for facilitating programming instruction

If students are to learn to program, they need to experience programming (Luo, 2005). Due to the problem-solving nature of programming, it is important to teach it using constructivist strategies. Numerous studies have reported on the effectiveness of different constructivist strategies for teaching programming. This section describes the constructivist strategies used in this study to facilitate programming instruction. These strategies are: zone of proximal development with a focus on instructional scaffolding, collaborative learning with emphasis on pair programming and cooperative learning.

- **Zone of proximal development**

The zone of proximal development (ZPD), as defined by Schunk (2014) and (Verenikina, 2008), refers to the distance between a problem which a student can independently solve and that which he/she cannot solve under the guidance of the teacher. They argued that since supporting a student's active participation in learning and his/her becoming self-regulated are the central part of ZPD, the teacher should work *with* the student on challenging tasks. Thus, in the ZPD, teacher and learner must work together for cognitive change to occur (Schunk, 2008, 2014). A student can thus restructure his/her ideas when information is received from a teacher or peers. This *restructuring* of knowledge is relevant to the learning of programming concepts, because it will help students correct their conception/s and encourage them to receive new information to the existing schemata. Figure 3.4 gives a representation of the ZPD level of comptence in a learner. It depicts the tasks which the learner cannot yet complete, even with assistance, and the tasks the learner can complete with assistance at the ZPD. The learner attains competence after reaching the ZPD. Vygotsky did not detail the practical application of ZPD in his work and this poses many questions on how precisely it should be used. However, common applications of Vygotsky's theory is: the concept of *instructional scaffolding*, *reciprocal teaching* and *peer collaboration* (Schunk, 2014). Relevant to this study is instructional scaffolding.

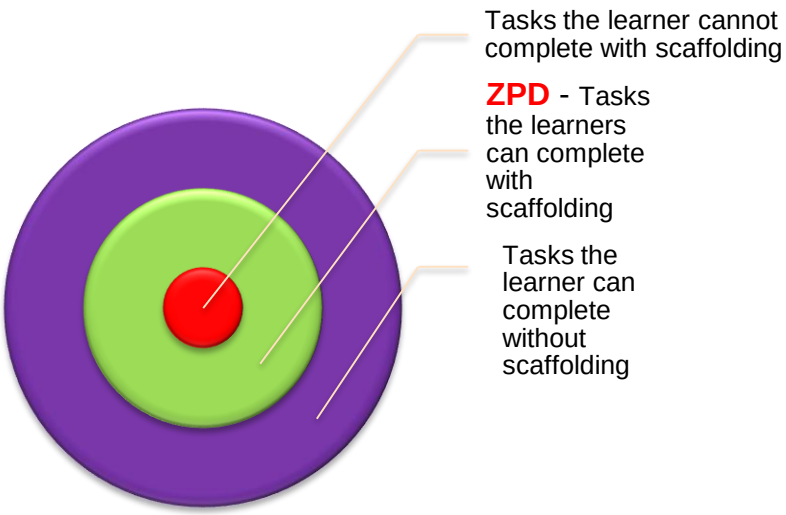


Figure 3.4: Zone of proximal development

- **Instructional scaffolding**

Instructional scaffolding, though not a formal part of Vygotsky's theory, was introduced in 1976 by Wood, Werner and Ross (Schunk, 2014). Instructional scaffolding is learning support afforded during a learning process with the intention of influencing learning tasks that are beyond the learner's competence, so that learning goals are achieved quickly (Schunk, 2014; Tiantong and Teemuangsai, 2013; Verenikina, 2008). During teaching, the teacher might initially do most of the work, but later tasks are shared with the learner. The teacher gradually withdraws the scaffold as the learner becomes competent. This scaffolding process is therefore divided into three sections, as explained by Van de Pol, Volman, and Beishuizen (2010). They are:

a) contingency which refers to introducing support based on the student's need,

b) fading support which must be gradual and

c) transfer of responsibility of a learning task to the student.

For such scaffolding to be effective and to work in tandem with the ZPD of Vygotsky, Verenikina (2008) opines that teaching and learning should be structured so that a) learners carry out the task which they cannot do on their own, b) the teacher brings the learners to a state of competence where they can do the task on their own and c) there is proof showing that the learners have reached the ZPD as a result of scaffolding.

Scaffolding is not without criticisms, Verenikina (2008) reporting Lave and Wenger (1991) stated that its literal explanation views teaching as a one-way communication as opposed to the teacher-learner collaboration specified in ZPD. As such, if not applied properly in

teaching, it might lead to a traditional way of teaching. Schunk (2014) stressed that scaffolding is not really related to the ZPD of Vygotsky's theory. There has been little debate on the adequacy of ZPD, but Verenikina (2008) stressed that scaffolding should not be abandoned, especially when used during the learning of a new concept. Instructional scaffolding, therefore, has five different functions: a) providing support, b) working as a tool, c) extending the learner's choice, d) allowing achievement of the task which seems impossible and e) using selectively as needed. Several research studies (Angelakopoulou, 2016; Panselinas and Komis, 2009; Raes, Schellens, De Wever, & Vanderhoven, 2012; van de Pol, Volman, & Beishuizen, 2012) have reported on the effectiveness of scaffolding. Nevertheless, its application in classroom teaching is rare (van de Pol, Volman, Oort, & Beishuizen, 2014). This is also true regarding its use in computer programming classrooms (Cheng, 2010). Reasons for this could be ascribed to the complexity of programming and the nature of classroom situations involving many students that do not allow for personalised scaffolding. Hence, software tools are popularly used as scaffolding techniques in programming classrooms (Cheng, 2010).

Awbi, Whalley, and Philpott (2015) researched scaffolding, the zone of proximal development and novice programmers. Using a narrative analysis of retrospective think-aloud interviews, they found that scaffolding can control learning progression and thereby extend learners' ZPD. Stachel et al. (2013) investigated the effects of scaffolding tools on the cognitive loads of CS students exposed to Visual Basic. In phase 1 students received scaffolding while in phase 2 they did not. Students' evaluations of their cognitive load and course grade were collected and compared in the final phases of the research. Results showed that phase 1 students experienced lower cognitive load than phase 2 students who received scaffolding from online, small groups and face-to-face interactions. Another study (Mbogo, 2015) investigated the impact of mobile phones in scaffolding Java programming to novice learners. Based on a six-level framework using 182 novice programmers, data were collected using questionnaires, computer logs, media and videos. Findings showed that students attempted and completed programming problems and identified syntactic errors earlier during program running. Students also found scaffolding useful for the teaching of programming.

- **Collaborative learning**

Collaborative learning is a mutual engagement of participants in a matched effort to solve a common problem together or create a product which could not be completed individually (Herrington and Herrington, 2006; Serrano-Cámara, Paredes-Velasco, Alcover, & Velazquez-Iturbide, 2014). Collaboration entails the whole learning process in which

students are responsible for their own as well as others' learning by assisting each other to understand. In effect, it diverts the responsibility of learning to the student. Thus, teachers must understand learners' learning styles and conceptions of learning (Dooly, 2008). Many studies (McKinney and Denton, 2006a; Williams, Kessler, Cunningham, & Jeffries, 2000) have established the benefits of collaborative learning as a way of brainstorming and sharing resources, monitoring of problem-solving, encouraging deep learning and high retention rates and facilitating higher achievement rates, amongst others. Studies, including that by McKinney and Denton (2006a), have used collaborative learning to teach introductory programming. They exposed students to team-based pair programming and their results show that students displayed higher interest, higher retention and higher academic performance. A good example of collaborative learning in the programming classroom is the use of pair programming.

- **Pair programming**

Pair programming (PP) is a collaborative form of doing programming whereby two people work side by side at the keyboard, based on prearranged rules (Denner, Werner, Campe, & Ortiz, 2014; Watkins and Watkins, 2009). One of the programmers is the 'driver' who operates the keyboard and focuses on language syntax and control structures details. The other programmer is the 'navigator' who sits next to the driver and offers suggestions relating to the overall program design and integration (Braught, Wahls, & Eby, 2011). They change roles on a regular basis and are both responsible for all aspects of the program. The two goals of PP, as stated by Werner, Hanks, and McDowell (2004), are that the pair 'code' and also 'own' the code collaboratively. A number of studies have reported on the pedagogical benefits of PP in introductory programming. Braught et al. (2011) found that PP improved the individual student's programming skills and that it also enhanced his/her confidence when programming. He also found that students are more likely to complete the course. Also, Simon and Hanks (2008a), using a qualitative study, reported that PP helped students to share ideas and to become more confident in solo programming when left alone. Other studies, such as that by Tafliovich, Campbell, and Petersen (2013) and Nuraminah and Shukor (2008), studied PP from the student's perspective in CS1, claiming promising results. A previous study, conducted by Cao and Xu (2005), reported on the benefits of PP but their findings also revealed that the pairing of incompatible partners could reduce satisfaction in students. The findings of Begel and Nagappan (2008) confirmed the study of Cao and Xu (2005) claiming that PP led to personality conflicts amongst experienced programmers who work in the industry.

- **Cooperative learning**

Cooperative learning (CL) is a well-structured form of learning which is centred on problem-solving and carried out within a group to achieve a common product, under the guidance of a teacher (Dooly, 2008; Millis, 2010). The objective of CL is to develop collaborative abilities by working with others (Schunk, 2014). To initiate a CL, five elements (namely positive interdependence, individual accountability, group processing, social and interpersonal skills and face to face interaction) need to be considered (Johnson and Johnson, 2009, pp. 366-370). *Positive interdependence* occurs when tasks, that cannot be completed by an individual, are divided amongst members of a group. Students share different components of the problem and merge their individual work to form a product (Millis, 2010; Schunk, 2014). *Individual accountability* is when each group member receives grades based on his/her individual contribution to the task. In *group processing*, groups are guided by the teacher to ensure that the group is productive and that it exhibits the required behavior, such as discussing and determining the required action for completing a task (Johnson and Johnson, 2009). Effective CL also involves *social* and *interpersonal skills* where team members practice leadership skills, decision making skills, manage conflicts, ensure trust building and employ effective communication. The last cooperative principle is face to face interaction where group members meet to give feedback to the group and to assist other groups to complete their task. There are different types of cooperative learning methods that can be used for teaching and in which all the principles of CL, as discussed above, must be implemented. Common examples used by previous researchers for the teaching of programming are the think-pair-share (Kothiyal, Majumdar, Murthy, & Iyer, 2013) and jigsaw (Kordaki, 2012).

Beck and Chizhik (2013) implemented CL instructional methods for a CS1 course which was designed to focus on some aspects of programming. Results showed that CL was effective in groups and that there was an increase in programming performance. Previous research work by Falkner and Palmer (2009) corroborates the above research and revealed that students were motivated to learn programming, showcased a changed attitude towards lectures and practical classes and developed better understanding of concepts through the cooperative method of teaching. In another study, Mentz, Van der Walt, and Goosen (2008) conducted a two year study amongst student teachers learning Delphi programming at a university in South Africa. Cooperative learning principles were incorporated into pair programming and both quantitative, and qualitative, findings revealed that students learned to program with the help of cooperative learning principles. It was also found that cooperative learning promotes engagement in students during classroom programming activities (Gebre, Saroyan, & Bracewell, 2014; Myller, Bednarik, Sutinen, & Ben-Ari, 2009).

3.2.3 Learning diversity in the classroom

Recent increases in student enrolment in higher education has resulted in challenges controlling the student population and a growing diversity of students. Teachers have to adapt and cater for different preferences in the classroom (Opdecam, Everaert, Van Keer, & Buysschaert, 2014). The way in which each student approaches learning is quite unique to him/her and reflects his/her different observations, interactions, behaviour and even preferences. This method of learning is termed 'preferred learning style' (Pritchard, 2013). Learning style is defined as the idea that individuals do learn differently and that teaching should be based on the type of instruction most suitable to them (Pashler, McDaniel, Rohrer, & Bjork, 2008). The study of students' learning styles is well document in literature, with different names used by different authors. It is commonly termed, and discussed, as learning styles, thinking preferences and approaches to students learning. Interestingly, students come from different social classes, backgrounds and ethnic groups and their individual qualities and ideas of self-worth should be celebrated. Entwistle, McCune, and Hounsell (2002) posited that teachers' understanding of students' learning preferences can assist them in selecting the correct strategies and structuring an academic environment in such a way that students' needs are well catered for. Therefore, for successful learning to take place and for teaching to be more effective, an awareness of learners' individual preferred learning styles need to be fostered (Pritchard, 2013). These assertions suggest that there may be a connection between teaching strategies and students' learning styles. Different studies have reported on the relationship between learning approaches, or styles, and teaching pedagogy. For example, Tulbure (2011) confirmed in his study that students perform better when they are faced with teaching strategies that meet their learning needs. Reports from the study of Hsieh, Jang, Hwang, and Chen (2011) and (Hwang, Sung, Hung, Huang, & Tsai, 2012a) confirm these findings.

Coffield, Moseley, Hall, and Ecclestone (2004) identified 70 different learning styles. They then presented an extensive report on 13 of these, along with their different components, and claimed that most of them lack validity. Other researchers in the field (Bishka, 2010; Coffield, 2012; Coffield et al., 2004; Cuevas, 2015; Howard-Jones, 2014; Pashler et al., 2008; Scott, 2010) have argued that, despite its wide use in literature, the learning styles research field is still riddled with controversies and criticisms. Some of the issues raised by these researchers are: a) learning styles are theoretically disjointed and conceptually confused; b) the absence of an agreed upon theory and vocabulary on which dichotomies can be based; c) the lack of benefits gained, even when learning is based on a preferred learning style; d) the recent commercialisation of learning styles and the existence of various

models which has reduced credibility. Coffield et al. (2004) identified reliability and validity flaws and a lack of impact on pedagogy in some of the preferred 13 models. Years after the findings of Coffield et al. (2004), other researchers (Choi, Lee, & Kang, 2009; Martin, 2010; Sankey, Birch, & Gardiner, 2012; Willingham, Hughes, & Dobolyi, 2015; Zacharis, 2011) conducted experimental research studies to further test its impact on pedagogy. Their findings could not ascertain a correlation between learning styles descriptors and students' learning. They, therefore, stressed that such instruments should not be regarded as theoretically sound reasons to change practice. However, the Hermann Brain Based Dominance Instrument (HBDI), amongst others, was regarded as promising. Based on the limitations highlighted, Coffield et al. (2004, p. 141) and Hawk and Shah (2007), building on Curry (1987) stressed that "it is unwise to use any one instrument as a true indicator of learning styles….using one measure assumes that measure is more correct than others". Educators were advised to be cautious when using any pedagogical intervention mostly based on a learning styles instrument.

In the light of the afore mentioned, the researcher categorically states that the use of learning styles in this study is *not* done in an effort to label learners *or* to determine the effect of interaction between learning styles and student performance. For this study, l*earning approach* were used to allow for differentiated instruction in the programming classroom. It is certain that students will benefit from the facilitation of a course using diverse forms of instruction (Pashler et al., 2008; Pritchard, 2013; Tulbure, 2011). In this thesis, two instruments (the KLSI and HBDI) are combined to determine students' preferred way of learning. The two learning styles are reviewed in the following subsections.

3.2.3.1 Herrmann Whole Brain Model

Herrmann's work focused on understanding the creativity of the brain (De Boer, Steyn, and Du Toit (2001). In his 1995 research, he established that the brain is divided into four learning modes (upper left, lower left, upper right and lower right) which function as a whole. His work built on the right and left brain theory of Roger Sperry and his colleagues in 1980 and Maclean's triune brain theory formulated in the 1970s (Bawaneh, Abdullah, Saleh, & Yin, 2011a; De Boer, Du Toit, Scheepers, & Bothma, 2013). Herrmann classified learning styles based on his model and stressed that each brain division is associated with a learning approach (Bawaneh, Zain, & Saleh, 2011b). His model used *thinking styles* instead of *learning styles* and quadrants to describe the four categories of thinking styles (Coffield et al., 2004). He termed each of the four learning modes as *quadrants* with a specific cognitive function, specialised language and ways of learning and solving problems (Herrmann,

1995). He noted that every individual possesses a fraction of each of these processes as present in all quadrants of the brain. Therefore, an individual's brain may be less active in these processes when one brain hemisphere is either motivated, or challenged. Figure 3.6 is a graphic representation of the Herrmann whole brain model.

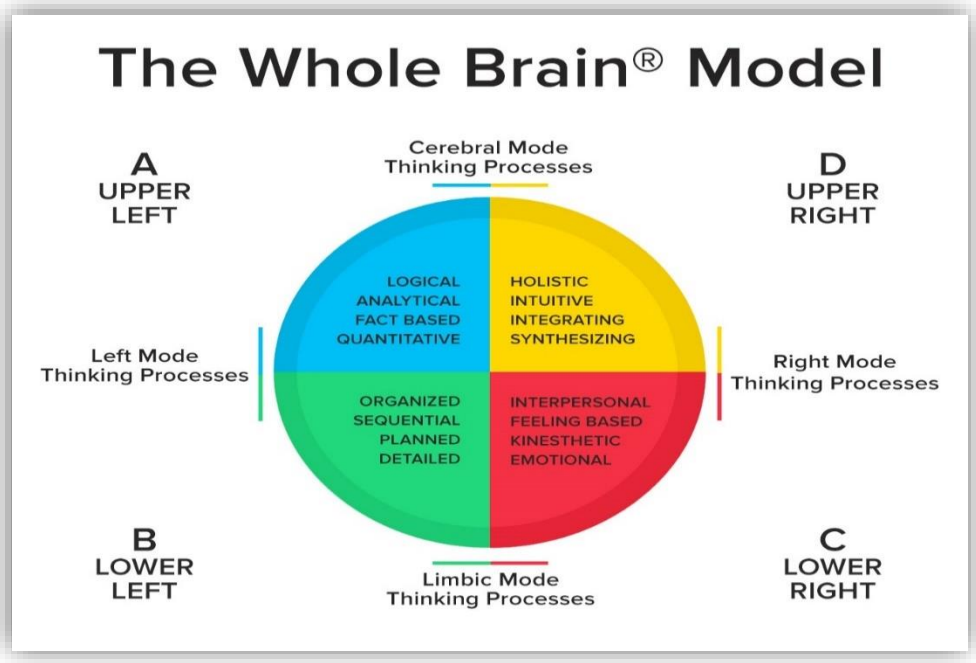


Figure 3.5: Herrmann's four-quadrant Whole Brain model
Source: Herrmann (1996)

From Figure 3.5, Herrmann stated that:

> *Quadrant A (upper left cerebral mode)* indicates that the individual favours logical, analytical, fact-based and quantitative activities.
>
> *Quadrant B (lower left limbic mode)* implies organised, sequential, planned and detailed activities, conservative in nature and striving to keep things as they are.
>
> *Quadrant C (lower right limbic mode)* means the individual engages in interpersonal, emotional, feeling based and kinesthetic activities.
>
> *Quadrant D (upper right cerebral mode)* favours a holistic and conceptual approach to thinking.

De Boer et al. (2001) and Herrmann (1996) noted that the thinking of an individual may be dominant in one preference, but the two hemispheres (left and right brain) contribute to all cognitive processes, albeit in different ways. Similarly, the brain must function in a holistic way if effective learning is to take place. By implication, learning experiences must be

designed to cater for the whole brain so that the material can address the diversity of learners in a learning situation (De Boer et al., 2001).

Herrmann developed a scientifically validated 120 item self-report instrument, the Herrmann Brain Dominance Instrument (HBDI), that can quantify the degree of thinking preferences. The instrument covers different types of preferences and gives performance ratings for: handedness, hobbies, energy levels, work elements and key descriptors, amongst others. Coffield et al. (2004) strongly held that the psychometric properties of the HBDI are sound. The instrument, as a learning tool, should be used in an environment which supports openness and trust and *not* for labelling individuals (Coffield et al., 2004). This study, therefore, follows suit. Herrmann (1996) and De Boer et al. (2013), in like-minded fashion, affirmed that the instrument cannot be used in isolation without aligning it to a whole brain teaching and learning environment. It serves as a basis to connect learners' learning needs with appropriate methods of facilitating learning. The following section discusses whole brain teaching and learning (WBTL).

3.2.3.2 Whole brain teaching and learning

WBTL is a framework for teaching based on the whole brain model. The framework supports the constructivist view of learning and it is well adapted to teaching and learning in the whole brain. It is designed to assist facilitators to plan and design learning outcomes which integrate and cater for each quadrant of the brain (De Boer et al., 2001). Herrmann, therefore, designed a *walk around* model (see Figure 3.6) for designing learning experiences in the four quadrants of the brain. The *walk around* model is based on the expanded whole brain model for teaching and learning (De Boer et al., 2001) and will enable the facilitator to move back and forth in the model. Many years of research work and application enabled the researchers to improve on the earlier *expanded whole brain* and adopt a comprehensive model. Figure 3.6 presents the walk around model in shematic fashion. This model was used for facilitating programming lessons.

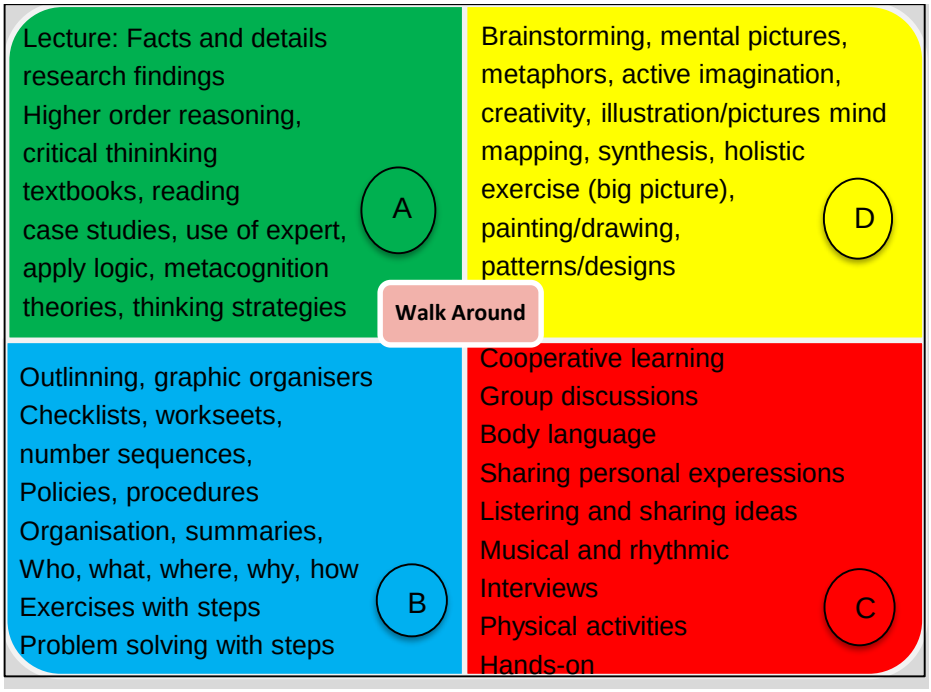


Figure 3.6:The whole brain walk around
Source: Adapted from De Boer et al. (2001)

The Herrmann whole brain model for teaching and learning assists the teacher in communicating with those who think like him/her and with those who think differently. Each of the *four thinking preferences can be infused within a single classroom teaching activity and, as such, equal learning opportunities for different learners can be attained* (Bawaneh et al., 2011b; Qudah, 2009). This can only be effective when learners with different learning preferences are placed in groups of four or six, with the teacher assigning one-quarter of the teaching time to the preferred style and the other three-quarters to the other styles (Bawaneh et al., 2011a). Varieties of creative materials for individual, and group activities, can motivate learners to move out of their comfort zones, thus enabling them to face the considerable challenge of having different mental models working in the same group (Coffield et al., 2004). This neurological process influences the way in which a teacher teaches as the understanding of a learner's brain processes will help the teacher to explore said learner's individual preferences. Teachers thus gain knowledge of their own thinking processes and the implications these hold for teaching and learning.

The effectiveness of the WBTL approach has been studied in different contexts but it has not gained influence in the field of computer science and programming. Bawaneh et al. (2011a) studied Jordanian Grade 8 students' thinking styles in physics, using the whole brain model. Three hundred and fifty-seven students were sampled from 14 classrooms within the BaniKenana school district. Students completed a 60-item Arabic version of the HBDI instrument. Results showed that quadrant C had the highest percentage at 34.5%,

quadrant D 26.9%, quadrant B 19.9% and quadrant A 18.8%. The chi-square test further showed no statistically significant differences in student thinking styles based on WBTL. The implication of the result was that students with certain style will not be different from students with another thinking style. In another study, Bawaneh et al. (2011b) investigated the effect of WBTL on students' understanding of simple electric circuits. Two hundred and seventy-three participants were randomly selected from four different secondary schools in the northern region of Jordan. Using a quasi-experimental 2 x 2 factorial design, students were grouped into an experimental and control group. The results showed that the experimental group (WBTL) was more successful than the control group (conventional method). In business management, le Roux (2011) used the whole brain model to determine students' thinking preferences which was then infused into the teaching of a large class. Using a mixed methodology, findings revealed that students could identify their thinking preferences. However, quadrant A and B thinkers, who dominated the class, were found to dislike group learning but showed a positive attitude to learning in the whole brain group after explanations. The WBTL has also been used in the field of taxation (van Oordtand, van Oordt, & du Toit, 2014), mathematics, (Özgen, Tataroglu, & Alkan, 2011) and auditing (Kirstein and Kunz, 2016) to determine the impact of a teachers' thinking preference on the designing of instruction and how it impacts on students learning.

3.2.3.2 Kolb's experiential learning

Kolb experiential learning was conceived from the experiential learning theory (ELT). The ELT is grounded in the theory of learning theorists such as John Dewey, Lev Vygotsky, Paulo Freire and Kurt Lewin, who place *experience* at the epicentre of learning (Kolb and Kolb, 2012). The ELT is a holistic theory which defines learning as an adaptation process involving the individual in his/her totality (Kolb and Kolb, 2012). Kolb's model is a "a cyclical pattern of learning from experience through reflection and conceptualising to action and on to further experience" (Reid, 2005, p. 61). Kolb's ELT classified individuals across two dimensions: (a) concrete experience (CE) and abstract conceptualisation (AC), as well as (b) active experimentation (AE) and reflective observation (RO) learning modes. Experiential learning is thus experienced when knowledge construction exists, bringing about pressure or conflict amongst the four learning modes towards providing a response to the required demands (Kolb and Kolb, 2005b). He, therefore, describes four-stage general learning styles for creating knowledge through the transformation of experience based on the two dimensions (De Boer et al., 2013; Kolb and Kolb, 2005b, 2009). The diagrammatical representation of Kolb's experiential learning with explanations is presented in Figure 3.8.

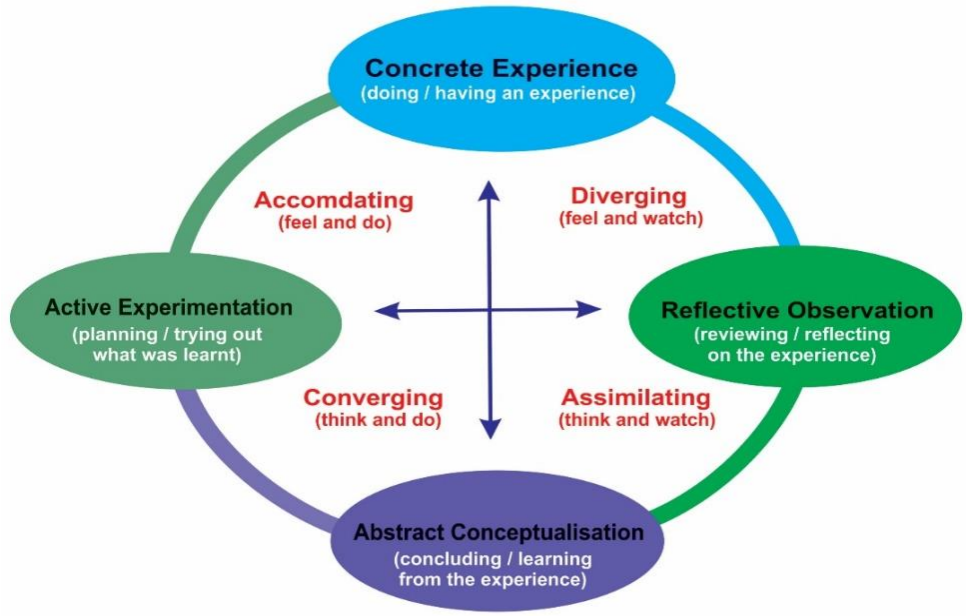


Figure 3.7: Kolb's cycle of experiential learning

- *Accommodators*: learn best through feeling (CE) and acting (AE) as a learning approach (Kolb and Kolb, 2012). They respond well to a learning environment that encourages hands-on experience. They are people oriented and because of that, during problem-solving, they depend on other people for information (Turesky and Gallagher, 2011). In a formal situation, assimilators respond well to group work, goal setting, fieldwork and testing different approaches for project completion (Kolb and Kolb, 2005a, 2005b).
- *Assimilators*: think (AC) and reflect (RO) as their learning approach. They respond well to organised and logical information (Kolb and Kolb, 2005a). They are not people oriented and prefer instructions involving abstract concepts and ideas. In a formal learning condition, the assimilators respond well to reading, lectures, thinking, reflection and exploration of analytical models (Kolb and Kolb, 2012).
- *Convergers*: learn via thinking (AC) and acting (AE). They prefer dealing with technical tasks and practically testing ideas and theories (Turesky and Gallagher, 2011). In a formal learning environment, the convergers respond to activities which experiment with new ideas, laboratory assignments, practical application and simulations (Kolb and Kolb, 2012).
- *Divergers*: choose to learn through feeling (CE) and reflecting (RO). These groups of learners respond well to situations that encourage explanation and brainstorming sessions (Kolb and Kolb, 2005a). They are people-oriented, imaginative and sensitive (Turesky and Gallagher, 2011). In a formal learning situation, they prefer

to engage in group work and discussions while listening with an open mind to varying ideas and, at the same time, receiving feedback (Kolb and Kolb, 2005b).

There are different versions of the Kolb's learning approach instrument, including version 3, 3.1 and 4.0. In more recent times, Kolb designed the adaptive style inventory which measures the degree to which the learning styles of individuals change in response to new learning situations (Coffield et al., 2004). The instrument has been used widely for research but, nevertheless, it has some weaknesses (Coffield et al., 2004; Manolis, Burns, Assudani, & Chinta, 2013). As stated by Coffield et al. (2004), too much expectation is placed on the instruments, and the Kolb's LSI is a 12 item inventory which does not seem to categorise one's behaviour accurately or objectively. The fixed questionnaire format does not allow respondents the opportunity to attune their concept of the item/s against the meaning of the author. Nevertheless, Kolb's experiential learning has been widely used and has also gained influence in the teaching of computer programming. For example, Campbell and Johnstone (2010) investigated the effect of Kolb's learning style on first year students' achievement in programming. A total of 74 students were exposed to Java programming and the OOP model to learn algorithms and problem solving for 16 semester weeks. Findings show that convergers and accommodators dominated the class with no statistical difference between their programming achievements. In an effort to study different students' performance and participation types in online programming, Shaw (2012) enlisted 144 students, who were doing a semester course in the ASP.NET programming language, in an experiment. The results showed that the "accommodator" style achieved higher scores than students with other learning styles. No significant correlation was found between learning and participation types. Çakıroğlu (2014) also studied the effect of learning styles and study habits in an online programming course among distance learners. Findings showed that all the styles were well represented in these distance learners with convergers topping the list who scored significantly better than other students.

These authors have drawn parallels between learning styles and students' performance in programming. This study will use students learning information to plan for instruction. In addition, the whole brain teaching and learning, *and* the Kolb's experiential learning will be combined within a constructivist learning environment for the planning and facilitation of programming to first year students. Relating the two learning modes, Potgieter (1999) explained that a relationship exists between Kolb's experiential learning theory and the whole brain model when the learning models were compared. Based on the author's classification, the following comparisons; a) abstract conceptualization (A quadrant), b)

active experimentation (B quadrant), c) concrete experiment (C quadrant) and d) reflective observation (D quadrant) were revealed.

Potgieter (1999) claimed that a further cross comparison of the four learning styles of Kolb's experiential learning theory with the whole brain thinking preferences results into the following classification:

- Divergers – C and D quadrants thinking
- Convergers – A and B quadrants thinking
- Assimilators – A and D quadrants thinking
- Accommodators – B, C and D quadrants thinking

The classifications above agree with Herrmann's findings on the double and triple brain dominances (Potgieter, 1999). However, one can infer from the classification that no one Kolb's learning style can be wholly be placed in a quadrant of the whole brain and vice versa, but instruction can be designed to cater for all the learning styles in the classroom.

3.2.4 Educational taxonomies and assessment in programming

Taxonomies are educational objectives used to provide a common language for describing learning outcomes and performance assessments (Fuller et al., 2007). In education, and particularly in computer science education research, various methods have been used to assess students' learning. The revised Bloom's and SOLO taxonomies have gained popularity and are considered relevant. The following subsections discuss the taxonomies.

3.2.3.1 The Revised Bloom's taxonomy

Outside the philosophy of teaching and learning, many studies have set out to explore approaches to the practice of teaching (Williams, 2015). One such study is the taxonomy of educational objectives by Bloom (1913 - 1999), an educational psychologist whose work *Bloom's taxonomy* gained wide support. He used the terms cognitive, affective and psychomotor domains. The basis for the cognitive domain of his taxonomy was the classification of educational goals and objectives which students must achieve during the learning process. This resulted in six categories of learning namely: knowledge, comprehension, application, analysis, synthesis and evaluation. The taxonomy was first described as a hierarchical model but further analysis and discussion have allowed for the overlapping of some categories (Meerbaum-Salant et al., 2013; Williams, 2015). However, the structure remains hierarchical in nature. Anderson, Krathwohl, and Bloom (2001) and Krathwohl (2002) refined the original taxonomy to create a newer version, *revised Bloom's taxonomy* (see Figure 3.8), which is believed to better fit the educational practices of the 21st century. In this model, each level is subsumed by a higher and more complex level. It

is believed that a student who has reached the higher level of “creating” has also mastered the other five lower levels.

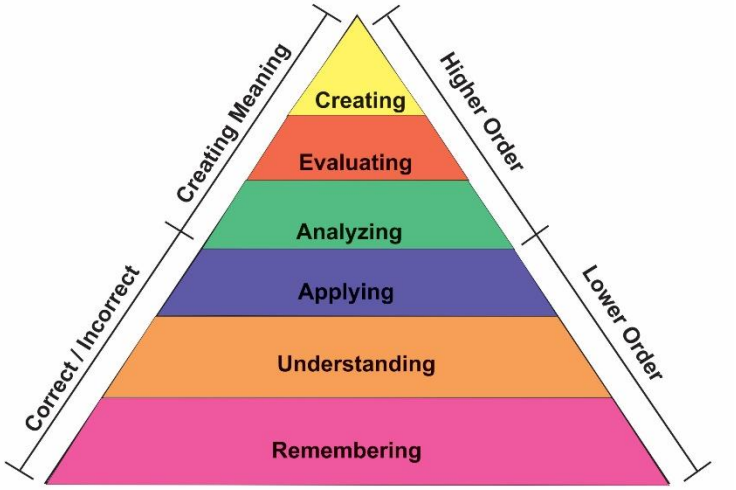


Figure 3.8: Revised Bloom’s taxonomy
Source: Forehand (2010, p. 43)

Krathwohl (2002) further refined the taxonomy in a two-dimensional matrix form, mapping *both* the knowledge and cognitive dimension (see Table 3.2). The application of this matrix involves examining and classifying the learning objectives into appropriate cells. In explaining the matrix, the verbs in the cognitive dimension (see Table 3.2) are self-explanatory. The factual, conceptual and procedural knowledge categories of the matrix fit well with the knowledge types for programming, as described by McGill and Volet (1997). Metacognitive knowledge refers to the general cognitive knowledge gathered by a student in combination with his/her awareness and knowledge of his/her own cognition (Krathwohl, 2002). A further explanation of the significance of metacognitive knowledge to teaching and learning has been described by Pintrich (2002). He stressed that metacognitive knowledge of the taxonomy supports the constructivist learning theory of Piaget and Vygotsky as both these theories stated that development occurs as students become aware of their own thinking and cognition. Even though the name ascribed to metacognition differs according to the theories of learning, knowledge of cognition and self-regulatory processes, both these terms are accommodated in the taxonomy (Pintrich, 2002). He further divided *metacognitive* into three domains namely strategic, cognitive and self-knowledge. *Strategic knowledge* entails the knowledge of approaches for learning, reasoning and problem-solving about the knowledge domain. *Cognitive knowledge* entails the ability to differentiate between the “what” and “how” of strategies and the “when” and “why” of the adopted strategies to the learning situation. The third type, *self-knowledge* is understanding one’s own learning

strengths and weaknesses. This tabular matrix can be applied in planning and teaching any subject course for a semester (Fuller et al., 2007; Meerbaum-Salant et al., 2013; Thompson, Luxton-Reilly, Whalley, Hu, & Robbins, 2008).

Table 3.2: Revised Bloom's taxonomy matrix

	Cognitive Processes					
The Knowledge Dimensions	1. Remember	2. Understand	3. Apply	4. Analyze	5. Evaluate	6. Create
Factual						
Conceptual						
Procedural						
Metacognitive						

Source: Krathwohl (2002, p. 216)

Several studies have researched the efficiency of the revised Bloom's taxonomy in computer science. Shneider and Gladkikh (2006) conducted a related study. They planned diagnostic assessments, system analysis and design by using the revised Bloom's taxonomy. In their study, Starr, Manaris, and Stalvey (2008) also researched learning objectives in computer science and found that the Bloom's taxonomy helped the faculty to communicate more effectively and aided in building a stronger departmental assessment. The revised Bloom's taxonomy was also found useful in formulating program comprehension questions, however, educators still believe it poses difficulty in categorising a completed question within the cognitive dimension. Thompson et al. (2008) therefore formulated a common understanding of how the revised Bloom's taxonomy could be used in the computer science domain. They focused on introductory programming exams using the revised Bloom's taxonomy.

Though the revised Bloom's taxonomy has been widely used, and has proved efficient, it was *not* designed with university teaching in mind *or* for formulating intended learning outcomes (ILOs), *but* for the purpose of selecting representative tasks for examination (Brabrand and Dahl, 2008). However, Biggs and Collis (1982) designed the SOLO (structure of the observed learning outcome) taxonomy which is used to investigate and assess students' competencies in higher education.

3.2.3.2 Structure of the observed learning outcome (SOLO) taxonomy

One of the prime themes of investigation in computer science education research is the development of students' program understanding, from *concrete* representations to *abstract* mental models (Cortney, et al., 2011). An investigation into students' ability to identify the connection/s between abstract programming concepts and concrete programming concepts is a recent global research area (Lister, Fidge, & Teague, 2009; Sheard et al., 2008). Biggs and Collis (1982), researching this domain, produced a promising educational taxonomy framework which has contributed greatly to how novice programmers manifest their understanding of programming constructs.

The SOLO taxonomy of Biggs and Collis (1982) was devised to help educators formulate desired learning objectives (Thota and Negreiros, 2015). It has been adopted in computer science as a framework for classifying students' responses to problems, specifically to analysing computer problems in simple English (Whalley et al., 2007). It differs from other taxonomies in that it determines the content of the learner's response on a programming code (Fuller et al., 2007) and level of comprehension. This can be used to measure a student's level of abstraction and understanding in learning to program.

Armoni et al. (2015), Brabrand and Dahl (2008) and Lister, Simon, Thompson, Whalley, and Prasad (2006) described the different levels of the SOLO taxonomy in line with the required cognitive processes necessary to attain them. The SOLO levels tend to be *progressive* in nature, from quantitative to qualitative and are defined to match the growing structural complexities of learning (Brabrand and Dahl, 2008; Thota and Negreiros, 2015). The diagrammatical representation is presented in Figure 3.9 and a description of each solo level is presented in Table 3.3.

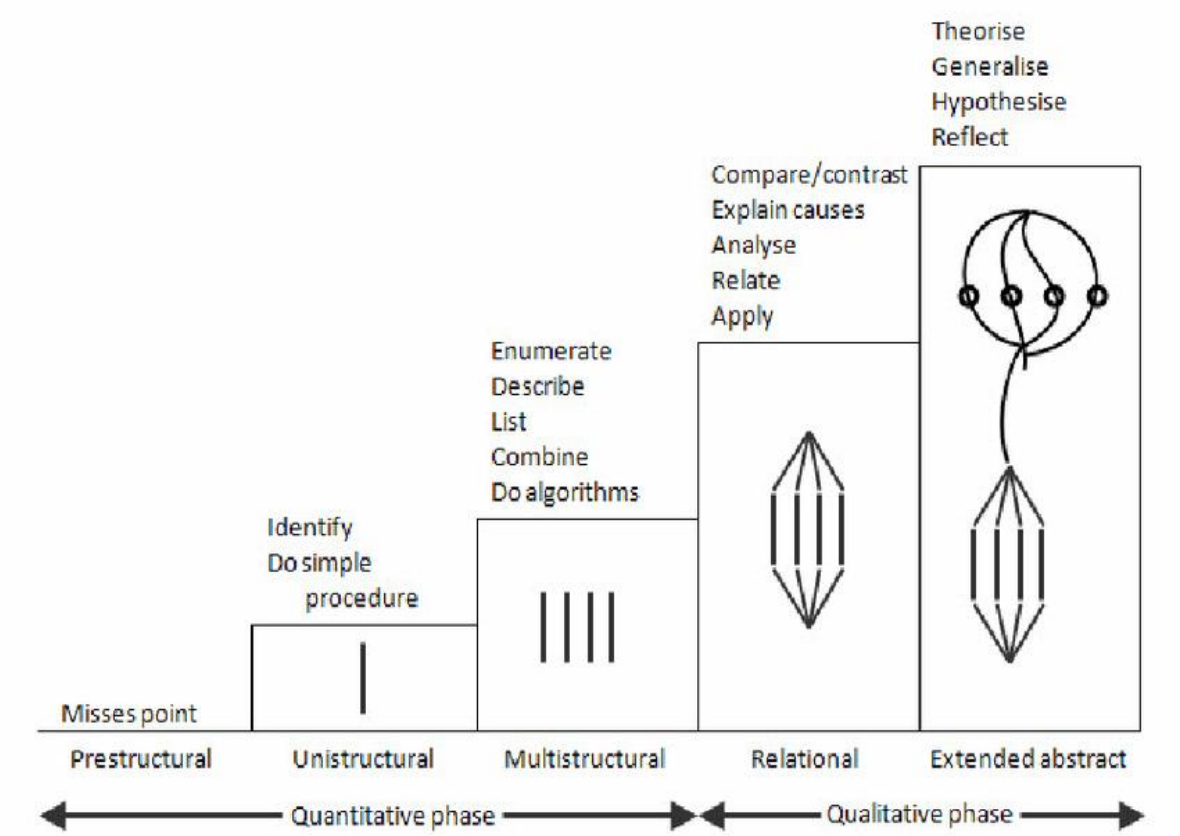


Figure 3.9:The SOLO taxonomy
Source: Biggs and Tang (2011, p. 91)

Other significant features of the SOLO taxonomy include its hierarchical structure and the progression of student learning from quantitative outcomes (associated with levels 2 and 3) to qualitative conceptions (levels 4 and 5) which leads students to higher order qualitative outcomes (Biggs, 2012). It is also noted that *surface learning* occurs at the quantitative stage while *deep learning* occurs at the qualitative stage (Biggs, 2012; Biggs and Collis, 1982; Brabrand and Dahl, 2008). The SOLO levels will be discussed in their order of complexity (Biggs, 2003; Biggs and Tang, 2007; Biggs and Collis, 1982; Clear et al., 2008b). The importance of the SOLO taxonomy is based on its potential to provide a reliable classification of student responses to the *explain in plain English* questions (Tan and Venables, 2010). SOLO can be used in all phases of teaching. According to Biggs, a) “it is useful for designing learning objectives or the intended outcomes of teaching a topic or course in either quantitative or qualitative terms”; b) “designing the teaching activities appropriate for achieving those outcomes”; c) “assessing how well the student has learned what is intended to be learned” and d) “for outcomes-based design for teaching known as constructive alignment” (Biggs, 2012, p. 123).

Table 3.3: Description of SOLO levels

SOLO levels	Description
Pre-structural level	The student has no understanding of the content. He/she may have acquired scattered pieces of information which are unstructured and negate the actual concept or does not reveal programming constructs and is unrelated to the question. This is the lowest response type the student can provide.
Unistructural	A local perspective in which a student can solve one aspect and make a clear connection.
Multi-structural	The student understands the concept but has multiple responses that are not related to each other. Allegorically, the student only sees the *trees* and not the *forest*.
Relational	The student can now see connections and has a holistic view of the content. The student now sees *many trees* from a *forest*.
Extended abstract	It is the highest SOLO level and goes beyond the current problem to a broader context. The student can apply the learning from this context to another domain.

Many researchers have conducted end-of-semester examinations using the SOLO taxonomy in programming courses and these have been well document in the BRACELet body of literature. For instance, Lister et al. (2006) applied the SOLO taxonomy to study novice programmers' levels of understanding about a code. Findings showed that most students read a program multi-structurally, line by line, and weak students have difficulties abstracting the code. Also, extended abstract was *not* observed in the focus group, meaning that students failed to see the forest from the trees. Sheard et al. (2008), while building on the work of Lister, addressed noticeable deficiencies found in the study, and reported that *their study* supported the contention that a better SOLO reading performance produces better code writing. This, therefore, produced higher SOLO responses in postgraduate than undergraduate students.

While investigating students' competency in a university program, Brabrand and Dahl (2009) used the SOLO taxonomy to analyse students' learning. They found that the SOLO taxonomy encouraged competency development, from undergraduate to graduate level. Other research studies (Lister et al., 2009; 2006; Sheard et al., 2008; Shuhidan, Hamilton, & D'Souza, 2009) confirmed SOLO to be a significant and efficient framework for studying students' progress in programming and representations of programming concepts, and useful for measuring syntax knowledge. Researchers have established the significance of the revised Bloom's and SOLO taxonomies in defining learning outcomes. However, some researchers (Fuller et al., 2007; Lahtinen, 2007) in the field of computer science have not fully endorsed the use of *either* the revised Bloom's *or* SOLO taxonomies to the computer science discipline. There was a strong argument that both Bloom's and the revised Bloom's taxonomies do *not* provide good programming related tasks. Therefore, a case was made

for combining the two taxonomies in computer science. Examples of such hybrid studies are now explained.

3.2.2.3 BLOOM'S and SOLO in computer science courses

SOLO levels are somewhat related to the revised Bloom's taxonomy and, as such, have been applied to different subject areas (Williams, 2015). Shuhidan et al. (2009) conducted a taxonomic study of programming assessment in a Java programming course. Questions were categorised and analysed in the context of the two taxonomies and findings showed no significant correlations between the two taxonomies, instructor level of complexity and novice level of difficulty. Another significant finding is that 35.6% of novices' responses were found to be situated in the relational and multi-structural category (23.3%) whilst less than 25% were found in the lower categories. The researchers however, find it difficult to classify questions using the Bloom's taxonomy and stressed that the taxonomy is specifically suitable to the educational field.

Whalley et al. (2006) extended the previous study of McCracken et al. (2001) and investigated novice programmers' reading and comprehension skills in programming using the revised Bloom's to analyse and generate reading questions while the SOLO taxonomy was used in the analysis of data. Their results produced two independent analyses. They found the revised Bloom's taxonomy very challenging to use in the categorisation of programming multiple-choice questions (MCQs), even by experienced educators. This indicates a deficiency in the taxonomy. They also found that weaker students' performance could not attain the higher level whilst stronger students tended to reach the higher level of the SOLO taxonomy.

Armoni et al. (2015) developed a two-dimensional taxonomy which combines the two taxonomies. The first dimension consists of three SOLO levels and the second dimension of three Bloom's levels. The combined taxonomy was adapted to computer science. The resulting hierarchy corresponds to the cognitive levels, from unistructural understanding to relational creating. Meerbaum-Salant et al. (2013) also combined the revised Bloom's and SOLO taxonomies for constructing questionnaires and tests on Scratch in middle schools. They used both taxonomies to create a new taxonomy. From the SOLO taxonomy, they focused on unistructural, multi-structural and relational while they used understanding, applying and creating from the revised Bloom's taxonomy. Findings from the study showed that student performance was higher on the *relational applying* than the lower levels of *multi-*

structural creating. Nevertheless, they stressed that the combined taxonomy captured cognitive characteristics in computer science practice.

The reviewed literature thus designed programming assessments using *both* the revised Bloom's and SOLO taxonomy. It is necessary to include and further investigate further in the context of the thesis as some researchers frown at the use of *both* taxonomies for designing programming tasks, even though this approach is widely employed. Table 3.4 depicts the nine new taxonomies resulting from the combination of the revised Bloom's and SOLO taxonomies which will be adopted in this study. The new taxonomies are: unistructural understanding, unistructural applying, unistructural creating, multi-structural understanding, multi-structural applying, multi-structural creating and relational understanding, relational applying and relational creating.

Table 3.4: Student's cognitive performance

Uni-structural			**Multi-structural**			**Relational**		
U	A	C	U	A	C	U	A	C

U - Understanding; A - Applying; C - Creating

The essence of assessment of students' knowledge is to determine their acquired skills in programming and several workshops and working groups, such as ITiCSE and Leeds, have been conducted to focus on programming assessment (Clear et al., 2008a). Lopez et al. (2008) and Philpott, Robbins, and Whalley (2007) highlighted that a component of examination questions must be based on the accepted set of fundamental skills in programming. Therefore, a student should possess explaining skills, tracing skills and writing skills, which are hierarchical in nature. The lowest level in the hierarchy (explaining skill) involves possessing knowledge and understanding of elementary programming constructs such as data types, selections and iterations. The middle level (tracing skill) involves the possession of accurate skills to read and trace code. The highest level of hierarchy identified refers to the ability to write non-trivial and correct code using programming constructs. It has been stressed that any examination question should comprise at least one of the three fundamental programming skills while relating the revised Bloom's taxonomy to the skills of understanding (explaining), tracing/reading (applying) and creating (writing) (Tan and Venables, 2010).

3.2.2.4 Constructive alignment for designing courses in Higher Education

Constructive alignment (CA) is a framework for teaching and learning, designed by (Biggs, 1996), which illustrates a paradigm shift from a *teacher-centred* to a *student-centred* teaching approach. The *constructive* element was taken from the meta theory of constructivism which theorised that learners construct their own meaning based on existing schemata. *Alignment,* as further explained, is a principle in curriculum theory which stipulates that assessment be well aligned to intended learning outcomes, such as in criterion-referenced assessment (Biggs and Tang, 2011). The idea behind criterion-referenced assessment is that the assessment should be about how well students meet standards of learning (Burton, 2006). This means that lecturers do not just *teach,* but *design* activities that will enable students to achieve the intended learning outcomes (Biggs and Tang, 2011). A definition of a constructively aligned course is summarised by (Brabrand and Dahl, 2008, p. 1).

A course is said to be constructively aligned when:
- Intended learning outcomes (ILO) are explicitly formulated as operational competencies; - The ILO competences are explicitly communicated to the students (early in the course); - The exams precisely measure the ILO competences and - The teaching/learning activities (TLSs) match the ILO competences.

Biggs (2012) and Wiggins (April, 2011) condemned the use of norm-referenced assessments in undergraduate education and supported the use of more qualitative standards of achievement at this level of education since students are being trained to be professionals. The Nigerian education system advocates learner-centred teaching (NCCE, 2012). This may be true but the use of constructive alignment in designing courses is absent. In order to fill this gap, the researcher set out to design introductory programming courses using CA. The operational framework of teaching design towards a constructive alignment (Biggs and Tang, 2011, p. 100) is discussed below.

Stage 1: Defining intended learning outcomes (ILO)

The intended learning outcomes (ILO) are statements indicating the level of understanding and performance expected of students through engagement with the teaching-learning experience (Biggs, 2012; Biggs and Tang, 2011). They are written from the students' perspectives. Students, therefore, view aspects of the world differently when understanding is fully gained. Thus, ILO should not only be a specification of "what to be learned" but "how

it is to be learned" and "to what standard" or level. As explained, the content topic is *what to be learned*, while the verbs selected should describe the *content* (how) and *context* (level) in which the students have to do it (Biggs and Tang, 2011). To describe the typical verbs that can be used to determine the level of understanding expected of students in achieving learning outcomes, the use of SOLO and the revised Bloom's taxonomy were suggested. A clear distinction should be made between the *kinds* of knowledge (declarative, functional and conditional) to be addressed when defining ILO.

Stage 2: Choosing teaching/learning activities (TLAs)

The teaching/learning activities centre on performing the verbs already highlighted in the intended learning outcome statements. Once the ILOs have been identified, teachers need to provide support along with several activities to help students learn all the ILO competences (Brabrand and Dahl, 2008). Complex TLAs may also be broken down into parts to enable students to practice those sections which they are having difficulties with (Biggs, 2012). Since there are different types of knowledge, the teaching/learning activities must be designed towards achieving the ILO of each of the knowledge types.

Stage 3: Designing assessment tasks

The assessment tasks address the same verbs enacted in the ILO. Biggs (2012) stressed that alignment is perfect when assessment tasks are TLA itself. Rubrics can be used for assessment at different grade levels or for affording qualitative percentage marks. Qualitative ILOs require using qualitative assessment wholly, and not analytically. Analytical assessment is suitable for formative assessment while summative assessment must be based on the total performance of the student. Assessment by portfolio requires students to place a sample of performances stated in the ILOs and stating reasons why think they do. This is used for reflective self-assessment and high-level verbs (Biggs, 2012).

Stage 4: Grading procedures

This is the final step which supplies evidence as to how students have achieved the ILOs. Grading involves two aspects: assessing students according to the stated ILOs and combining results from several ATs which can be done quantitatively or qualitatively (Biggs, 2012). CA is a systemic innovation which cannot be effectively used in any institution that requires students to be graded through pegging a certain grade for examination assessment (Biggs and Tang, 2011). This will hamper the alignment process between ILOs and assessments. Therefore, for CA to be effective in any institution, it needs to be entrenched in an accommodating culture, from department to faculty to institution and national levels (Biggs and Tang, 2011).

Research has indicated that CA has been used as a reflective framework in several study courses including: marketing (Kuhn and Rundle-Thiele, 2009), biochemistry (Hartfield, 2010) and computer science (Cain and Woodward, 2013; Thota and Whitfield, 2010), while findings are linked to positive results. In the context of this thesis, the combined revised Bloom's and SOLO taxonomy of Meerbaum-Salant et al. (2013) were used, taking into consideration the relevance of constructive alignment in designing a procedural programming course. These were done towards determining students' programming skills of tracing, reading and writing. All these aspects were incorporated into the design of the first teaching and learning framework for procedural programming in the college. The framework is discussed below.

3.3 TEACHING AND LEARNING PROCESS FRAMEWORK

Salient points from literature necessitated the design of a teaching and learning process framework (see Figure 3.12) for the teaching of programming to first year college students. The teaching and learning process framework (TLPF) was designed and developed based on the ADDIE framework for instructional design (Bichelmeyer, 2005). The framework was used because of its simplicity and its prescriptive characteristics for developing instruction (Brown and Green, 2015). This framework allows for systematic design and planning of learning experiences, with the outcome of each phase informing the other phases (Khalil and Elkhider, 2016). It has five phases namely; analyse, design, develop, implement and evaluate. There are some criticisms against the ADDIE relating to inefficiency, an inability to generate quality and obsoleteness. However, despite these criticisms, ADDIE continues to hold solid ground (Bichelmeyer, 2005). He, therefore, suggested that teachers, instructors and designers should see ADDIE as a conceptual framework and use its different manifestations in everyday settings.

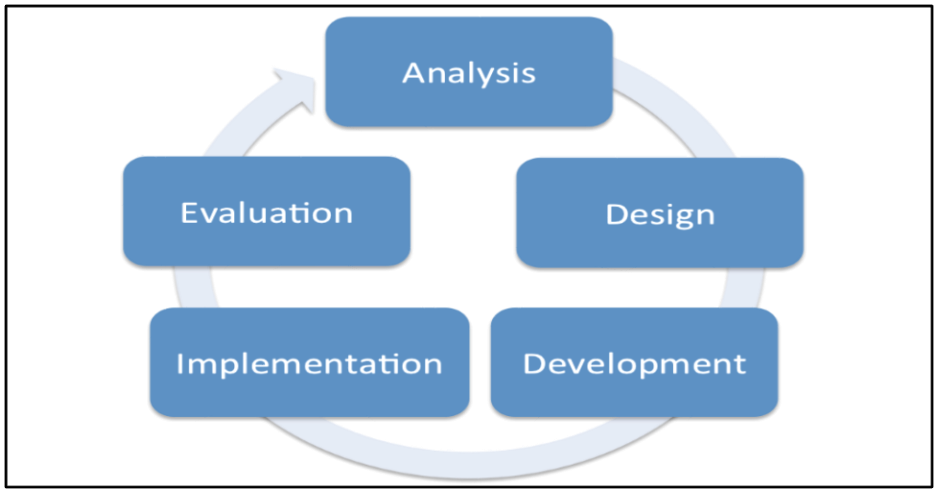


Figure 3.10: The ADDIE instructional design framework

In describing the phases of ADDIE, the ***analysis phase*** requires identifying the causes for a performance gap (Branch, 2010) and the setting of goals. At this stage, three major steps were taken. As described by Branch (2010) and Brown and Green (2015), a *needs analysis* was conducted to determine *what* the learners know and what they do not know. The essence of this was to avoid duplication of the performance gap. The *task analysis* involved the gathering of information relating to the content of the instruction to be designed and other documents necessary for designing said instruction. Brown and Green (2015) noted that the task analysis should involve: a statement of goals and learning objectives, a well-prepared component of each learning goal, a knowledge type and the necessary skills students must have, teaching strategies and a selection of appropriate media. The *learner analysis* refers to the determination of learner characteristics that will aid in the designing of an effective instruction. The three processes were considered in this research study during the analysis stage. The underlying motivation for the design of this framework was to teach students Scratch and procedural programming using a holistic approach in order that they learn programming skills. As a result, the researcher, and one other lecturer teaching introductory programming, identified the learning goal and objectives from the NCCE curriculum document for computer science. Once the learning objectives had been identified, the knowledge type and skills which students should possess, were determined. Strategies needed for teaching and learning were identified and discussed in Chapter 2 of the thesis. The media needed for disseminating instruction was also identified. Since the students were newly enrolled, their characteristics were determined using the learning approach instruments (KLSI and HBDI) to inform the instruction design.

The design phase involves verifying the preferred performances (Branch, 2010). It involves the organisation and creation of instructional goals which will guide students in the instruction (Brown and Green, 2015). In this study, the design phase involved constructive alignment (*q.v.* section 3.2.4) which guided the instruction and assessment planning (Appendix B 1 and Appendix B 2) for the procedural programming course. The researcher designed the CA for procedural programming with guidance from the study of Khalil and Elkhider (2016) and Thota and Whitfield (2010). Figure 3.11 represents the process through which the constructive alignment was conceived. The outcome of the design phase guided the planning of the ***development phase*** which gave rise to the initial teaching and learning process framework ($TLPF_0$). In this instance, the content and supporting media were organised to achieve an objective and create a satisfying learning experience for the students (Branch, 2010; Brown and Green, 2015), by incorporating different methods to communicate content to the learner. As shown in Figure 3.12, the content learned by students were the QBASIC and Scratch programming, designed based on the Block model.

This is the assimilation process students used in building a cognitive structure and form a mental representation of the program text (see section 2.3.2.4). In addition, the prepared content was further developed in a student-centred format, based on the constructivist theory of learning and whole brain ways of facilitating (see sections 3.2.2 and 3.2.3.1). Attention was given to the processing of information from the environment and this also guided the design of instruction.

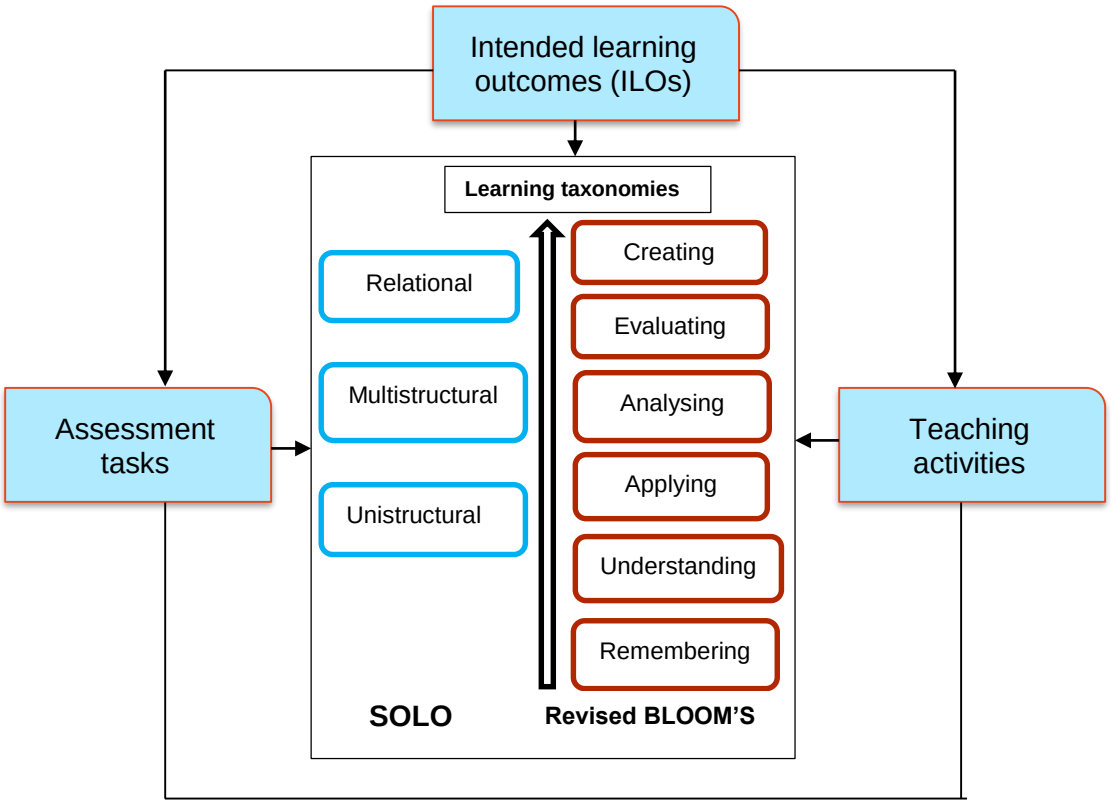


Figure 3.11: Constructive alignment (CA) for Programming at the Design phase
Source: Adapted from Khalil and Elkhider (2016, p. 152)

The next ADDIE phase is the ***implementation phase*** where the actual teaching and learning process are situated and where the student is prepared through engagement with the learning environments (Brown and Green, 2015). Therefore, teaching and learning were implemented in a socially constructed environment by allowing the student to construct meaning in the learning experiences. To achieve this, instructional activities were based on cooperative learning, pair programming and scaffolding towards the achievement of the designed learning objectives. The last phase is the ***evaluation phase*** where the effectiveness of the teaching and learning process framework and student achievement were determined (Brown and Green, 2015). According to Branch (2010), the evaluation phase is based on three criteria namely "determination of evaluation criteria, selection of

tools and conduct evaluation". He further noted that the framework should also be evaluated to ascertain whether it satisfies the standards set in the design stage. Therefore, in this research study, student achievement was evaluated through the SOLO and revised Bloom's taxonomies after learning had taken place. Criteria for evaluating students include formative and summative assessments such as tests, assignments, quizzes and project works amongst others, which align with the intended learning outcomes for student learning. Authentic assessments which focuses on the essential experiences students gained during learning (Wiggins, April, 2011) was also used. Frey, Schmitt, and Justin (2012) noted that authentic tests must include *real-world activities*, reflect the *complexity*, involve *high level of thinking* and *group-based* with each student's contribution towards the success of the project. The product rendered at the end of these phases, during the first cycle, will be the first $TLPF_1$ (represented as $TLPF_1$+1) which is meant to be develop further as the action research cycle continues. The outcomes of this first cycle action research enhanced the design of an improved teaching and learning process framework for the second cycle action research ($TLPF_2$). The outcome of the second cycle was further incorporated into the $TLPF_2$ towards the development of a new teaching and learning process framework ($TLPF_3$) for the teaching of procedural programming in Nigeria. The teaching and learning framework were further evaluated by experts in the field.

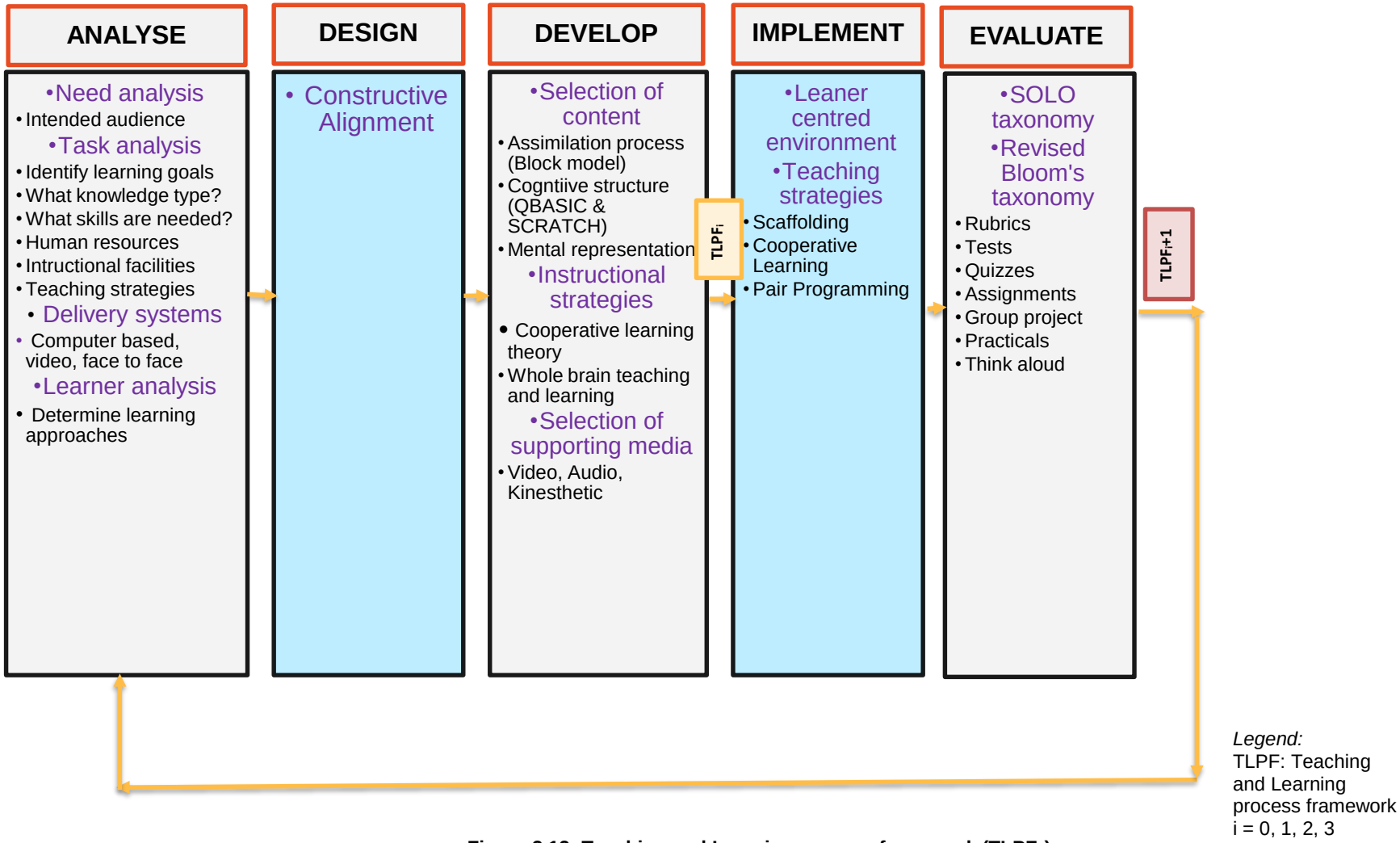


Figure 3.12: Teaching and Learning process framework ($TLPF_1$)

3.4 CHAPTER SUMMARY

In summary, this chapter was set out under four headings and commenced with a brief introduction. Since the research work is about teaching first year students programming, the researcher reviewed the literature on learning theories with constructivism as the focus. Further examination of learning led to the review of the literature on learning taxonomy (see section 3.2.2) while focusing on the revised Bloom's and SOLO taxonomies. The importance of constructive alignment in the design of a programming course that is student-centred was discussed. In addition, consideration was given to diversities of learners in a classroom and how this would affect the design of instruction. Therefore, literature on the Herrmann whole brain model and Kolb's learning approaches were reviewed. To design a conceptual framework for the study, the salient points in both Chapters 2 and 3 were incorporated to form a teaching and learning process framework for the study. The framework served as a guiding principle on which the research questions in the study were based and further reflected upon.

CHAPTER 4: METHODOLOGY AND METHODS

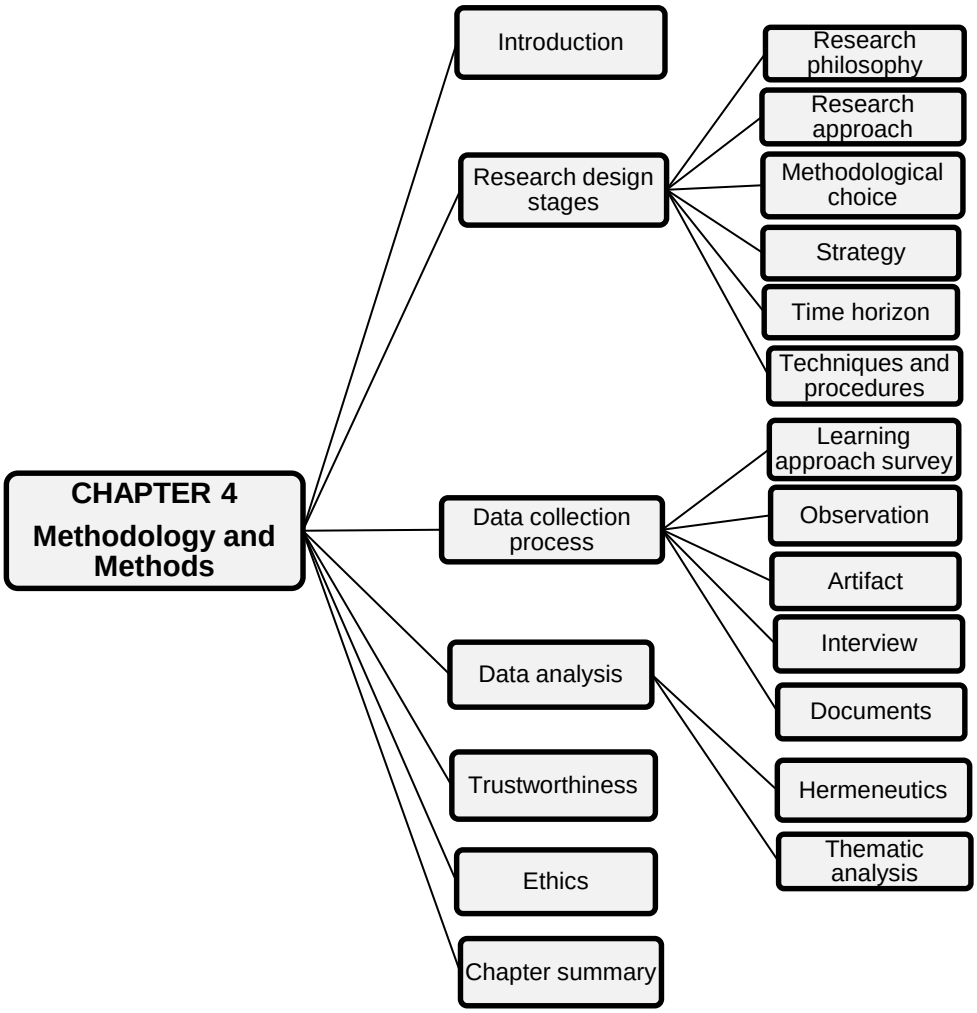


Figure 4.1: Diagrammatical representation of Chapter 4

4.1 INTRODUCTION

Research is the art and science of collecting trustworthy data to answer predetermined research questions. The previous chapters reviewed the literature on programming and program comprehension (*q.v.* Chapter 2) as well as learning perspectives and learning theories (*q.v.* Chapter 3) on which this study is based. The teaching and learning process framework for the study (*q.v.* Chapter 3) was also derived. This chapter will explain *how* the research design developed from the teaching and learning process framework and how the data collection and data analysis progress are made explicit from the research design. It is, therefore, necessary to revisit the research questions and purpose of the study. The primary research question investigates how the implementation of a visual programming environment inform the design of a new teaching and learning framework and as well

support the learning of procedural programming skills in such a manner that teaching intervention planned upon students' learning approaches is positively experienced while promoting mental representations of the Block model in first year college students?

To investigate the primary research questions further, the following secondary research questions were addressed:

SRQ1: How do first year college students in the procedural programming classroom approach learning?

SRQ2: How do the participating students experience the teaching intervention designed based on their learning approaches with a view to promote the learning of procedural programming?

SRQ3 How does the visual programming environment (VPE) support learning of procedural programming in first year college students?

SRQ4 What are the mental representations of the Block model held by first year college students in the procedural programming classroom?

The purpose of these questions was to describe the way in which students approach learning in the classroom whilst planning and designing instruction based on the learning approaches to facilitate learning within a whole brain and constructivist learning environment. In addition, the individual and shared experiences of students regarding visual programming were studied to determine how this supports the learning of procedural programming. Section 4.2 discusses the research design processes along with the research design stages (Crotty, 1998; Saunders, Lewis, & Thornhill, 2012). Saunders' *research onion* was unpacked and used to describe and guide the study's research design process. The stages follow the order: research philosophies, approaches, choices, strategies, time horizon, techniques and procedures. In section 4.3 the data collection process is discussed while a description of the data analysis is presented in section 4.4. Following this, trustworthiness (section 4.5), ethics guiding the research (section 4.6) and the summary of the chapter (section 4.7) are presented.

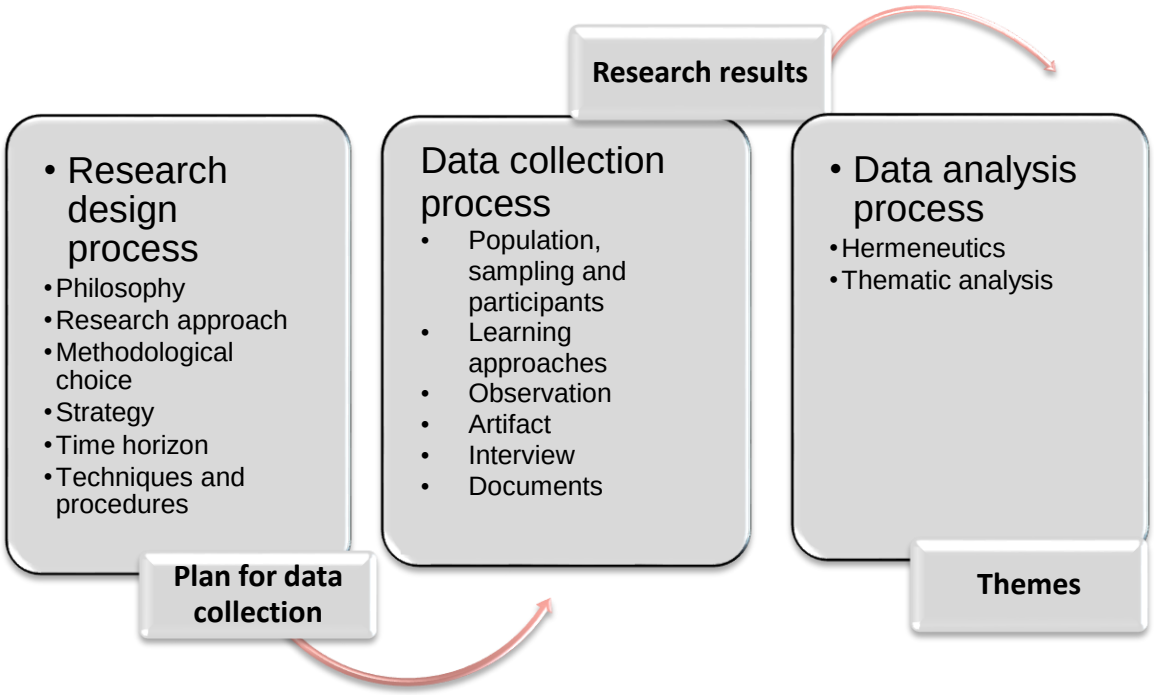


Figure 4.2: Main processes in research methodology

The research methodology chapter is further divided into three main processes (Langenhoven, 2015). The *research design process* outcomes inform the data collection. The *data collection process* informs the research results which are used for data analysis. The end products of the *data analysis process* are the generated themes and subthemes (q.v. chapters 5 and 6). The diagrammatical representation of the chapter is presented in Figure 4.2.

4.2 RESEARCH DESIGN PROCESS

This section explains the first part of the three research methodology processes.

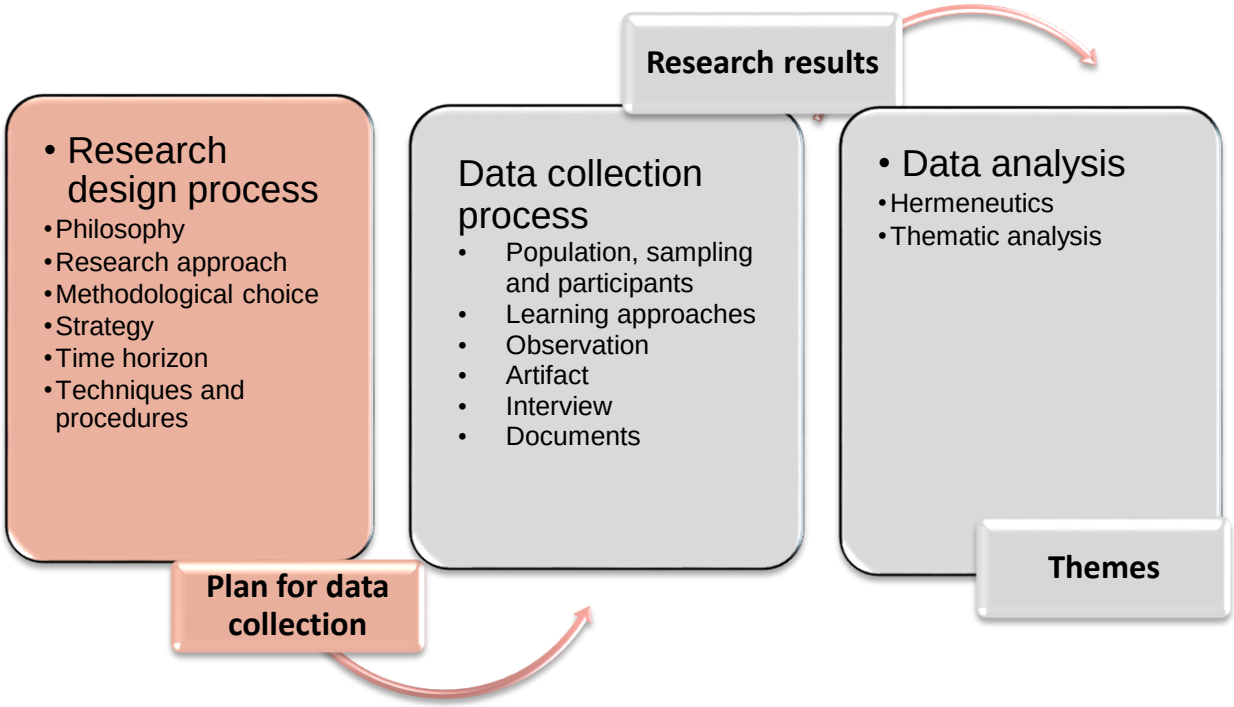


Figure 4.3: Research design process

Research, as defined by Saunders et al. (2012, p. 5), is "something that people undertake in order to find out things in a systematic way and thereby increasing knowledge". In the process of research, many scholars disagree on the names, stages and nature of the research process. This disagreement, as noted in Saunders et al. (2012) and Crotty (1998) will be discussed in order to clarify the study's research process to the reader. Crotty classifies research into four stages: epistemology, theoretical perspective, methodology and methods. *Epistemology* is the philosophical base for identifying, explaining and justifying possible and acceptable knowledge. It deals with *how* we come to terms with *what* we know. Statements or assumptions brought into the research reflect or guide the methodology and defines the *theoretical perspective*. *Methodology* refers to strategies that guide the choice of methods, connecting them to the desired outcome. *Methods* are techniques and procedures engaged to gather and analyse data. Saunders and colleagues, on the other hand, classified the research process into six layers: philosophies, approaches, strategies, choices, time horizons, techniques and procedures. These phases are illustrated in the *research onion* (*q.v.* Figure 4.4). *Research philosophy* is researchers' assumptions about the way the world is viewed. These assumptions support the research strategy and methods chosen. Hence, Saunders et al. (2012) and Crotty (1998) expressed philosophy as *ontology* and *epistemology*. Ontology is the nature of reality while epistemology is what adds up to acceptable knowledge in a research study. Here, Crotty used epistemology and claimed ontology as meaning theoretical perspective. He added that ontology should be used when

we need to talk about “being”. According to him, the two research assumptions (theoretical perspective and ontology) overlap. The use of methodology and strategy by these two authors has common characteristics and meaning. In addition, the methods of data collection and data analysis, as used by (Crotty, 1998) and (Saunders et al., 2012) differ, but still reveal the same meaning. Saunders’ layers also extend to choices, time horizon and approaches which, in contrast, are not revealed in Crotty’s stages of research. The researcher found Crotty’s stages of research too tightly packed. Therefore, to ensure uniformity throughout the study and to describe the study in detail, Saunders’ research onion was helpful in justifying the researcher’s decision. The layers are discussed in subsections 4.2.1 to 4.2.6.

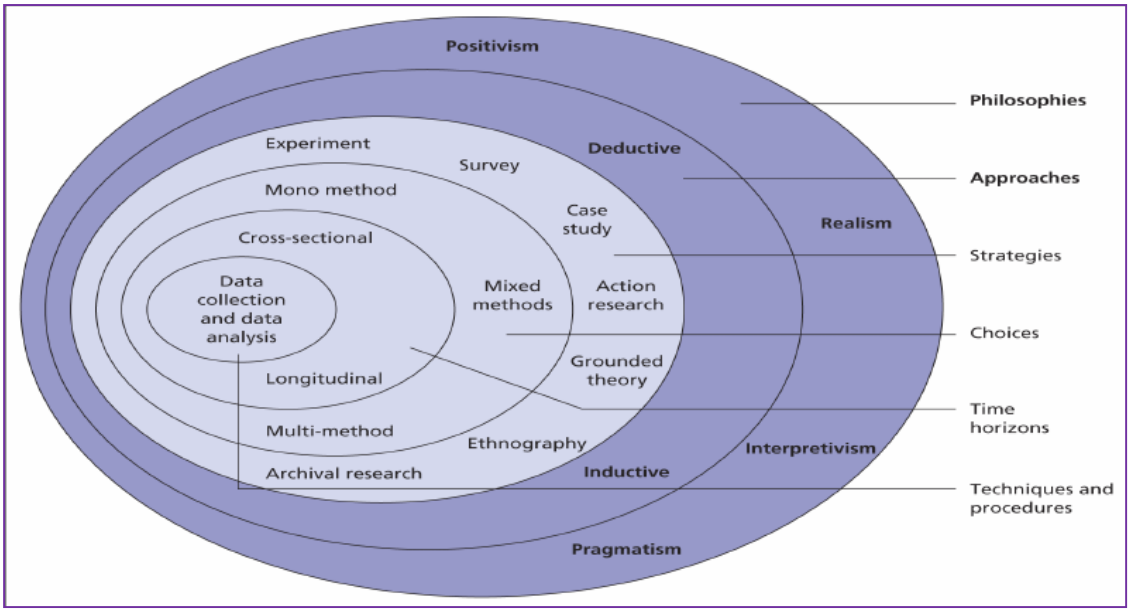


Figure 4.4: The research onion

Source: Saunders, Lewis, and Thornhill (2009, p. 138)

4.2.1 Philosophies

Philosophy is the most abstract level of decision making in research and, as such, it controls research practice (Creswell, 2014). Cohen, Manion, and Morrison (2011, p. 3) described *philosophy* as “perception about the world which is driven by one’s orientation, understanding and purpose of understanding the world, as well as what is deemed valuable”. The consideration of philosophical issues early enough in the research is important, as it forms an integral part of the research process namely researchers’ thoughts about the research (Scott and Usher, 2011).

Social science has two major ways of viewing research philosophy namely *ontology* and *epistemology.* These two viewpoints directly influence the strategy and methods adopted

in the research process (Saunders et al., 2012). *Ontology* relates to the *study of being*, the nature of reality or the essence of the phenomena being studied (Cohen et al., 2011; Crotty, 1998; Saunders et al., 2012). *Epistemology,* on the other hand, studies knowledge - its nature, scope, possibility and broad spectrum (Crotty, 1998). Epistemology answers the question: *How do we recognise the reality we desire to know?* Hence, Saunders et al. (2012, p. 132) explain the relevance of epistemology stating that "it deals with the philosophical underpinning for deciding varied possible and acceptable knowledge". Thus, epistemology creates sets of rules for knowing the acceptable knowledge by locating boundaries and putting relevant mechanisms against those boundaries (Scott and Usher, 2011). Practically, this means that only some sets of knowledge are considered valid while others are rejected.

As put forward by Mertens (2010), the researcher's philosophical worldview has an impact on all decision made in the research process, including methods. A philosophical assumption is a paradigm, or worldview, shared by members of a research community (Given, 2008; Mertens, 2010; Willis and Jost, 2007). There are varieties of philosophical paradigms in literature, but those commonly referred to are found in the outer layer of the *research onion* namely: positivism, realism, interpretivism and pragmatism. The following discussion of the philosophies elucidates the reasons for the researcher's choice of *interpretivism* as guiding philosophy for this study.

- ***Positivism***

Positivism is the paradigm favoured by the natural sciences. It is an epistemological position which establishes facts about the world and it is only through scientific methods that these facts can be revealed (Saunders et al., 2012; Scott and Usher, 2011). Positivism's first assumption is the belief that both the social *and* natural world can be studied in the same way. As such, the explanation of methods employed in a value-free social world and the natural world are provided by breaking a complex problem into smaller parts (Lodico, Spaulding, & Voegtle, 2011; Mertens, 2010). Secondly, facts are objective but a clear difference must, however, be made between *objects* (the world) and *subjects* (knower) (Scott and Usher, 2011). In addition, the collection and analysing of data refer to the observation of reality as well as a search for regularities and causal relationships which can be generalised (Cohen, Manion, & Morrison, 2007; Saunders et al., 2012). Positivism is not the best choice for this research work as the present study is aimed at understanding the subjective meaning of participants regarding the phenomena as lived in the context.

- ***Realism***

Realism shares the philosophical stance of the natural sciences to the development of knowledge. With realism, reality exists independently of the mind and its socio-cultural practices (Scott and Morrison, 2005). The authors add that realism centres on the relationship which exists between the world and the way reality itself is described. To the realist, the world existing around us is real, yet it does not align, at any point in time, with our hopes and desires. Therefore, for objectivity to be maintained, the investigator must not interfere with the participants being studied (Lodico et al., 2011). Realism contradicts the perspective from which this study was conducted. The researcher holds the stance that reality exists in the mind and hence the knower must interact with the participants being studied.

- ***Pragmatism***

Pragmatism arises from situations and consequences that call for a solution to problems by focusing on the research problem and using every means available to comprehend the research problem (Creswell, 2014). A major tenet of this philosophical world view is that there are many ways of interpreting the world and conducting research (Lodico et al., 2011; Mertens, 2010). The pragmatists sidestep the endless argument between *truth* and *reality* whilst holding on to the philosophical belief that there are singular *and* multiple realities in empirical inquiry with the goal to solve problems in the real world (Creswell, 2014; Feilzer, 2010; Mertens, 2010; Schunk, 2014). Pragmatism also tends to place more emphasis on the research *questions* than the *methods* used to address them (Creswell, 2014; Feilzer, 2010; Onwuegbuzie and Leech, 2005).

- ***Interpretivism***

Interpretivism has its origin as hermeneutics, a theory of practice and meaning (Nieuwenhuis, 2014b). Interpretivists refute the importance positivists place on the objectivist way of ascertaining truth and, as such, found a place for subjectivism (Scott and Usher, 2011) while remaining in the scientific tradition and maintaining the objectivity of research. Sefotho (2015) noted that the move towards adopting multiple orientations, or worldviews to research, originated from interpretivism. Interpretivists believe in social interaction as the basis for obtaining knowledge (O'donoghue, 2007). Lodico et al. (2011) added that social phenomena must be studied from a context-specific perspective. Therefore, interpretivism accepts as truth that researchers seek to understand the world in which they reside (Creswell, 2014; Lodico et al., 2011). They further hold that knowledge can only be discovered by becoming immersed in the research to understand differences between humans. The researcher then makes sense from the meanings of individuals by

generating a theory (Creswell, 2012). This description typically supports the researcher's worldview. Therefore, interpretivism is suitable for this study. Table 4.1 summarises the research philosophies with their epistemological and ontological assumptions.

Table 4.1: Summary of philosophies

Philosophies	Ontology	Epistemology	Axiology
Positivism	Objective and external; social actors are independent.	Knowledge is obtained through observable phenomena. Uses generalisations, reduction and causality of phenomena.	Value-free. Researcher maintains an objective stance.
Realism	Objective; reality exists distinctly from the human mind.	Acceptable knowledge is known through observable phenomena which are explained in context.	Value-laden. Researcher's biases, cultural world and upbringing influence the study.
Interpretivism	Reality is multiple and socially constructed; it may change.	Acceptable knowledge is known through subjective meaning which focuses on the details of a situation.	Value-bound. Researcher maintains a subjective stance and is part of what is studied.
Pragmatism	Reality is external and multiple; chosen to answer a research question.	Acceptable knowledge is obtained through observable phenomena and subjective meanings.	Value-laden. Researcher maintains both objective and subjective stance.

The researcher thus adopted the interpretive philosophy as it allowed her to understand, or make sense of, the ways in which others understand their social world by using her skills as a social being (O'donoghue, 2007). The interpretive perspective to research is based on the following assumptions:

- The *social world of humans can only be understood from the inside and not from the outside*. Hence, the participants' subjective experiences, shared meaning construction and interactions amongst them in visual and procedural programming can only be understood by the researcher who shares their frame of reference (Cohen et al., 2007; Nieuwenhuis, 2014b; Saunders et al., 2012).
- Interpretivists assume that *reality is subjective and not an objective means of dealing with people's experience*. Students are placed within a context-specific environment where the researcher can explain and interpret social reality through their eyes (Cohen et al., 2011; Nieuwenhuis, 2014b).
- *The social world does not exist independent to our knowledge of it* (Grix, 2004; Nieuwenhuis, 2014b). The researcher's knowledge, understanding, beliefs or even

background knowledge constantly influence understanding of the phenomena being studied.

- *The human mind is the basis of meaning* (Cohen et al., 2007; Crotty, 1998; Nieuwenhuis, 2014b).Through an in-depth exploration of students' comprehension skills in procedural programming, the researcher developed a sense of understanding based upon the meaning which students bring to programming and the social context (Nieuwenhuis, 2014b).
- *There are multiple realities to the same phenomenon and it is possible that these realities change over time and place* (Crotty, 1998; Nieuwenhuis, 2014b). The reality, however, is a concurrence created by co-constructors (Pring, 2000). Simply put, knowledge can be culturally derived and historically situated.

Taking these assumptions into consideration the researcher can, therefore, study the programming skills of first year college students, from visual to procedural, and analyse and establish shared meanings of students on the phenomena from their own perspective.

4.2.2 Approaches

Research approach helps in making an informed decision about the use of theory in designing a research study (Saunders et al., 2012). The three forms of reasoning common to research are: deduction, induction and abduction.

- ***Deduction***

Deductive reasoning makes use of a top-down approach to research (Lodico et al., 2011). It involves commencing the research study with the development of a theory and then putting this theory to rigorous testing in order to verify the outcome (Saunders et al., 2012; Scott and Morrison, 2005; Scott and Usher, 2011). Its features can be described as: hypothesis-driven, follows a structured methodology, uses quantitative data, explains the causal relationship between concepts and variables and uses data reduction and generalisation of results (Lodico et al., 2011; Saunders et al., 2012; Scott and Morrison, 2005; Yin, 2011).

- ***Induction***

With inductive reasoning, a relationship is established between theory and data (Fox, 2008; Saunders et al., 2012) collected through direct observation and empirical evidence (Grix, 2004). This mode of reasoning usually conflicts with deductive, where sense is made of data and situated within a theory. In inductive reasoning, the inference is made when there

is a specific case and this inference entails generalisation to a wider context (Grix, 2004; Heit and Rotello, 2010; Hyde, 2000) as opposed to its deductive counterpart. Inductive reasoning is used in interpretive research; hence it is suitability for this study.

This research work aims to generate a theory from the findings of the research and test a theory. Therefore, both deductive and inductive reasoning guided the approach adopted by the researcher in the research study. Data collected from the participants' subjective meanings on visual and procedural programming generated themes which were used in theory generation and building.

4.2.3 Choices

Choices, as defined by Saunders et al. (2012), are wide-ranging approaches to answering a research question. A few choices are common in educational research and it is necessary to explore their characteristics, as reflected in the *research onion* (*q.v.* Figure 4.4). The three types of choices are explored in the following subsections.

- ***Mono-method***

The mono-method requires single data collection and favours either a quantitative, or qualitative, analytical procedure to answer a research question (Saunders et al., 2012). Different data collection methods are used in this research study and thus the mono-method is not suitable.

- ***Mixed methods***

Mixed methods research, as defined by Creswell (2014), is an approach to an inquiry involving the collection of both quantitative and qualitative data, integrating the two forms of data and using distinct designs that may involve philosophical assumptions and theoretical frameworks. Mixed methods use quantitative and qualitative phases in the design, analysis and interpretation of an entire research study (Johnson and Onwuegbuzie, 2004; Mertens, 2010). This study does *not* attempt to use the mixed method since it is purely a qualitative research work. Although some performance data, such as project scores and test scores may be quantitative, they simply support the findings.

- ***Multi-methods***

Unlike mono-method, multi-methods methodology involves using more than one data collection method within either quantitative, or qualitative, analytical procedure (Geelan, 2007; Saunders et al., 2012). Multi-methods can be further divided into: multi-method quantitative and multi-method qualitative (Saunders et al., 2012). In *multi-method quantitative,* various quantitative data such as structured interviews and observations can be collected within a study using quantitative statistical procedures which do not form the basis of this study. *Multi-method qualitative* combines a series of qualitative data such as interviews, documents and observation data collection methods within a single study qualitative procedure (Saunders et al., 2012). To explore first year students' programming skills, from visual to procedural programming, the researcher needs to understand, observe and assess students' learning and describe and interpret their experiences. Such processes are not dominated by objective but by subjective facts. In effect, the researcher's choice is multi-method qualitative. To gather necessary data within a multi-method qualitative methodological choice, *hermeneutic phenomenology* can be best used. That said, it is necessary that the researcher acquaint herself with the philosophical thinking on which hermeneutic phenomenology was founded.

4.2.3.1 Hermeneutic phenomenology

Hermeneutic phenomenology aims to understand the nature of experience through the description and interpretation of human experiences (Tan, Wilson, & Olver, 2009). Hermeneutic phenomenology originated from a phenomenological tradition conceptualised by Edmund Husserl (1859 - 1938). The Husserlian phenomenology centred on identifying the essence of the phenomenon through *epoche* or bracketing and phenomenological reduction, thus experience is transcended to discover reality (Langdridge, 2007). Bracketing means renouncing one's preconceived idea/s about the object under investigation (*ibid*). Martin Heidegger (1889 - 1976) made a sharp departure from the Husserlian phenomenology and rejected the idea of bracketing. He suggested the interpretative description of the phenomenon under investigation (Kafle, 2013). Heidegger holds that one cannot bracket off the essences of a phenomenon as it is seen and identified (Langdridge, 2007). In addition, Smith, Flowers, and Larkin (2009) described language use and the interpretation of researchers' meaning making from phenomena as central to Heideggerian phenomenology. Following Heidegger are two other significant proponents namely Hans Georg Gadamer (1900 - 2002) and Paul Ricoeur (1913 - 2005). Methodologically, hermeneutic phenomenology has no defined set of rules (Kafle, 2013; Laverty, 2003) but the work of Heidegger (1962) in the areas of method and interpretation was developed

further by Ricoeur (1981) and Gadamer (1997) in order to address the difficulty. Gadamer (1997), while contributing to Heidegger's work on the role of language, stressed that language informs both our understanding and interpretation of the participants' experiences. Ricoeur's work centred on interpreting the text, and through this he developed an elaborate theory of interpretation (Langdridge, 2007). Hermeneutic phenomenology can be used within a research study as either a theoretical perspective or a methodology on which the method used in a particular study is based (Crotty, 1998). It is used as a methodology in this research study.

There is a strong relationship between phenomenology and hermeneutic phenomenology. Nevertheless, differences also exist. Laverty (2003) and Creswell (2007) made a clear case on the differences. Hermeneutic phenomenology is oriented towards lived experience (phenomenology), as well as interpreting and understanding the *texts* of life (hermeneutics). The methodology of phenomenology attempts to unfold meanings as they appear in everyday experience by asking: *What is the experience like?* (Laverty, 2003). Schdmit (2006) added that in Husserl's phenomenology, a careful description of experience without judging the meaning of experience, is involved. *Hermeneutic phenomenology* concerns human experience as it is lived (Laverty, 2003). The focus, however, is to elucidate the important aspects of the *taken for granted* human experience with the aim of generating meaning through understanding. This research is a hermeneutic phenomenological study. Both the assumptions of phenomenology and hermeneutics are combined and then used to guide the research study methodologically.

In the light of the above, the method of inquiry is phenomenological because the study aims to understand the *lived* experiences of first year college students, from visual to procedural programming, as phenomena. It is hermeneutic because it presents the understanding of the lived experiences of participants through interpretations that are varied and complex (Van Manen, 1991).The approach helped the researcher to understand participants' lived experiences in their procedural and visual programming classrooms. The researcher's understanding of the perceived participants' experiences emerged through her interpretations and interactions with them. The multiple ways of interpreting the participants' experiences, in order to gain meanings and construct realities, are recognised by the researcher (Van Manen, 2002).

Heidegger, which is seen as the father of hermeneutics, stressed that one must understand the *situated meaning* of being before *knowledge* of being is discussed (Schdmit, 2006). Heidegger believed that prior understanding of a person's background is necessary in order

to understand the lived experience. The background knowledge cannot be put aside since it is believed to be with us, in the world (Schdmit, 2006). The researcher ascertained this key hermeneutic phenomenological assumption by situating herself, and the participants, within the context of the study and explicating their individual characteristics before the study commenced.

Using hermeneutic phenomenology within an interpretive paradigm, the reality was seen as an individual subjective construct reliant on different situations (Kafle, 2013). The reality was obtained by exposing the participants to series of activities such class-works, assignments, tests, group projects and presentations on visual and procedural programming. The researcher listened to a group of participants regarding their perceived experiences in order to reveal their reality, through interpretation of their experience, by using their own verbatim words from interviews (Kafle, 2013). Ontologically, multiple subjective views of reality were presented by the participants' interpretations of their experience, the researcher's interpretation and the audience's interpretations of the research study (Van Manen, 2002).

It was important to build a relationship between the researcher and the participants in order to gain an understanding of the students' lived experiences in the programming classroom (Schdmit, 2006). Within the context of a relationship, observation of the participants in the programming classroom and interaction in an interview deepened the researcher's understanding of participants' experiences (Schdmit, 2006). The researcher acted as a participant observer, thus the distance between the researcher and participants was minimised (Saunders et al., 2012). Therefore, the researcher encouraged a positive relationship through rich discussion, both amongst the participants and between the researcher and participants. The researcher thus gained an understanding of the different orientations participants brought to the programming classroom.

Relationship building between the researcher and participants has implications for the role of values and biases in the study. Firstly, the voices of the participants were necessary in order to ascertain their lived experience in programming since it played a key role. The researcher was thus value-laden and brought her own biases and assumptions into the study. She remained conscious, through self-reflection, of how her biases could colour the findings of the study (Laverty, 2003).

As stated by Heidegger, meaning resides in the world and as we are understood by it. We understand the world in terms of our own background and experiences. Interpretation of the historical meaning of experience, its development and overall effect on the individual and

social levels is critical to the process of understanding (Schdmit, 2006). One way in which the researcher addressed the use of language in the research study was by being cognisant of the programming language used by the participants to describe their experiences during the classroom activities, written text and in interviews. This was done to explore their origins and underlying meanings. The importance of language in shaping our experience and interpretations was stressed by Gadamer (1997). The researcher also integrated the use of language by focusing on the process of writing the text while probing its origin to discover the background meaning that might clarify understanding.

Methodologically, Laverty (2003) itemised the guiding principles for using hermeneutic phenomenology. A purposive sampling of participants with rich information, multiple tools such as interviews, observations and other data gathering methods were suggested (Kafle, 2013; Schdmit, 2006). As explained, the researcher engaged in regular reflection on her experiences during the teaching and learning process. Also, reflections were made regarding the information gathered from participants' lived experiences in procedural and visual programming. Participants were selected based on: different learning styles, having lived the procedural programming experience, having unique stories and a readiness to discuss their experiences. The researcher kept a journal to record the process of reflection and interpretation.

Hermeneutic phenomenology allowed the researcher to carry out the before mentioned within the qualitative study based on its potential to deduce or interpret the *texts* of the lived experience. Through this, a better understanding as to the nature and quality of the phenomena, as they present themselves, were gained. All description entails a form of interpretation as language and semiotics (signs) provide the means of obtaining data. In using hermeneutic phenomenology, it is important to foster an awareness of: the research question, building trust and emphatic understanding with participants as well the importance of imaginative and ongoing reflections. The schematic representation of the study's choice of methodology is represented in Figure 4.5. The way in which hermeneutic phenomenology was used to gather data on the lived experiences of the participants and phenomena are explained further in the data collection process (section 4.3) and data analysis process (section 4.4).

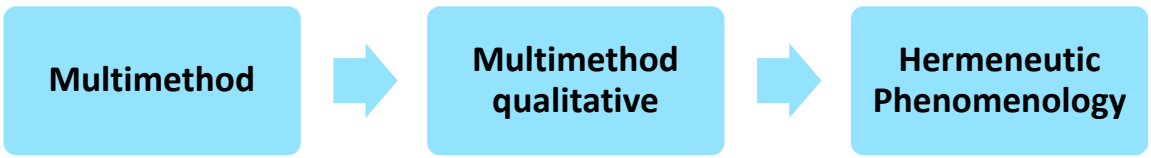


Figure 4.5: Research choice for the study

4.2.4 Research strategies

Research strategies are designs generated from the underlying philosophy for a research study. As such, they inform the methods used in gathering and analysing data (Nieuwenhuis, 2014b). The research strategies, listed in the *onion,* are now explained along with the reasons as to why they were *included* or *excluded* in the study. In *experimental research strategy*, conditions which determine events of interest are manipulated by researchers through the introduction of intervention and measurement of the difference made (Cohen et al., 2011). This involves a change in the value of an independent variable while observing its effect on the dependent variable. The experiment does not support the assumption of a qualitative research; hence it does not suit this study. The study aimed to establish the *subjective* and not *objective* meaning of lived experiences.

Archival research strategy studies secondary data in the form of archived administrative records or documents of an organisation for research purpose (Saunders et al., 2012). This strategy answers questions about past events and changes over time. Such data may be exploratory, descriptive or explanatory. This study did not focus on past events, or documentary evidence, hence this approach was not deemed relevant. A *case study* strategy investigates a phenomenon within a context through the study of one or more cases (Given, 2008; Saunders et al., 2012). The phenomenon being researched could be a social group, a process, an event, a person or an institution (Willis and Jost, 2007). Though a qualitative strategy, the researcher did not intend to study a case/s and thus the approach was deemed inappropriate for the study.

Ethnography studies the behaviours of groups of people within a context using the case as an illustration. Ethnography, therefore, is a qualitative research strategy which studies a culture-sharing group with the aim of understanding the meaning of behaviour, language and forms of interaction that exist among them from an insider's point of view (Creswell, 2007; Given, 2008). This necessitates that the researcher live amongst the cultural group during data collection. He/she should participate in their daily life, interact and carefully observe and record details regarding the group's behaviour, lifestyle, language and interactions (Saunders et al., 2012; Tedlock, 2005). This research study does *not* intend to study a cultural group, with a shared meaning, or learn more regarding values, behaviours, beliefs and language. In addition, the researcher does *not* intend to live within the context. *Grounded theory* is a qualitative research strategy that focuses on generating theory (Patton, 2002). While the other approaches describe a theoretical content, or an aspect of the social world, grounded theory focuses on a process of discovering or generating a

theory (Creswell, 2007; Patton, 2002). Grounded theory is not the best strategy for this study, because it does not describe an aspect of the social world, as lived by the participants, but rather focuses on a process of constantly comparing collected data to generate a theory.

Narrative research is a type of qualitative design that allows the researcher to engage in the intimate study of participants' experiences over time in a sequential order by collecting and analysing them as complete stories (Creswell, 2007; Given, 2008; Saunders et al., 2012). Narrative inquiry places importance on building *relational engagement* between the researcher and the participant telling the story. The aim of this study is *not* to investigate and give an account of the intimate experiences of participants, rather their *lived* experiences, hence this approach is not applicable to the study. *Survey research* is used to gather information regarding people's opinions, perceptions and attitudes, towards planning and evaluating programs. The survey method has the potential to generalise large populations and it yields a high measure of reliability and validity if the questionnaires are properly constructed (Efron and Ravid (2013). The *survey design* was selected for the quantitative component of this study, because of its strength in obtaining information from a range of cases in the sample population and to centre exact characteristics under consideration (Maree and Pietersen, 2007). After looking at all the research strategies, it seems that Action research was the most appropriate. Action research is thus the major research strategy adopted in this study.

- ***Action research***

Action research (AR) is defined as a systematic procedure and self-critical inquiry that educators use to gather information on the practice of their teaching and student learning, while making practical changes based on findings (Daniel, 2010; Hendricks, 2013; Koshy, 2009; Mills, 2011; Stringer, 2007).To put it differently, Cohen et al. (2011) see AR as a combination of *action* and *research*. Piggot-Irvine (2015, p. 32), building on the latter, described AR as the "popular developmental research methodology which has both data collection (research) and change (action) elements". Thus, action research is a strategy that involves the systematic study of problems by developing theories to effect changes. The goal of AR is to combine theory and practice in such a way that the supposed constant failure of research to impact and improve practice is overcome (Cohen et al., 2011). In effect, theory contributes to education and teaching and educational practice becoming more reflective (McNiff and Whitehead, 2006b). AR is best suitable for this study. The researcher's assumptions for using the AR strategy are discussed in the following section.

4.2.4.1 Action research strategy: ontological and epistemological assumptions

In considering action research, McNiff and Whitehead (2010) posited that the researcher must keep in mind the ontological and epistemological assumptions towards the generation of a theory when thinking about research. The assumptions, and how they guided the study, are detailed below.

- **A commitment to educational improvement**

Educational improvement is all about improving learning and such an improvement is influenced by the context in which the researcher belongs. While reflecting on this, the researcher entertained some questions including: W*hat aspect of practice do I wish to investigate?* and *Why do I want to investigate this practice?* In order to develop her own living educational theory, the researcher decided to improve her practice in the context where she belonged by improving first year college students' programming skills and so awaken an interest in programming. The researcher wanted her students to see the connection between the use of concepts in visual and procedural programming and so become conscious of how they support each other in learning. During teaching, the researcher wished to encourage her students to use the language of programming to construct meaning through different methods. Such methods include: (i) working cooperatively to socially construct meanings on programming, (ii) engaging in critical thinking, (iii) doing presentations on programming problems and (iv) writing down that which they have learnt in group projects. It is hoped that through this, students would become lifelong learners as they grasp the connection between visual programming and procedural programming. Secondly, the researcher wanted to develop a teaching and learning framework that might assist other departmental staff and colleges of education to improve teaching and learning in their respective programming classrooms. Therefore, the researcher viewed this improvement as a continuing process which would improve over time. Her capacity, as a lecturer in the context, facilitated the fact that she could influence the future of her students by acting in the "now" (McNiff and Whitehead, 2010).

- **A special type of research question asked with educational intent**

This type of question, asked with educational intent, enables the researcher to pose the question: *How do I improve my practice?* The researcher's intention was to improve her professional practice to support students' understanding of programming by introducing visual programming through a holistic learning. This is the first time she is motivated to research and improve her practice. Therefore, the researcher enters the research field with an open mind such that ideas are held in the interim (McNiff and Jack, 2010). The

researcher did not anticipate concrete solutions from the participants since meanings created are tentative and open to further modification. In addition, she aimed to explore her aptitude for learning, since the new learning she gained would bring about the improved action.

- **Putting the 'I' at the centre of research**

In any action research study, the researcher is at the centre of the research. The report is a description of the researcher's personal account in which 'I' in relation with other 'I's' must be used during the explanation of the research story. The researcher is the ontological point of research as she studies herself and her own practice. She is thus studying her life. She brings value to her own research as well as her own personal way of doing things. She questions her own actions and the way in which she does things. Therefore, she takes responsibility for every action and decision made during the study and through doing this, she can influence both her learning and the learning of the students in the context. The students can influence their own learning of procedural programming and that of others in other contexts.

- **An informed, committed and intentional educational action**

AR starts with the need to do something which, in turn, leads to intent and then becomes active. With these three elements in place, the practice can be turned to praxis through the production of knowledge for personal and social use (McNiff & Whitehead, 2009). The researcher was consciously questioning her own motives about the study and critically treated her interpretations of findings while shelving judgements and being open to receive the views of others. The researcher is committed to the teaching of programming in a better way by constantly living according to her values, as discussed above. She thus ensured a constant check on what she was doing by always moving back and forth to the research questions to ascertain whether she was still committed to her values. The intentional action of the action research has been detailed in the research process (see section 4.2.4.2). The next section will discuss action research characteristics, as used in the study.

4.2.4.2 Action research characteristics

Action research differs from traditional research. There are some important characteristics of action research which precipitated the choice of this strategy as investigative lens from which this study is conducted. Headings adapted to the study follow. Action research is:

- **Concerned with making improvement/s in the context of the study**

The context is studied and not controlled so that the ways in which context influences the study can be understood. This is done by continuously feeding back the various sources of qualitative data collected into the action research cycle for the purpose of informing practice (Hendricks, 2013). With the difficulties first year college students experienced in programming over the years, the researcher needed to initiate a better practice which would hopefully promote change in the context of the study, by encouraging her students to learn visual programming and to be actively involved in the learning process through a holistic model of learning.

- **Improving practice**

An action researcher usually employs various and suitable interventions to collect and analyse data and solve educational problems (Daniel, 2010; McNiff and Whitehead, 2010). This is the reason why it focuses on change through solving a problem and making necessary improvement. Since action research is about improving practice, the outcome of the study would be used to generate a theory which would, in turn, inform the practice of programming teaching in Nigeria.

- **A collaborative process**

AR involves a partnership between researchers and participants or stakeholders. The researcher fostered this by acting as a facilitator of learning whereby opportunities that encouraged learning programming within a social constructivist and whole brain learning environment were created. Through this, participants were encouraged to accept full responsibility for their learning through exposing themselves to cooperative learning, pair programming and engaging in discussions during group presentations (Du Toit, 2010).

- **Improves learning and not behaviour**

Action research, as explained by McNiff and Whitehead (2010), studies individuals' learning in relation to others. Based on this premise, as practitioner, the researcher had both a personal and social aim (McNiff and Whitehead, 2010). The personal aim being to develop a deep understanding of the intervention and as an insider researcher to improve learning and behaviour in teaching practice. Secondly, the social aim was to use learning and understanding in this regard to improve students' learning; such that learning impacts on their behaviour towards programming.

- **Encourages facilitator to reflect on their practice**

Daniel (2010) stressed that action research is more concerned with reflection and employing practical ways to solve the problem under study. During the ongoing collection of the data, the researcher reflected on data collected at the end of every class in order to modify that which happened in the next class. The researcher constantly reflected on the practice during the final analysis of data.

- **Encourages testing of an innovative idea**

Action research studies a problematic situation, or innovative idea, in a systematic, continuous reflective process which is cyclical in nature with each cycle leading to the next and continually beginning anew (Ebersöhn, Eloff, & Ferreira, 2016; Hendricks, 2013; Pine, 2008). With this in mind, the researcher was interested in the action research model of Zuber-Skerritt (2001) which was used to initiate the study. The Visionary Action Research model (Du Toit, 2012), which combine other important stages, was used to substantiate her innovative ideas. These two models are discussed in the following section.

- **Action research contributes to social and cultural transformation**

Social and cultural transformation refers to action research practitioners having the capacity to influence the future through a collaborative effort. The study was not conducted in isolation but the researcher involved her students who served as participants for the study. Also, lecturers in the department served as non-participant observers. Therefore, the researcher was afforded the capacity to influence the future by bringing these participants together in the AR study.

4.2.4.3 Types of action research

The choice of action research will depend on features such as “research question, research purpose, the unit of analysis, time frame, access and relationship between partners, contextual variables and partner preferences” (Ebersöhn, Elof, & Ferreira, 2007, p. 126). Various scholars have categorised action research types. Table 3.5 presents the types *with* their meanings and reference sources. This research study aims at practitioner’s understanding and professional development in the context of the study, hence practical action research (Creswell, 2012; Mills, 2011). This type of action research is done *by* the teacher, and *for* teachers towards making a judgement on how to improve practice. The inherent characteristics of AR, as explained before, guided the study. Two cycles of the *Plan-Act-Observe-Reflect* (Kemmis, McTaggart, & Nixon, 2014; Zuber-Skerritt, 2001) spiral were adopted to collect extensive data on the study.

Table 3.5: Types of action research

Type	Author	Description
Collaborative action research	Hendricks, 2013; Pine, 2009	Coming together of multiple researchers from different institutions to solve an educational problem.
Classroom/ technical/ practical action research ***(Research study focus)***	Hendricks, 2013; Mertler, 2008; Mertens, 2010; Zuber-Skerrit, 1996; Kemmis & McTaggart, 2005, Creswell, 2012	Conducted by a teacher or team of teachers in their own classroom with the purpose of improving own professional practice. It is small-scale and centers on a specific problem.
Participatory action research	Mills, 2007 Kemmis & McTaggart 2005 Hendricks, 2013; Pine, 2009; Christ, 2010 Cohen, 2011	Teacher-researchers have authority over their everyday practices by collecting and interpreting data, developing an action plan and reflecting on practice. Shared ownership of the project. Analysis of community-based problems. Community action orientation. The social and collaborative process to investigate reality for a possible change. Doing research with people and communities democratically to bring about social justice for ordinary people.
Community-based action research	Stringer, 2009	Mutual efforts of all parties affected by the problem under study with the aim of providing solution collecting data, analyzing data and taking action.
Critical action/ emancipatory action research	Christ, 2010; Mills, 2007; Ebersohn et al. 2016; 2014; Stringer, 2007; Kemmis & McTaggart 2005	Democratic and participatory. Encourages the participation of stakeholders. Takes place in the context and socially responsive. Self-inquiry of taken for granted professional practice. Encourage self-reflection of practitioners.
Critical participatory action research	Kemmis & McTaggart, 2005; Kemmis, 2008; Kemmis, McTaggart & Nixon, 2014	Explores social realities with the aim of dealing with character, conduct and consequences for untoward practice by opening space for communication and reflection.

4.2.4.4 Action research models

AR involves systematic processes, or cycles, of which there are many models, some somehow similar. The models provide guidelines to inquiry, most of which being adaptations from the original *cycle of steps* of Lewin to AR (Kemmis, McTaggart & Nixon, 2014:18) The cycle was later popularised by Kemmis and McTaggart (1998a) as *spiral steps*. This spiral includes planning, acting and observing, reflecting, for Cycle 1 and repetition of the processes of Cycle 2. Various models were later designed but the different models of action research share general elements – identifying the problem, collecting data, analysing data and performing an action that leads to the next process in the cycle (Mills, 2011). Another model by Zuber-Skerritt (2001) comprises plan, act, observe, reflect and revise the plan, and so on. The diagrammatic representation of the action research spiral is presented in Figure 4.6. The diagram shows a continuous cycle, hence it involves planning the enquiry, implementing and observing action through innovation, doing a critical self-reflection as well as revising the plan.

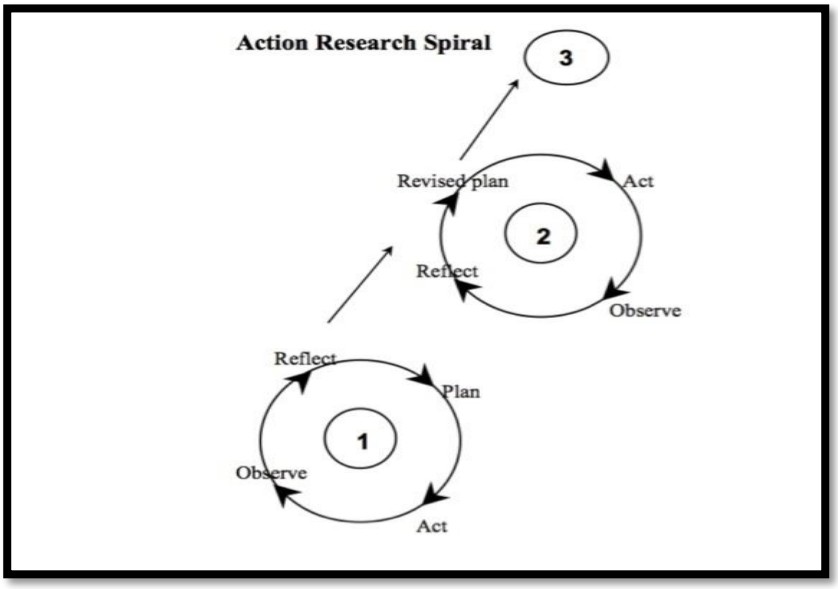


Figure 4.6: The action research spiral
Source: Zuber-Skerritt (2001, p. 15)

The dialectical action research spiral, designed by Mills (2011), contains four stages (see Figure 4.7) which work back and forth between data collection and focus and data collection and data interpretation. He stated that the spiral is meant to be used by teachers to study their teaching techniques.

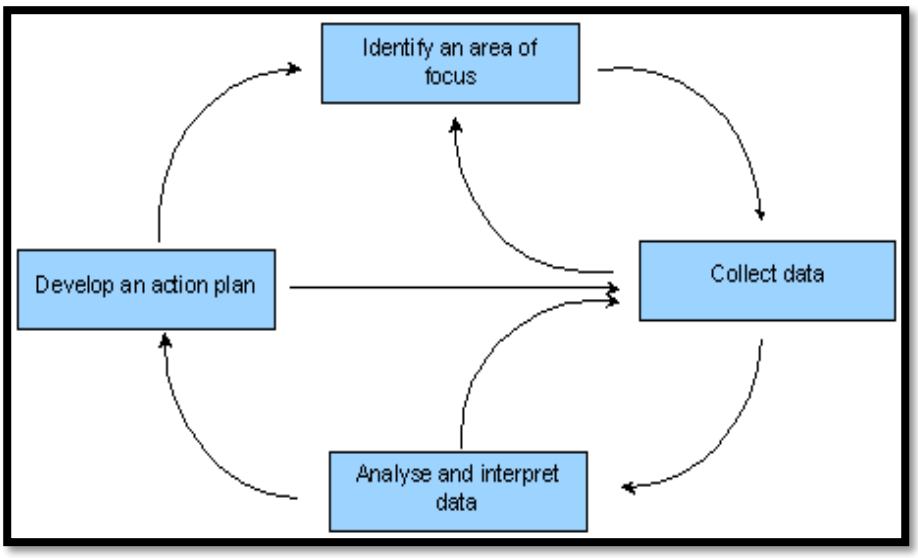


Figure 4.7: Dialectical action research spiral
Source: Mills (2011, p. 20)

The recent visionary model (see Figure 4.8), which was designed by Du Toit in 2009 (Du Toit, De Boer, Bothma, & Scheepers, 2010), presents an alternative to the gaps deficit found in McNiff and Whitehead (2010) and Zuber-Skerritt (1996) models. The authors described their model as asset-based, unlike the problem-based models supported by most action researchers. They believed it is better to experiment with new ideas rather than solving an existing problem. This model does not represent a neat cyclical process. It is messy, multidimensional and flexible, thus allowing the first cycle to occur and then enabling the researcher to start in the middle of visionary model cycle. This may lead to the discovery of another fact which the researcher will then reflect upon. In this way the model follows the following cycles, resembling other action research models.

Step 1: Planning for innovation/transformation

Step 2: Acting to innovate

Step 3: Observing the effects of the new action

Step 4: Reflecting in/on action

Step 5: Evaluating

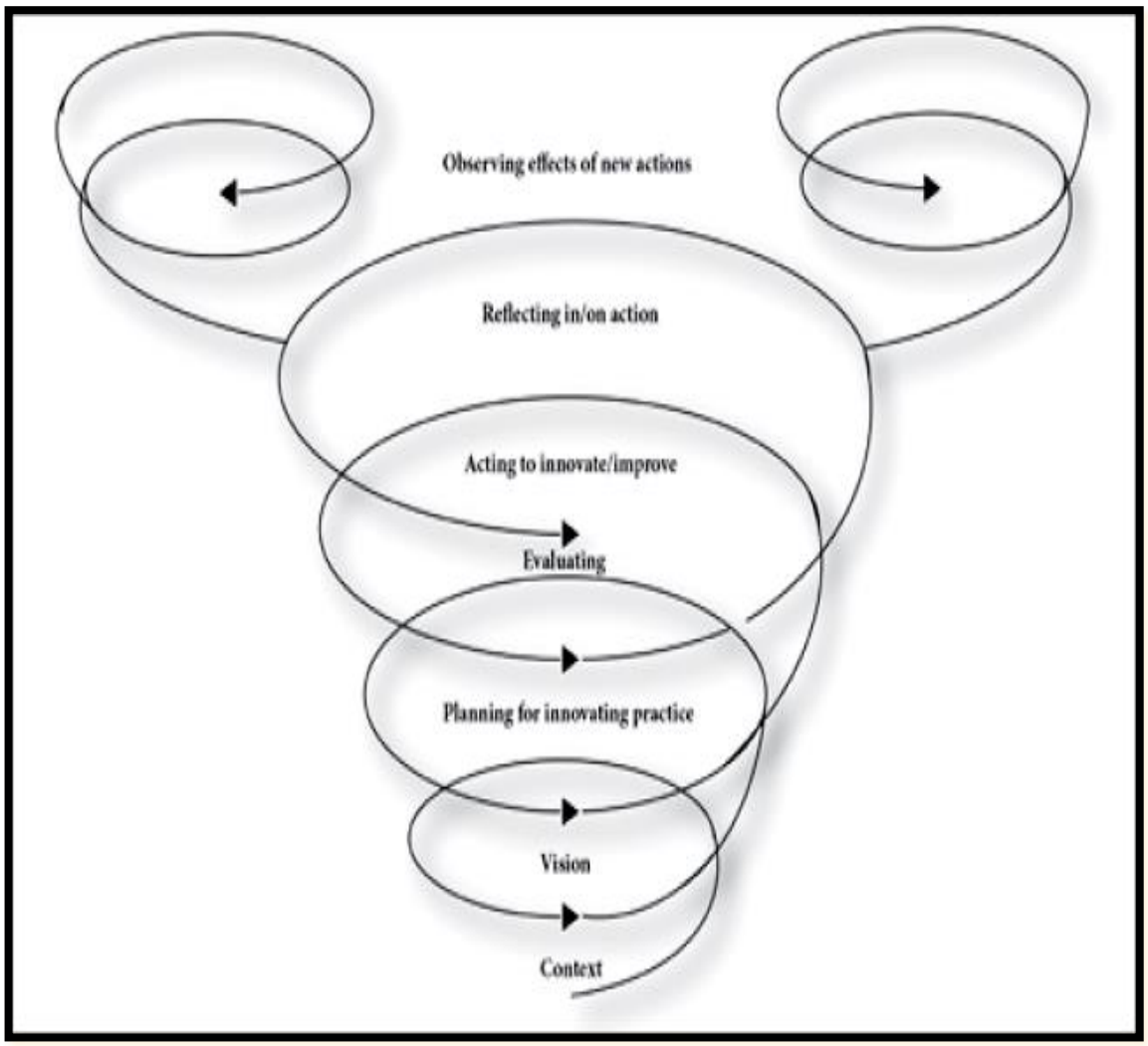


Figure 4.8: Visionary action research model
Source: Du Toit et al. (2010, p. 10)

The model which interested the researcher and which met the planned AR is the Zuber-Skerritt and Du Toit visionary action. The researcher integrated the ideas pertaining to *context* and *vision*, as found in Du Toit (2009) visionary model, with the Zuber-Skerritt action research spiral to help promote her professional learning and, in turn, facilitate the participants' improvement. The researcher also sought to solve existing problems by experimenting with new ideas. In making the model practical within the study, the researcher considered herself a *visionary practitioner* who wishes to improve "current practice" by "radically transforming it" (Du Toit, 2009, p. 14).Therefore, the researcher focused on her own assets and those of her students. These assets include generating her won learning approaches, and those of the participants, towards innovating practice or radically transforming it.

An innovating practice which the researcher wanted to investigate was the support of teaching procedural programming with the visual programming environment. This innovation was not solely based on a transformed view of facilitating learning, but also included holistic teaching, through the whole brain, as an essential part of teaching and learning. By transforming *educational practice* through *active participation* in the course of professional learning, the researcher considered herself a “transformational leader” who is open to further development (Du Toit, 2010, p. 14). *Transformational* is emphasised as it is the first time that the researcher is introducing the learning approaches and a holistic approach to the teaching of procedural programming in her classroom teaching. It was also the first time that such and innovative research project would be conducted in the college. The systematic application of AR models to plan for the research study is presented in sections 4.2.4.5 and 4.2.4.6.

4.2.4.5 The Action Plan

The action plan describes *how* the practical action research was conducted over two cycles and how it was situated within the research questions and context. Cycle 1 involved different activities which investigated all the research questions listed in Chapter 1. Reflections and lessons learnt from Cycle 1 resulted in a revised plan which was then implemented in Cycle 2 of the research. The diagram presented in Figure 4.9 and Figure 4.10 summarise the two cycles employed in the study. The action was planned with the knowledge that it might alter during the course of the actual study and was, as such, considered as a *guide* to the researcher’s thoughts as they pertained to the study (McNiff and Whitehead, 2006a).

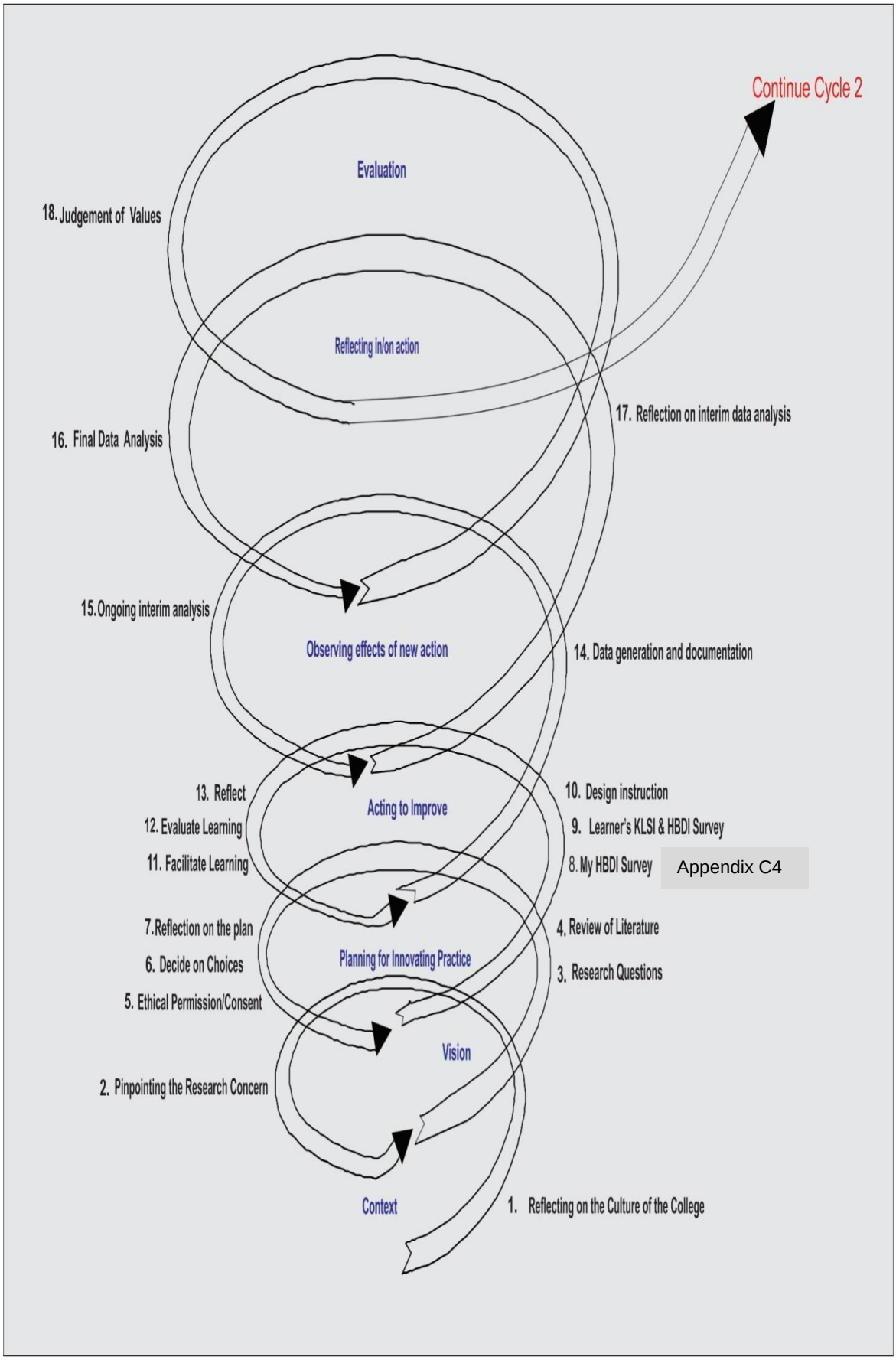


Figure 4.9: Cycle 1 action research plan

Practical action research: an explanation of AR Cycle 1

The ***context and vision*** in Cycle 1 commences with careful observation and the development of an understanding regarding the research context. It also pinpointed a focus for the study (Coghlan and Brannick, 2014). The authors called this stage *pre-understanding* and it involved having personal experience and full knowledge of the system in which the research is being conducted. The researcher attended the college as an undergraduate student in 1997 and was employed as a junior staff member three years (2002) after graduation, and as a lecturer in 2011 up until the time of conducting this research. As a former student and employee, she thus possesses valuable experience regarding the formal and informal cultures of the context although she does not claim that her knowledge is by any means complete. A detailed description of the context is presented in section 4.2.4.6. Her unique background knowledge and critical reflection of the context lead the researcher to the idea of improving her own professional learning. After discussing ideas with colleagues *inside* and *outside* the context, including her supervisor, a focus was identified and her research concern was formulated: "*How can I explore first year college students' programming skills, from visual to procedural, within a holistic environment?*"

Planning for innovation involves the consideration of possible investigations into the teaching situation, and possible potential improvements (Zuber-Skerritt, 2001). As a lecturer in programming education, the researcher explored her values so that she could contribute to educational theory (Coghlan and Brannick, 2014). She is committed to improving her own professional learning by improving the ways in which students develop programming skills through engagement with visual programming within a holistic environment and social context. These students can, in the long run, influence a wider context. The rationale for this study has already been explained in Chapter 1. The following section describes planning for the innovation by decomposing her values into a workable plan. This idea was first coined by Hendricks (2013) who stressed that the initial plan of the action research study involved four stages. The first step is to develop the *research question* (*q.v.* Chapter 1), secondly *literature* is researched and reviewed and best practices from literature are imported into the study (*q.v.* Chapters 2 and 3). Lastly, appropriate *choices* and *ethical permissions* are taken into consideration and decided upon in sections 4.2.3 and 4.6, respectively. The following explanations details *how* the plan will be acted upon and implemented.

Acting to innovate asks the question: *How do I improve my practice and help my students to learn with an educational intent?* The values held by the researcher influence the epistemological basis according to which data will be gathered and also serves as the

judgement by which practice will be held accountable (McNiff and Jack, 2010). Hence, the researcher is finally responsible for her own actions and thoughts and *not* for those of others. Participants' opinions will be welcomed to check her actions as the study progresses. The researcher will reflect upon how the *teaching and learning process framework* (*q.v.* Chapter 3) can be applied to the study while emerging observations will be noted. The researcher also liaised with an HBDI practitioner to complete an online *HBDI learning preference survey* (*q.v.* section 2.3.5.3) at the required cost. The resulting profile provided both baseline data and a point forward into the professional practice. The clarification supplied by the practitioner regarding the profile served as background knowledge to guiding the participants.

Knowledge is *not* fixed and there can be multiple answers to a single question (McNiff and Whitehead, 2006b). Participants were acquainted with both the KLSI (*q.v.* section 2.3.5.1) and HBDI learning approach instruments at the beginning of the semester. The reason being that the researcher needed to produce her own "living educational theory" (McNiff and Whitehead, 2006b, p. 38). The participants' profile results would then be considered to encourage flexibility in instructional planning where all quadrants would be catered for. In addition, care was taken not to plan instructions solely on the researcher's learning approach since it is possible that a diversity of learning approaches exist in the classroom. The learning approach result will serve as baseline data for the teaching of programming. Once some of the participants' learning approaches (KLSI) have been ascertained, a *constructive alignment* of courses (Appendix B 1) and (Appendix B 2) will be used to design instruction for the teaching of visual and procedural programming, respectively. Instructions will be facilitated by a *constructivist and whole brain teaching* and learning environment in which participants are arranged in groups of four, based on commonalities, or variation in their learning style. The researcher holds that this arrangement would strengthen collaboration amongst participants in each group as well as between the participants and the researcher.

The underlying assumptions of constructivism (*q.v.* section 3.2.2) and whole brain teaching and learning (*q.v.* section 3.2.3) will be adopted to accommodate teaching flexibility and afford students the opportunity to develop their less preferred learning approaches. During the thirteen weeks of the semester, the participants will first be introduced to visual programming followed by procedural programming. Individual class work, assignments, group work, project work and pair programming will be done cooperatively within groups. Participants will also receive scaffolding where needed and will be exposed to authentic assessments at different stages of the teaching sessions. After facilitating learning,

participants' individual conceptual understandings of visual and procedural programming concepts will be evaluated through classroom test. Three different types of tests (an individual concepts test, an interim test and final tests) will be done near to the end of the learning process. However, at this stage the researcher is aware that some unanticipated factors may arise which will urge reflection of the initial plan, allowing for new changes to be made.

To **observe effects of the new action**, data will be generated and documented while the researcher will be *open-eyed* and *open-minded*, actively involved in the classroom. Through engagement with the participants before, during and after the teaching and learning process, data generation will be possible. Therefore, observations as to her own teaching and professional learning by the researcher as well as students' group discussions and learning conceptions will be gathered. The various ways in which data will be collected are explained in detail in section 4.3.

Reflection in/on the teaching situation is a *meaning making process* that will facilitate thoughts regarding data and its interpretation as well as recommend future actions and implications for real-life application. Based on experiences during the action and observation stages, the researcher will critically reflect upon different sources of data while establishing a link between teaching practice and the participants' learning (Pelton, 2010). This can only be possible if one engages, in what Schön (1983, p. 55) calls "*reflection-in-action"* and "*reflection on action".* These reflections happen during data collection of surprises from data; and after all, data are collected respectively. Therefore, reflections will be done on the data collected by making teaching decisions that will support students' learning (Pelton, 2010) as well as the researcher's questioning of her decisions in order to think differently. The researcher will also engage in an *ongoing interim analysis* as data collection unfolds (Huberman & Mills, 1994) and thus reflect on data to check *how* data informs and captures the instruction. This will be done at the end of each lesson through watching a video which captures the teaching and interactions of the class. Issues noted will guide subsequent lesson instruction. Also, students will be given a reflective lesson log wherein they can reflect on the teaching and topics taught. Their reflections will further guide the researcher to plan subsequent instructions. Lastly, a *final data analysis* and further reflections will be made (see section 5.3). These reflections will guide the interpretation of data in corroboration with the literature (see section 5.4), however, no conclusion will be made. Since the researcher is a participant observer, it is possible that biased conclusions about the result could be made. Therefore, the concern for action will be questioned and

the outcomes evaluated. This process will be facilitated by critical friends who will assist the researcher in the critiquing of interpretations.

The researcher will **evaluate** the research work by judging her own practices, based on the values guiding the research, and by checking values taken for granted to see how these values have influenced her practice and professional thoughts. The outcome of Cycle 1 will inform decisions for teaching in Cycle 2 as presented in Figure 4.10.

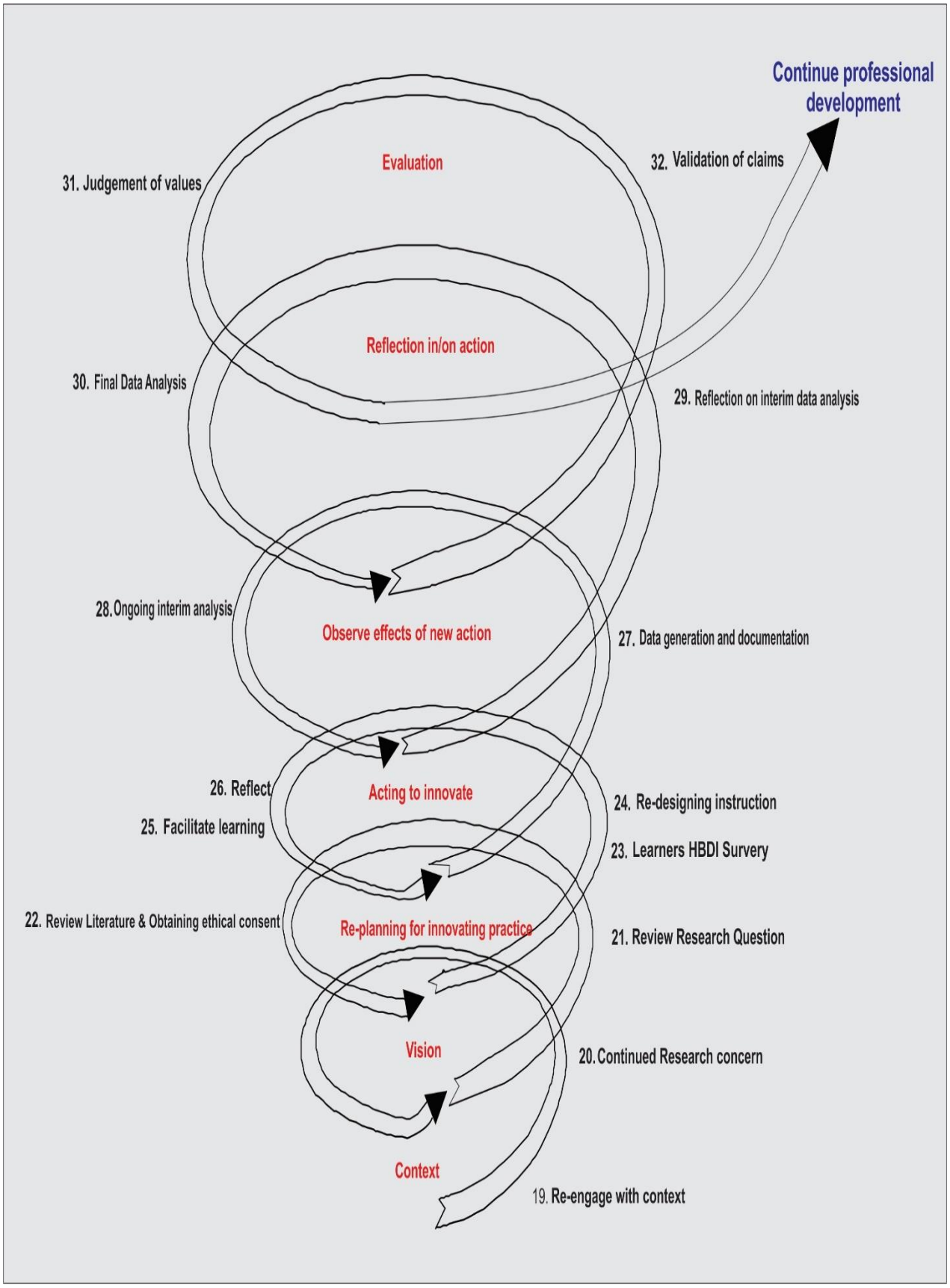


Figure 4.10: Cycle 2 action research plan

Practical action research: an explanation of Cycle 2

The outcome of the action plan in Cycle 1 will inform **re-planning for innovation** in Cycle 2 (see Figure 4.10). While committing herself to improving her professional learning by improving participants within the social context. The researcher will review the research questions and possible literature that can help to further utilise new practices. A new set of participants will be worked with in this cycle and ethical consent will be obtained from them.

Acting to innovate: participants will fill the HBDI learning preference survey at the beginning of the semester. The participants' profile results will then be considered to encourage flexibility in instructional planning and to ensure that all quadrants are catered for. In addition, the researcher will take care not to plan instructions solely on her own learning preference since participants' learning approaches may vary from her own. This will serve as baseline data for the teaching of programming. Consequently, the participants will be facilitated within a constructivist and whole brain teaching and learning environment. All the processes in the acting phase of Cycle 2 will be executed in line with the processes of Cycle 1, except where changes, or improvements, are deemed necessary.

To **observe the effects of new action in Cycle 2**, data will be gathered and documented in line with the processes undertaken in Cycle 1.

Reflecting in/on teaching situation will enable the researcher to analyse and do self-critical reflection on data and teaching experience. The reflection will occur before, during and after all data are collected, as discussed in Cycle 1. The data gathered will enable the researcher to decide what worked and what did not work in the classroom and, consequently, which practices should be retained for future use. In order to limit bias about the data, the researcher will therefore query concerns for action and evaluate outcomes by involving critical friends who will assist in critiquing interpretations.

Evaluating

The researcher will discuss how rigour, as discussed by Munn-Giddings and Winter (2013, p. 53), is demonstrated in line with "reflexive and dialectical critique". These claims will be validated with evidence from practice and offered for public authentication, including unbiased evaluations from colleagues and students.

4.2.4.6 The context

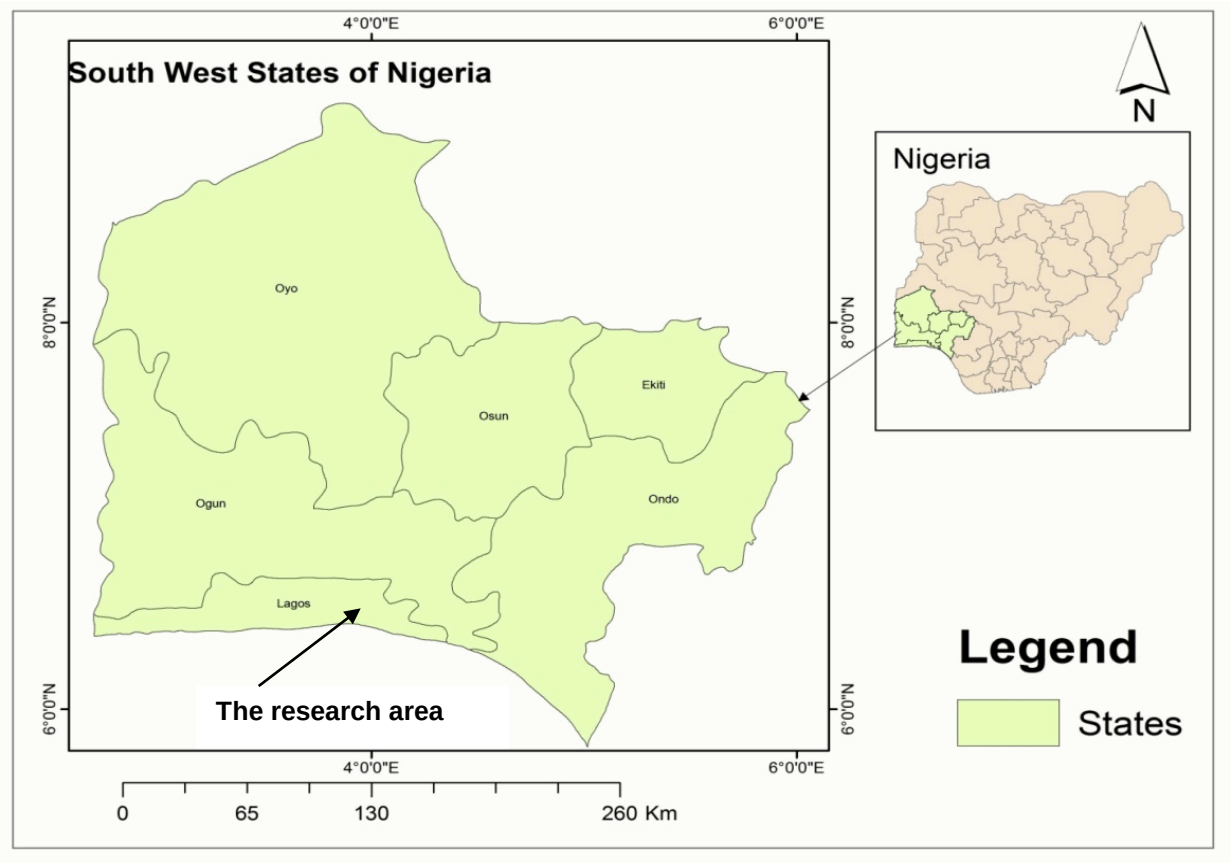


Figure 4.11: The context of the study

Context describes the setting of the study, the participants and the researcher. The context of this study is Lagos state (see Figure 4.11), a south-western part of Nigeria. On the west, Lagos is surrounded by the Republic of Benin and to the north and east, Ogun state. It also borders the Atlantic Ocean to the south. Lagos state is divided into five divisions namely: Lagos Island, Lagos mainland, Ibeju-Lekki, Badagry and Epe. The divisions are serviced by two Lagos state and one federal government colleges of education, one of which being the setting of this study.

On 1 December 1994, Lagos State Government decided to establish a college of education strictly dedicated to delivering pre-service training to and certification of graduates for the primary school system. Teachers who graduated from these schools, service the schools in the community and greater environment. The school contains students and staff from the community and other parts of Nigeria. Since its inception in May 1995, the college has run 19 academic programs distributed across six schools namely: School of Education, Languages, Primary Education studies, Vocational Education, Sciences and Arts as well as Social Sciences. All academic programs are fully accredited. The computer science

department has nine lecturers who teach various computer science courses. Three technologists serve as supporting staff.

The setting is rural and most of the residents fall within a low to middle income class grouping. The community is surrounded by the Atlantic Ocean, one of the major reasons why most residents embrace fishing. Other residents work as teachers, artisans and entrepreneurs, to name but a few. The culture and tradition of the community focuses on the *Ebi festival* which originated in the 13th century. This festival, during which the community worships their ancestors and deities, is usually observed in February and March. The *oro (*meaning 'no one must see') *festival* kick-starts the celebrations. The oro festival impacts on this study as it is mostly performed at night and, sometimes, in the late afternoon. Before the 'oro' is however done, the residents of the community (especially the women) are warned to stay in-doors so that they do not witness the oro's activities. The study's data collection period coincided with the oro festival and this impacted negatively on the study. The lecture times fell within the oro period and participants were afraid of being near the college. The researcher managed this by rescheduling classes before the oro activities commenced.

A classroom and computer laboratory are the milieus in which the action research took place (Phillips and Kevin, 2014). They are equipped with computer systems (both functional and non-functional) to facilitate students' practical work. However, the laboratory is used for other purposes as well such as computation of results. This necessitates the frequent movement of staff and students in and out of the laboratory which had a negative impact on the action research study. The researcher resolved this by closing the laboratory while a class was in progress and hanging a tag - class in progress - on the door. This helped to curb traffic in the laboratory. Classes were consequently not disturbed.

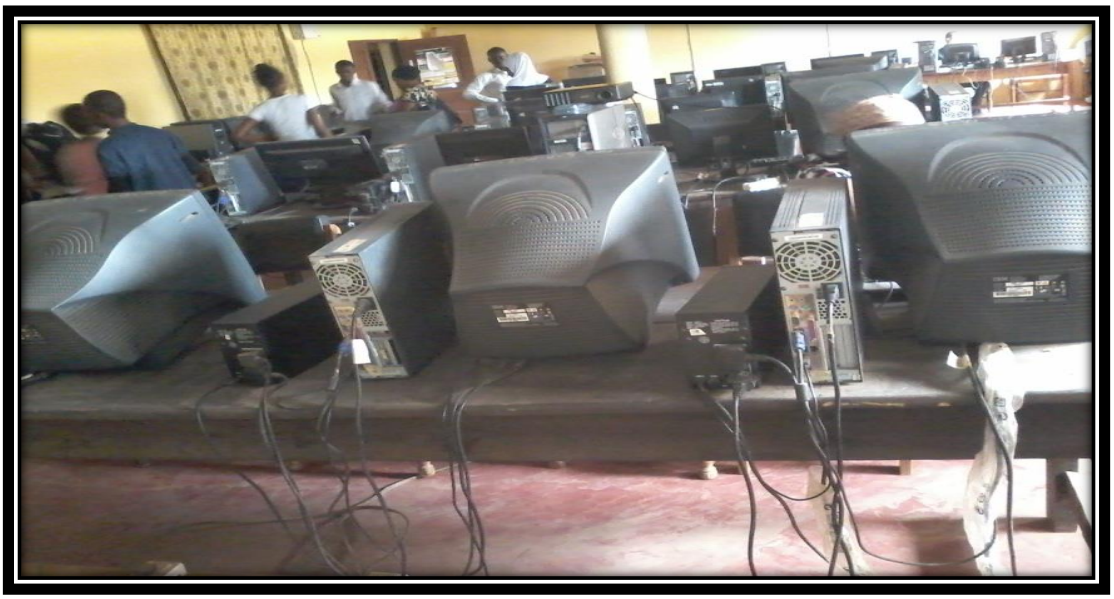


Figure 4.12: Back view of computers in the laboratory

Figure 4.13: Front view of computers in the laboratory

Although the researcher is an employee of the college, the research work was done independently but with full support, in accordance with college policy on staff development. The research product belongs to the researcher whilst the data belongs to the college and it is hoped that it will be of great value. The researcher is both participant observer and human instrument. Due to previous experiences and knowledge regarding *how* this problem was addressed in another setting, certain biases were brought to this study which may have shaped the way in which data collection and interpretations were made. It is believed that

students' preconceived idea *that programming is hard* would alter as they develop the skills needed for programming. This may have impacted on the way data were collected and experiences interpreted. Nevertheless, the rigour of the research process was preserved.

4.2.5 Time horizon

The two common types of time horizon used in designing research are cross-sectional (studying a phenomenon at a particular point in time) and longitudinal (collecting data by studying the progress, change and development of the same individual over time) (Cohen et al., 2007; Saunders et al., 2012). These two are explained in detail.

- ***Longitudinal studies***

A longitudinal study focuses on the aspects of human growth and is thus mostly used in conjunction with the word *developmental* (Cohen et al., 2007). The strength of longitudinal studies to span a long period of time gives researchers the opportunity to collect data and pay attention to details during the research (Scott and Morrison, 2005). One major characteristic of longitudinal studies is the establishment of causal relationships from an association, or correlation, statistical study since such studies involve identifying a change in one characteristic that results in the other (Cohen et al., 2007; Scott and Morrison, 2005). Despite the strengths of longitudinal studies, they suffer from high attrition and control effects as well as being expensive and time-consuming to conduct (Cohen et al., 2007; Scott and Morrison, 2005).

- ***Cross-sectional studies***

A cross-sectional study holds the *disadvantages* of longitudinal studies as *advantages,* but is not appropriate for use in a study that involves causal relationships. Another challenge of a cross-sectional study is the issue of sampling. Since it deals with different participants at different stages of the study, different samples have to be used. This makes the result incomparable (Cohen et al., 2007). For this study, a cross-sectional time horizon was considered. The justification for a cross-sectional study was based on the time frame (two years) where samples were chosen at different stages of the study. It was also less expensive to conduct.

4.2.6 Techniques and procedures

The last stage of the *onion* refers to techniques and procedures which include data collection and data analysis. It discusses different methods that can be used to gather and analyse data, addressing the research question (Crotty, 1998). Nieuwenhuis (2014b) and

Hendricks (2006) assert that multiple data collection strategies, such as interviews, observations and document analysis promote the credibility and trustworthiness of the research. An interpretive research involves multiple collections of data. The data collection methods used in this study are: (1) learning approaches, (2) observations, (3) artefacts, (4) interviews and (5) documents (see section 4.3). To describe the methods, it is necessary to first describe the population, sampling and participants.

4.3 DATA COLLECTION PROCESS

Data is the evidence researchers use to respond to questions about their research interest (Phillips and Kevin, 2014). From a phenomenological perspective, Van Manen (1984) stressed that the data collection process is a form of data generation which allows the researcher to develop an understanding of the phenomenon under investigation. The data collection process aided the researcher in getting to know the participants, changing her practice and developing her teaching identity. This section details the data collection process and how it emanated from the research design process. The data collection process for the two cycles follows the same procedure as depicted in Figure 4.14. However, since *each cycle involves a different set of participants*, changes occurred in the data collection process of the two cycles, specifically in the determination of population, sampling and selection of participants and learning approach. This further impacted on other stages of data collection. Cycle 1 commenced with a selection of participants followed by a collection of baseline data regarding the researcher's learning preference and that of the participants. The data collection processes and other activities are systematically explained in the following sections.

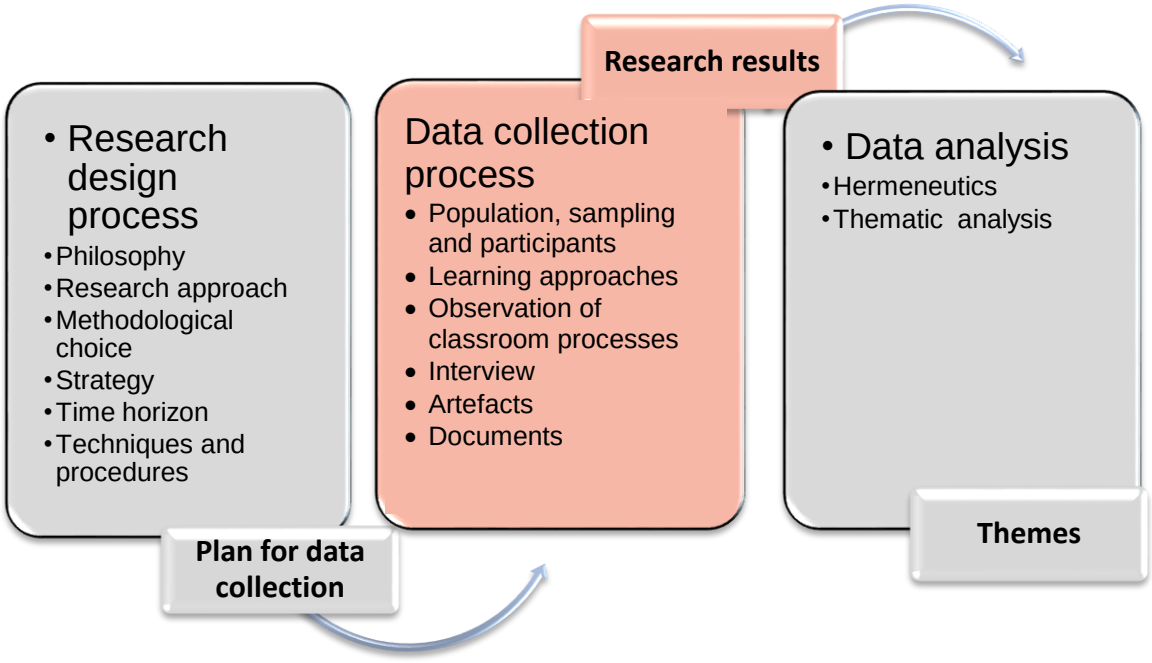


Figure 4.14: Data collection process

4.3.1 Population, Sampling and Participants

- ***Population and sample***

All computer science students enrolled in 2015/2016 and 2016/2017 academic sessions formed the population of the study. In determining the sample, purposive sampling methods were used (Efron and Ravid, 2013) to elicit participants' perceptions and lived experience on the intervention. These students were purposively sampled because they were all required to learn procedural programming as one of the compulsory courses for graduation identified in the computer science curriculum.

- ***Participants***

The participants were enrolled to study computer science combined with other courses, for example computer science and mathematics, for a minimum of three and maximum of five years. The participants comprised both male and female 100 level computer science first year students (2015/2016 and 2016/2017). In the first cycle of the study (2016), 10 of the students were male and 14 were female. The percentage distribution of the students, according to their states of origin were, Lagos state (50%), Osun state (16.7%), Oyo and Ogun states (15%) and Abia and Ondo state (4.1%). All students were between the ages of 18 and 22 years. The students spoke mostly English and Yoruba while some spoke Igbo. Table 4.2 gives an overview of data collected, as well as the participants selected, for the first cycle of the study.

Table 4.2: Overview of data collected and participants selection for the first cycle

Data collection strategy	No of Participants	Place and time	Duration	Research instrument	Role of the researcher
Questionnaire (Appendix A 2 and Appendix A 3)	24 students	At school	Depending on student's pace	Learning approach instruments	Administered and guided students
Questionnaire (Appendix A 3)	1 lecturer	Outside the research area	30minutes	Learning approach instruments	Fill questionnaire
Observation	24 students	During classroom session	2 hours	Video, field notes	Participant observer
Observation (Appendix A 18)	24 students, 1 researcher	During classroom session	2 hours	Checklist	Non-participant observer
Artefact (Appendix A 6 - Appendix A 14)	24 students	During classroom session	Depending on the activity	Test instrument, class work, projects, assignments	Facilitator of learning and administering of test questions
Retrospective Think Aloud (Appendix A 5)	6 students	At school	10 minutes per student	Programming questions	Observer/ interviewer
Interview (Appendix A 4)	6 students	At school	1 hour per student	Interview protocol	Interviewer
Document (Appendix A 22)	6 students	School/home	Not defined	Students' reflective journal and researchers journal	Document study

In the second cycle (2017), 13 of the students were male while 21 were female. The percentage distribution of the students, according to their states of origin were, Lagos state (55.8%), Ondo state (17.65%), Ogun state (8.82%), Delta state (5.9%) and Kogi, Benue, Edo and Enugu (2.94%). All students were between the ages of 17 and 24 years old. The students spoke mostly English and Yoruba while other languages were also spoken. Table 4.2 represents the participants' distribution for the second cycle of the study. The researcher acted as both participant and human instrument for data collection in these two cycles of the study. As *participant observer*, her immersion, reflections and attitudes influenced the changing real-world situation of the classroom (Creswell, 2014; Kosky, 2009; Saunders et al., 2012). As *human instrument*, she became part of the population analysed in the research study (Phillips and Kevin, 2014).

Table 4. 3: Overview of data collected and participant selection for the second cycle

Data collection strategy	No of Participants	Place and time	Duration	Research instrument	Role of the researcher
Questionnaire (Appendix A 3 and Appendix A 15)	34 students	At school	Depending on student's pace	Learning approach instruments	Administered and guided students
Observation	34 students	During classroom session	4 hours	Video, field notes	Participant observer
Observation (Appendix A 18)	24 students, 1 researcher	During classroom session	2 hours	Checklist	Non-participant observer
Artefact (Appendix A 6 - Appendix A 14)	34 students	During classroom session	Depending on the activity	Test instrument, class work, projects, assignments	Facilitator of learning and administering of test questions
Retrospective Think Aloud (Appendix A 5)	8 students	At school	10 minutes per student	Programming questions	Observer/ interviewer
Interview (Appendix A 4)	8 students	At school	1 hour per student	Interview protocol	Interviewer
Document (Appendix A 22)	8 students	School/home	Not defined	Students' reflective journal	Document study

4.3.2 Learning approaches

The first stage of the data collection process was to determine the researcher's learning preference (*q.v*. Chapter 5). She liaised with an HBDI practitioner and obtained an access key at a cost which was filled around September 2015. The practitioner explained the profile results and this information served as background knowledge to explain the concepts to the participants. Consideration of the results lead to flexibility in instructional planning so that *all* quadrants were catered for and not only her preference alone. Following this, students completed the KLSI instrument (*q.v.* section 3.2.3.2) in the first meeting along with the researcher. Not all the students were present at this first meeting. The responses of the participants were elicited through the instrument during a group administration. Other students completed the instrument as they joined the class in subsequent classroom sessions. A scoring key was used to determine the learning styles of each student. This helped in grouping the students, based on their communicated preferences. The filling of the HBDI instrument was done subsequently. A further explanation of this section is presented in section 5.2.2.2.

4.3.3 Observation of classroom setting

The second stage was observing the classroom setting. The researcher's role here was that of a participant observer (Saunders et al., 2012). In doing this, her voice and the viewpoints regarding participants were recorded (Phillips and Kevin, 2014). The real-life setting of the classroom was properly and purposely observed, giving a powerful insight into the setting (Cozby and Bates, 2012; Creswell, 2014; Efron and Ravid, 2013; Saunders et al., 2012). The participatory observation afforded an insight into the group interactions of participants in the procedural programming classroom. The researcher kept notes on observations made, as soon as possible after they were made, and this included verbatim, or near verbatim, quotes, concrete descriptions of the setting, people and the events involved (Willig, 2008). The observation was done with three forms. Students were well engaged, so much so that they did not really focus on the recording. In some instances though, they tended to act clownish so that their funny acts would be recorded.

Anecdotal records: field notes in conjunction with a reflective journal were kept. Field notes contained descriptions of the researcher's reflections and observations during the teaching process and interview session (Kosky, 2009). Notes written at the seminar attended (Nieuwenhuis, 2014b), intuitions and new ideas, discussion with experts/colleagues in the field and students' thoughts during the intervention program were all documented. A reflective journal containing ideas and thoughts regarding experiences and professional activities were kept while adjustments were made, where necessary.

Video-recording: An open-ended qualitative observation was also done using a *video.* Videotape recording was used to observe participants' actions and behaviours during the teaching process since there was limited time to record observational notes. The videotape provided a permanent record of participants' behaviour, shared meanings during problem-solving, attitudes and social interactions which the researcher intended to notice (Efron and Ravid, 2013). A person was employed to do the video recording and he was instructed on what to capture in the sessions as well as where and how. A fee was attached to his service. The recording of each class lasted two hours after which the video clips were transferred to a memory card which could then be viewed on the researcher's laptop. The researcher made it a point of duty to carefully watch the video footage of each day's class to so record and reflect upon noted behaviours, thus making the necessary adjustments in subsequent classes.

Structured observation: This was also used to quantify the behaviour of participants in the classroom through: tally sheets, behaviour logs (Appendix A 19) and checklists (Appendix A 17 and Appendix A 18). A *behaviour log* is a running record that captures a more focused observation of participants' behavioural patterns during a given situation (Efron and Ravid, 2013). This was possible through the observation of one, or two students per day, while these participants were engaged in class activities.

4.3.4 Artefacts

The third stage of the data collection is the creation of artefacts. An artefact is physical document that produces additional data regarding the research question (Phillips and Kevin, 2014). These artefacts provide physical evidence of students' work and were generated by different classroom activities and group projects, mostly on visual and procedural programming. The generation of artefacts was broken down into three stages.

Firstly, the participants' artefacts included a *test of individual concepts* (Appendix A 6) administered towards the end of the learning process to examine the achievements of students at a lower cognitive level. Secondly, *interim tests* (Appendix A 7) and (Appendix A 8) and practical project work on visual programming were given to students toward the end of the teaching and learning process. The tests examined the achievements of students at a high cognitive level. Reflections were made regarding the test results and this guided teaching and learning in subsequent classes. Thirdly, a *final test* (Appendix A 9) which covered concepts on control structures, looping and arrays were given to the participants. This was a long and complex question which tested students' comprehension level in procedural programming from a holistic point of view. They also produced a poster drawing of their experiences in visual and procedural programming in the classroom. The tests were face validated by my supervisor and colleagues in the department before they were administered. The reliability of the tests was further determined by a statistician, after administration. Results regarding the reliability of the test helped in re-designing the constructs with lower reliability coefficients for the second cycle of the study.

4.3.5 Interviews

The fourth data collection process was conducting interviews with the participants. The interview is a dialogue (Nieuwenhuis, 2014b) between the teacher researcher and participant in order to provide an understanding of participants' experiences by the voicing of ideas, knowledge, values and opinions about the issue under investigation from their own perspectives (Efron and Ravid, 2013). At the end of the semester the researcher selected

participants, based on their performance in the different assessments which they were exposed to during the classroom sessions. Therefore, a total of six and eight participants were selected for the first and second cycle, respectively. These participants, comprising of high, medium and low achievers, were selected for conducting a hermeneutic phenomenological inquiry.

Pilot test: The questions were pilot tested with two students (medium achievers) who took part in the intervention, but who were not part of the selected students for the final interview. From these two interviews, the researcher noticed that the students offered similar responses to some questions regarding on the protocol. Therefore, I had to remove one of the questions to avoid repetition. More probing questions were added before exposing it to the participants interviewed.

Retrospective interview: Before the interview was conducted, a one to one retrospective think-aloud procedure was done in which participants were asked to rewrite a series of code on the computer (Whalley and Katso, 2014). Participants verbalised their thoughts, actions and reflections (Schunk, 2014) while watching a video of their task processes. The session lasted for a minimum of ten minutes and the retrospective interview (Appendix A 5) was embedded within the main interview session.

After the retrospective phase, a face-to-face semi-structured interview (Appendix A 4) was used in which the interviewer prepared a set of questions with sub-questions which could be used to probe ideas further and gather more information (Kosky, 2009; Langdridge, 2007). Accordingly, the researcher was cognisant of data quality issues associated with semi-structured interviews as well as forms of bias and validity (Saunders et al., 2012). She thus checked her tone, or verbal behaviour, so that the biases did not influence interviewee responses to the questions being asked (Saunders et al., 2012, p. 381). Also, the participant may be sensitive, have misconceptions, or not be prepared to discuss certain questions in the interview, even if he/she is willing to participate (Cohen et al., 2011; Saunders et al., 2012). To avoid these pitfalls, the researcher carefully prepared an interview protocol with possible appropriate and probing questions. Leading questions were used when the participant did not understand the question/s while answers given were probed further to ensure exactness. Likewise, the participants had prior knowledge that the researcher wanted to explore their understanding on the intervention and, where a participant was not willing to answer a question, the researcher moved to the next question without showing any irritation.

The interview protocol (Appendix A 4) ranged from more *general* questions on the students' demographics to *specific* questions regarding the intervention and programming knowledge. The interview lasted about one hour. A tape recorder was used to record students' views and perceptions during the interview process. The interview took place at an on-campus residence which was quiet enough to facilitate the audio and/or video recording process. In conclusion, an interview transcript was made. Although the interviews obtained rich data from participants and probed their perceptions further, possible limitations included participants not being comfortable with a tape and/or video recorder in front of them and the fear that shared information might be used against them in future (Mertler, 2008). The researcher assured the interviewee that his/her data would not be shared with anyone.

4.3.6 Documents

Documentary evidence provides a helpful background and context to the phenomenon under investigation (Kosky, 2009; Nieuwenhuis, 2014b). Whilst reflecting on textual data, *participants kept a reflective journal* (Appendix A 22) in which they recorded their feelings, struggles, successes and/or personal accounts of growth and learning for the whole semester. The *researcher also kept a personal diary* for data collection.

4.4 DATA ANALYSIS

This section discusses the data collection process, as shown in Figure 4.15. The analysis of various data collected in this study is based on an interpretive philosophy which is also a philosophical grounding for hermeneutics (Nieuwenhuis, 2014b).

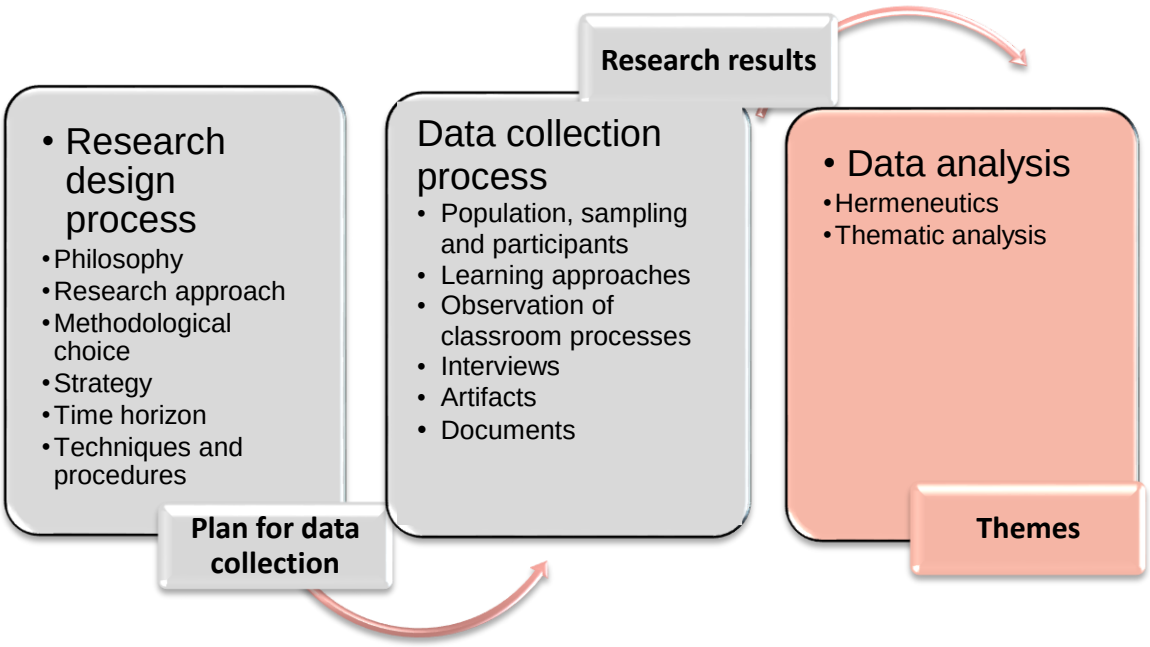


Figure 4.15: Data analysis process

Hermeneutics, as an approach to data analysis, strives to understand or make meaning of text in context (Langdridge, 2007; Nieuwenhuis, 2014b). This understanding of text within the context, as explained by (Willis and Jost, 2007), can only be found in *tacit* knowledge, thus knowledge obtained through experience. Hermeneutics share characteristics with phenomenology and interpretivism which focus on meaning residing within people, the interest and reason for the subjective meaning of their daily activities (Willis and Jost, 2007). Participants' lived experiences regarding procedural programming in the context of the study, collected through the data collection methods, were analysed in the study. Hermeneutics also support action research based and its emphasis on reflexivity in inquiry. This opens the researcher's eyes to the fusion between understanding and the action taken during inquiry (Coghlan and Brydon-Miller, 2014). Data analysis in hermeneutics involves a fundamental principle called *the hermeneutic cycle* on which six other principles are built. The hermeneutic cycle suggests that the movement of understanding is from the *parts* to the *whole* and from the *whole* to the *parts* (Klein and Myers, 1999). Gadamer (1997) noted that data analysis in hermeneutic phenomenology can be done using the hermeneutic cycle.

4.4.1 Hermeneutics

This section describes the seven sets of principles, as recommended by Klein and Myers (1999), for the interpretation of data in interpretive research. The hermeneutic steps were applied continually during the data analysis stage. An understanding of *how* the data were collected and its meaning, is explained while making the result the researcher's. The essence is to form a connection between interpretations (Klein and Myers, 1999), as discussed in Chapters 5 and 6. The hermeneutic principles, their descriptions and how they were applied in the study are presented in Table 4.4.

As explained by (Klein and Myers, 1999), the hermeneutic cycle (*q.v.* section 4.4.1) is an encompassing principle from which the other six principles stem. Each of the principles is interdependent and together they create the 'whole'. A detailed description of the context of the setting (*q.v.* section 4.2.4.6) has been discussed (Creswell, 2007). The whole (final story) derived from the setting guided the researcher in making a judgement while applying each principle individually. The whole impacted upon the parts (application of each principle), and vice versa, to produce themes and interpretations. In interpretive analysis and hermeneutics especially, an understanding and interpretation of the text within the context studied is the ultimate goal (Klein and Myers, 1999). This helped the researcher to interpret the text based on its story from the context of the setting.

Table 4.4: Hermeneutic cycle processes

Principles	Description	Application
Hermeneutic cycle	The movement of human understanding is by iterating amongst mutually dependent meaning of parts to the whole and from the whole to the parts they form.	Iterating among observation, interview, various artefacts of each participant and in groups as parts, and whole context, that determines the full understanding of the teaching programming and the development of program comprehension skills in procedural programming.
Contextualisation	A social and historical reflection of the background of the context in a way that the participants will understand how the situation under examination came into being.	This was applied in section 4.2.4.6 by reflecting upon the social and historical background context, as observed by the researcher.
The interaction between researcher(s) and subjects	A critical reflection on the interaction that exists between the researcher and the subject, and how the data obtained were socially constructed.	A reflection on how data was collected through interactions between the researcher and the participants. The data obtained came in two forms: determining the participants' learning approaches through the learning approach instruments, which were discussed together with the participants to understand them. Reflections on the learning approaches guided the researcher in designing a teaching and learning framework for procedural programming. Also, artefacts produced by the participants were discussed by the researcher and participant at each stage to influence changes in other artefacts produced.
Abstraction and Generalisation	It requires connecting the idiographic details made known by the data interpretation through the application of principles to theoretical abstractions which depict the nature of human understanding and social action as they were experienced.	The findings of the study were discussed in relation to important concepts which emerged from a thematic analysis. The findings finally helped in designing a teaching and learning process framework for programming.
Dialogical reasoning	This principle requires the researcher to be conscious of possible theoretical preconceptions, or biases, guiding the research design and the actual findings that surface from the research process.	Throughout the study, the researcher was constantly conscious of her biases which guided the research and how it coloured the findings.
Multiple interpretations	Requires the researchers' sensitivity to the influence the social context has on the possible differences in multiple meanings given by participants who experienced the same study, as well as the reasons behind their meanings.	The participants' interpretations of given concepts and reasons behind their meanings were documented. Their multiple meanings were discussed under each theme and reported on in the quotations (*q.v.* Chapters 5 and 6).
Suspicion	It requires the researcher to be sensitive to possible preconceptions and systematic misrepresentation (distortion) in the data obtained from the participants.	The researcher applied this principle by questioning throughout the data analysis whether the participant's stories were really what they described.

4.4.2 Thematic analysis

Data analysis was done using thematic analysis (TA). As stressed by Langdridge (2007), hermeneutic phenomenology involves the thematic analysis of data. TA is a method used for analysing, classifying and reporting themes that relate to data (Alhojailan, 2012; Braun and Clarke, 2006). Thematic analysis was introduced as a basic skill for conducting other qualitative research (Vaismoradi, Turunen, & Bondas, 2013). TA is appropriate for a study which attempts to discover, through interpretations, the analysis of the frequency of a theme within the whole data content (Alhojailan, 2012). A theme is the building block of thematic analysis of data, hence it is a disparate construct that appears in the data and constitutes the findings (Given, 2008; Vaismoradi et al., 2013). Braun and Clarke (2006, p. 10) noted, "it captures something about the data in relation to the research question and represents some level of patterned meaning within the data set", noted (Braun and Clarke, 2006, p. 10). As a result, a *theme* should relate to the research question and thus appear in several instances in the dataset.

For clarity's sake, a few terms will be defined. "*Data corpus*, the entire data collected for a research study; *data set*, all data from the corpus being used for a particular analysis; *data item*, each individual piece of data collected (both dataset and corpus); *data extract*, an individual coded chunk of data extracted from a data item" as per (Braun and Clarke, 2006, p. 5). A number of decisions for data analysis, which should be made explicit by the researcher before the research commence with an ongoing reflexive dialogue, were followed as suggested by Braun and Clarke (2006). Below are the measures the researcher used in making these decisions in the study:

- The researcher was flexible in her judgement on what constitutes a *theme* and its prevalence within the data set.
- The researcher gave a *rich* and *comprehensive* account of the entire data set instead of just describing the theme that emerged. Through this rich description, readers will have a sense of the significant and important themes from the data set.
- The researcher used inductive *thematic analysis* and *deductive analysis*. This decision stemmed from the researcher's approach to research (*q.v.* section 4.2.3). In the analysis, themes are linked to the data (*data drove*) and from pre-existing themes. This allows an appreciation of how each theme contributes towards an understanding of the whole picture. A global view of data is obtained as the researcher described similarities and differences among participants. Nevertheless, the data was later linked to the theoretical concepts during the interpretation stage (*q.v.* Chapter 7).

- Regarding the level on which themes are identified, the s*emantic content* (surface meaning) of data was first identified. Followed by *latent conten*t which relates to a critical examination of the background issues supporting participants' ideas and perceptions which informed the semantic content of data. This second phase involves interpretation of themes.
- It is very important before the study commences that a researcher states the theoretical position for using thematic analysis within a study (Braun and Clarke, 2006). Thus, the researcher conducted thematic analysis within the *constructionist* paradigm which acknowledges the ways in which each participant gives meaning to their experience as well as *how* the social context influences meanings. This constructionist, however, uses latent themes.
- A thematic network which shows a visual representation of the interrelationship between themes (*q.v.* sections 5.3.2 and 6.3.2). The thematic data analysis process used in the study is explained in Chapters 5 and 6.

4.5 TRUSTWORTHINESS

Trustworthiness refers to the way in which the researcher convinces the audience and establishes that the data obtained in the research findings originate from the context in which it was studied (Hendricks, 2006) and that it was not influenced by the researcher's biases, worldview and particular perspectives (Stringer, 2007). It is essential for teacher-researchers to ensure the quality of their data (Mertler, 2008). Trustworthiness in qualitative research has been widely discussed and it can be achieved through credibility, transferability, dependability and confirmability (Guba and Lincoln, 2005). Regardless, the definition of trustworthiness has not been widely discussed among action researchers (Hendricks, 2013). There are, however, five widely supported criteria by theorists for accessing trustworthiness in action research (Coghlan and Brannick, 2014; Hendricks, 2013; Mills, 2011). *Democratic validity* was achieved with the researcher giving the participants' the opportunity to articulate their opinions and positions during the interview session, and reporting on all spectrums of these multiple voices. The first cycle of the action research study, *outcome validity*, serves as a baseline for continued planning and reflection in the second cycle of the study while deepening the researcher's understanding of the study. *Process validity* was achieved by deeply studying the research problem and obtaining varieties of rich qualitative data while correct interpretations of findings were also made. *Catalytic validity* deals with active involvement of the researcher and participants, and consideration for the extent to which processes and outcomes change the researcher's practice. Finally, *dialogic validity* refers to the researcher's discussions regarding planning, research results and findings with a critical friend and/or supervisor.

As summarised by Creswell (2014) and Hendricks (2013) engaging multiple sources of data (including interviews, documents and observations) can establish trustworthiness which is enhanced through identified strategies including: triangulation, member checking, thick description, clarification of bias, presentation of negative case analysis, prolonged time in the field, peer debriefing and the use of external auditor, presentation of result to key audiences, engaging in continuous ongoing reflection and recording data accurately. To establish trustworthiness in this study the researcher had to self-reflect and communicated the methods used in great depth (Phillips and Kevin, 2014). In this light, the following checklist (see Table 4.5) guided the researcher in ensuring trustworthiness in the study (Creswell, 2014; Hendricks, 2013; Millis, 2010; Stringer, 2007; Stringer, 2014).

Table 4.5: Checklist for trustworthiness criteria

Trustworthiness criteria	Applications in the study
Triangulation	Evidence from different sources such as interviews, observations, student journals, researcher journal, whole brain feedback transcripts were used to build sound validation for the themes developed.
Member checking	Participants were asked to cross-check the transcriptions to confirm whether the researcher reported the exact discussions from the interview.
Thick description	The researcher presented a detailed description of each theme from different participants' experiences.
Clarification of bias	The researcher clarified her biases brought into the study and how it may have influenced the interpretation of the findings (*q.v.* Chapter 5, section 5.2).
Presentation of negative case analysis	Evidence from the transcripts that do not coalesce with the findings related to each theme were discussed (q.v. sections 5.3 and 6.3).
Prolonged time in the field	The researcher was in the field for each semester, during the first and second cycle, and she built relationship with the students in the context of the study.
Peer debriefing	The researcher's supervisor constantly checked and questioned her interpretations of the findings such that the account could be easily understood by the reader.
External auditor	The researcher sought assistance from a critical friend from a university in the Netherlands to critique her thinking. She then reviewed her interpretations based on his comments.
Continuous reflection	The researcher engaged in constant reflection throughout the research (q.v. sections 4.2.4, 5.4 and 6.4).
An accurate record of data	The researcher ensured that she recorded the participants' opinions verbatim.

4.6 ETHICS

Ethics procedures are a crucial aspect to all research. The Helsinki declaration of 1972 affirmed that it is important to ensure ethical standards which protect the rights of human subjects or participants. Stringer (2014) added that because of the participatory nature of action research, ethical guidelines should provide provisions for care, rights to safety and informed consent for all stakeholders. These were ensured in the study, according to the

University's ethics and research policy. An explanation, at the hand of subheadings, of how ethics standards were achieved throughout the research study follows.

- ***Access***

Access to the college to conduct the study was awarded in principle, and in writing, by the provost/management of the college. Permission to conduct the study, as well as access to other necessary materials for the research study, was also given (Appendix A 1). A tentative approval was first given before the final approval to commence the study was released.

- ***Informed consent***

All the participants received a consent letter which was completed during an informal meeting and during class sessions. The option to withdraw their consent at any time of the study was explained to participants. They all consented to participate in the study. An informed consent letter was also given to the non-participant observer and he gave his consent to participate in the study.

- ***Care***

Saunders et al. (2012) advised researchers that the research design should not expose the participants to embarrassment, harm or any other material disadvantage. During the study, there was no harm done to the students and the researcher continuously monitored the situation to avoid harm and embarrassment.

- ***Confidentiality and Anonymity***

Creswell (2012) stresses the importance of participant confidentiality in today's research. The lives and experiences of the participants used in the research study are to be kept private. Nieuwenhuis (2014a) also added that both the researcher and participants must understand the results and findings of the study. Based on this premise, all participants' information and shared responses were carefully described so that the identities of the participants were not revealed. Pseudonyms were used in the data analysis instead of original names. The audiocassettes and videotapes used during the research study were given to the researcher's supervisor to be kept for at least fifteen years at the University of Pretoria before being destroyed.

4.7 CHAPTER SUMMARY

This chapter described the *research design*, *data collection* and *data analysis processes* as well as the researcher's choice of research methodology. Figure 4.16 depicts each layer of the research onion used in the study after many other options had been considered. An *interpretive philosophy* stands as the foundation on which the *cross-sectional*, *multi-method qualitative* and *action research* helped to investigate how first year college students learn programming, from visual to procedural. The *inductive* and *deductive approach* was used to explore the collected data and to generate themes which will be analysed and interpreted in Chapter 5. Reference was made to the researcher's role and her position as *participant researcher* was also established. Ethical considerations and issues of trustworthiness were made explicit. Trustworthiness assisted the researcher in validating the data collected from different sources in relation to students' programming skills. The next chapter will focus on results and findings gained from the first action research cycle.

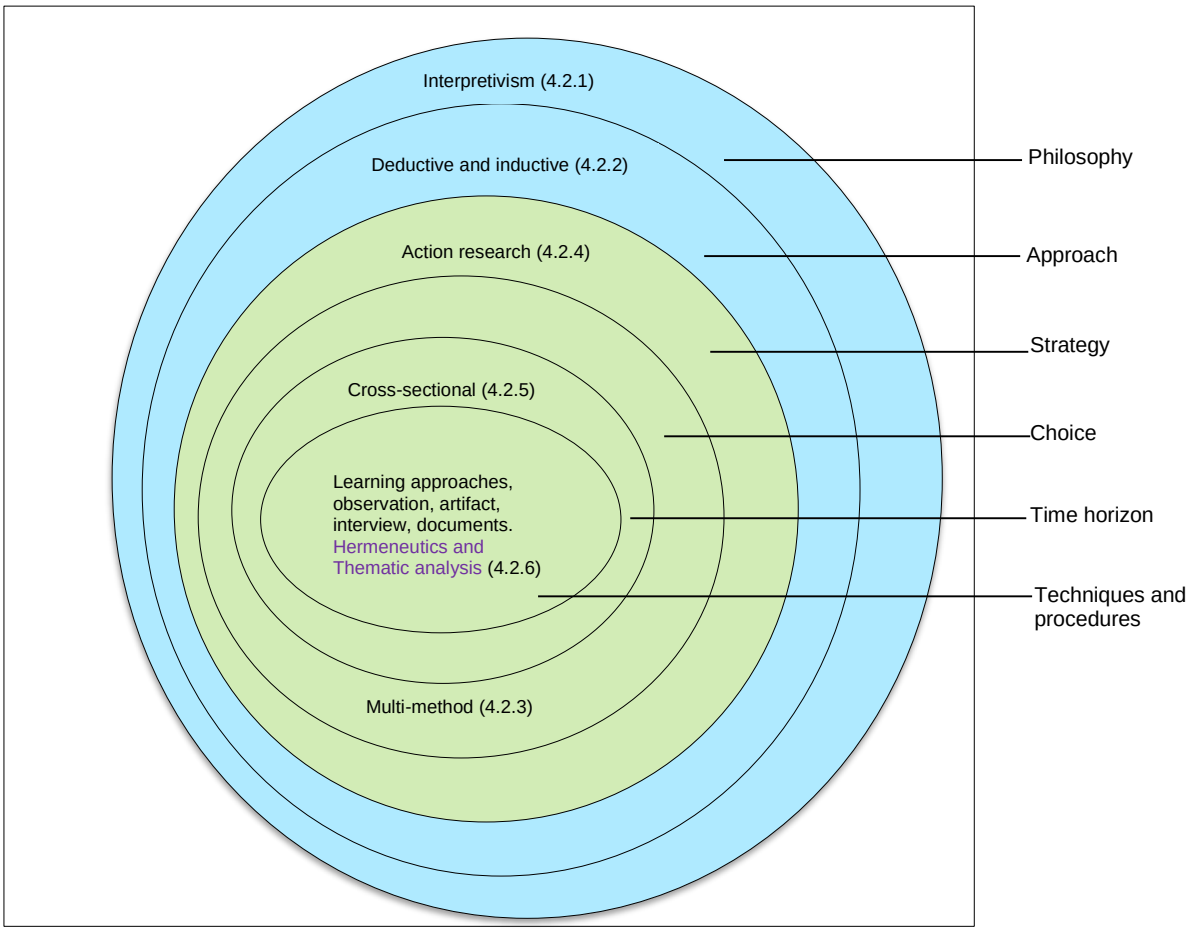


Figure 4.16: The applied research onion

CHAPTER 5 – RESULTS AND FINDINGS: ACTION RESEARCH CYCLE 1

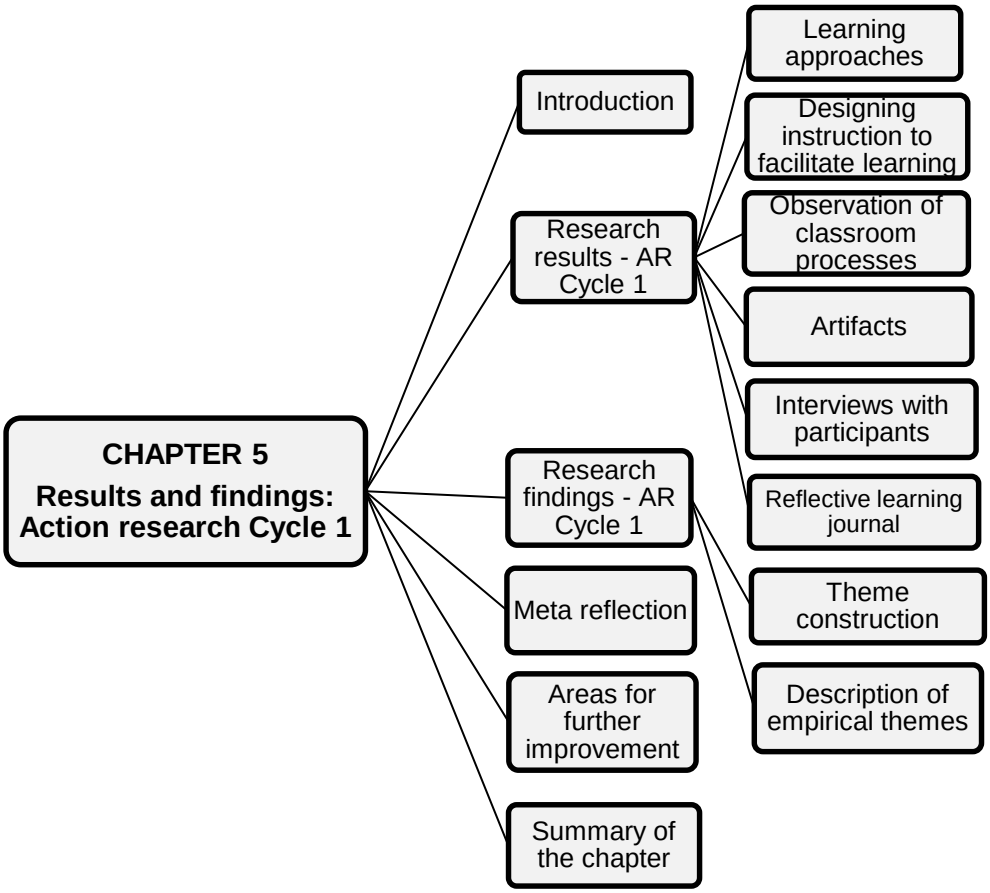


Figure 5.1: Schematic representation of Chapter 5

5.1 INTRODUCTION

In the methodology section (*q.v.* Chapter 4), the researcher discussed how theory guided data collection, the processes involved in data collection as well as the data analysis process. This chapter presents and discusses results gained from the data collected and findings emanating from data analysis of the first cycle action research. This chapter is divided into six sections with the first section focusing on the introductory aspect. The second section (*q.v.* section 5.2) discusses the results of the data collected (learning approaches, observation, artefacts, interviews and documents) whilst observing effects of the new action. The third section (*q.v.* section 5.3) details the findings from the data collected. The findings are discussed in relation to generated themes extrapolated from the data. Section 5.4 presents meta reflections while the last section summarises the chapter. The discussion commences with a presentation of the research results.

5.2 RESEARCH RESULTS – AR CYCLE 1

This section presents results which depict the *'observing the effects of new action'* phase of the first action research Cycle 1 (*q.v.* Figure 4.9:). The results focus on learning approaches (*q.v.* section 2.1); observation (*q.v.* section 5.2.2); artefacts (*q.v.* section 5.2.3); interviews (*q.v.* section 5.2.4) and documents (section *q.v.* 5.2.5). Each of the sections is described and supported by raw data. This section commences with a description of the researcher's learning approach and that of the participants. **The 'I' described in this section is the epistemological focus of the action research, but the narrative is not described in a self-centered way**. As a participant, I acknowledge that I brought biases into this study which may have influenced the way in which results and findings were interpreted. It is possible that my approach to learning influenced the way in which I designed the whole brain learning as well as the way in which I dealt with situations in the programming classroom. In addition to this, I am more knowledgeable about programming than my students and it is thus possible that my knowledge influenced the way in which I judged my students. Lastly, because one becomes familiar with one's students over time, I might have awarded marks in favour of some of the students. Figure 5.2 below presents a schematic representation of the data collection timeline for the first cycle.

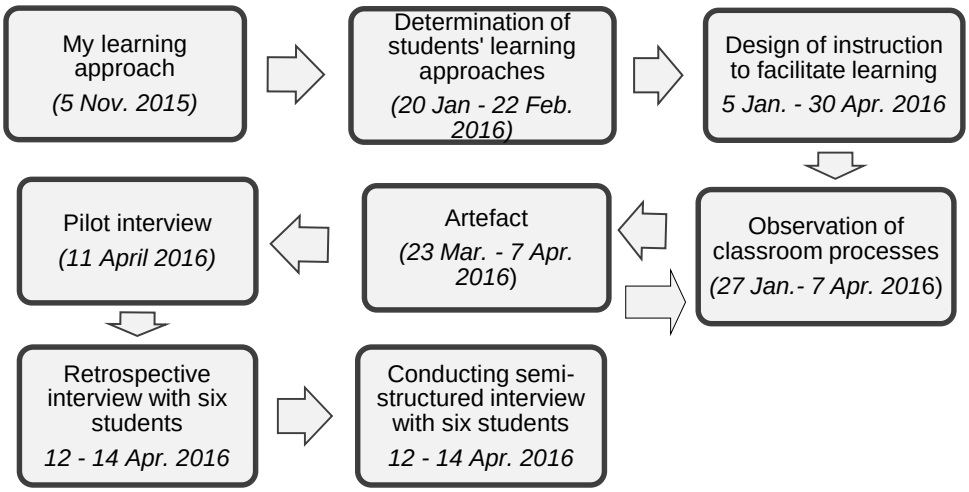


Figure 5.2: Data collection timeline for the first cycle

5.2.1 Learning approaches

The first step taken during data collection was to determine how my students and I approached learning. The results of this section are presented in two ways: the researcher's Herrmann Brain Dominance Instrument (HBDI) data summary and profile overlay and the

participants' Kolb's learning approach inventory (KLSI) and HBDI learning approaches, as discussed in subsections 5.2.1.1 and 5.2.2.2, respectively. The rationale for using both instruments was based on two reasons. Firstly, the online (HDBI) survey (*q.v.* section 3.2.3) is expensive and it was therefore not possible to cater for all students in the classroom. Secondly, Coffield et al. (2004) advised *against* the use of only one learning approach instrument to ascertain the best indicator of pedagogical change.

5.2.1.1 Description of the researcher's learning approach

I completed an online HBDI survey instrument created by the Ned Herrmann Group to discover my thinking preference. The survey enabled me to specify descriptors relating to the way I learn, think and communicate. The survey also shed light on how I prefer others to learn and how this influences my own teaching style. The questions were answered honestly. The HBDI data summary sheet (*q.v.* Appendix C 4) provides a breakdown of the quadrants in which my responses fell. My profile results, in Figure 5.3, are accompanied with a 2112 preference code, thus indicating a double dominance in B and C quadrants. This profile is also characterised by the secondary preferences in the A and D quadrants which are occasionally functional (Herrmann International, 2009). To simplify, in quadrant A my preference is secondary (2) and in B it is primary (1) with a highest score of 108. For quadrant C it is also primary (1) whilst in quadrant D it is secondary (2).

Quadrant:	A	B	C	D
Preference code:	2	1	1	2
Adjective pairs:	5	9	5	9
Profile scores:	57	108	75	59

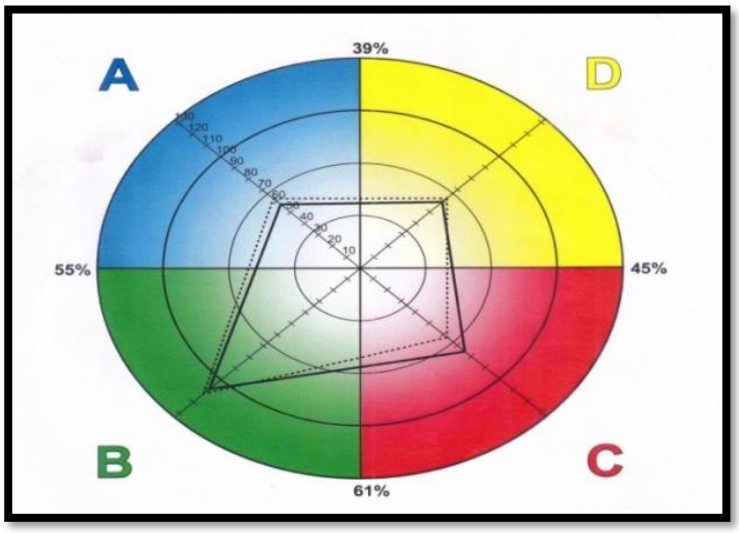


Figure 5.3: Researcher's profile
Source: Herrmann International (2016)

As described in the qualitative summary report from the HBDI result, my dominant preference is quadrant B. The descriptors that best describe my general day-to-day mental preferences are: *conservative, controlled, detailed* and *reader*. For work-related activities, my mental preferences are strongly rooted in the B quadrant and include: *organisation, planning* and *administration*. My next dominant preference is quadrant C with 75 points. I chose *spiritual* and *reader* as descriptive of me. Work elements that I associate with in the C quadrant are: *teaching*, *writing*, *expressing* and *interpersonal.* In quadrant D, *holistic* is descriptive of me. My least preferred thinking preference is quadrant A, with 57 points, in which *logical* and *mathematical* represented a good number of my descriptors.

The implication of these results in terms of the facilitation of programming is that the approaches which I am most comfortable with may include: establishing a rapport, listening to students' ideas, following procedures, being organised, breaking down a problem into steps, involving others and being sensitive to how my students feel. Qualities I might overlook during teaching are: ensuring creativity and active imagination, providing a big picture, illustrations, technical accuracy and critical thinking. It is possible that I might be biased in the way I design learning. With this understanding noted, it is necessary to look at examples of my students' profiles (see Figure 5.5), their expectations, struggles and how I reduced my biases during learning design.

5.2.2.2 Description of participants' learning approaches

The approaches through which students learn were determined using two different survey instruments. The rationale for using these two instruments is discussed above. All the participants' learning approaches were, firstly, determined using the Kolb's learning approach inventory (KLSI version 3) (q.v. section 3.2.3) while the HBDI was used at a later stage for selected participants. I should note that determining students' learning approaches was only done to facilitate their placement into groups for learning purposes, and *not* to determine performance.

KLSI profile score of participants

The results of the participants' KLSI surveys were determined using a scoring key (the cycle of learning, version 3.1 and the learning approach type grid, version 3.1) which the Hay group sent to me at my request, at no extra cost. Participants' responses were plotted against the "cycle of learning" and the "learning approach type grid" to determine their learning approach. Results, as depicted in Figure 5.4, show that from the 26 participants,

69% were *diverging* learners, 27% were *assimilating* learners and 4% constituted *converging* learners.

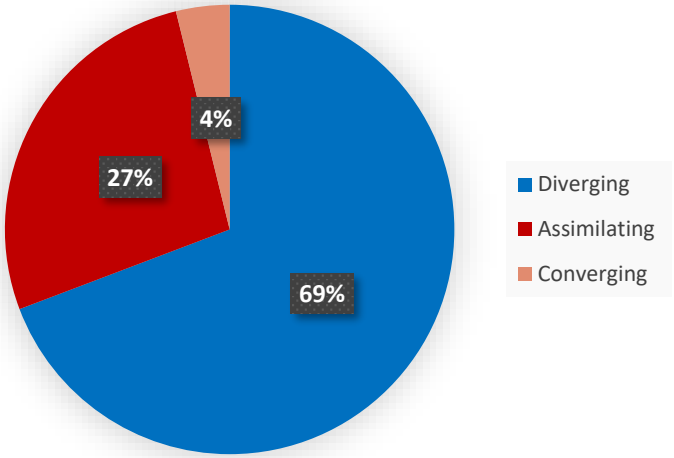


Figure 5.4: Pie chart of students' profile on the KLSI

In a quest to better understand my students, I mapped the experiential learning characteristics of Kolb's to the HBDI quadrants, based on a study by Potgieter (1999). This was discussed in Chapter 3 (*q.v.* section 3.2.3.2) of the study.

HBDI profile score of the participants

During the fifth week of the semester, 13 of the participants were randomly selected to complete the online HBDI survey, because the online HBDI was expensive. It was challenging to complete the online HBDI at the college computer centre, because the HBDI database was unable to load properly. Therefore, I asked a few of the participants to use my tablet while photocopies of the instrument were given to the others to complete. This process was properly guided. I then transferred the completed forms to the HBDI website using an individual web key. The Herrmann practitioner prepared the participants' profile results which enabled me to identify profile scores for each participant. From this result I discovered that there were diverse types of learners in my class. Examples of students' profiles are presented in Figure 5.5.

The profile of each of the thirteen students (*q.v.* Appendix A 20 and Appendix A 21) is not discussed individually, but samples are included to explain possible preferences in my programming classroom. I thus present profiles of students which *contrast* with mine, since the results confirmed that I tend to accommodate students whose profiles mirror mine. Students, with these profiles, would prefer that I accommodate them when facilitating the

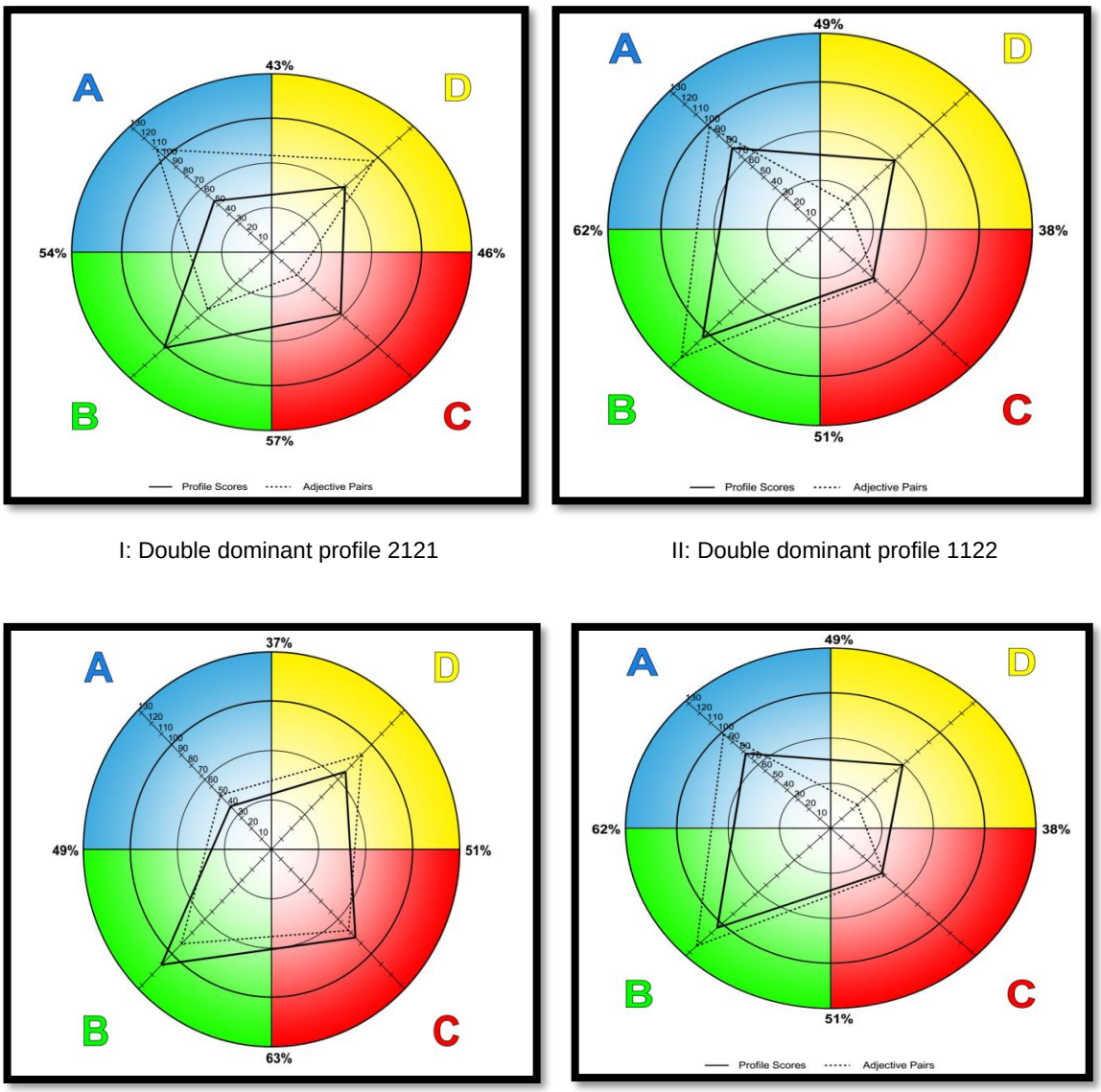
programming instruction. I am thus faced with the task of adjusting and adopting a flexible approach when planning and facilitating instruction.

- *Profile 1*: indicates a strong preference for quadrants B and D, while quadrants A and C are less preferred. The person who displays this profile would thus desire: organisation, planning and conservative processes, taken from the B quadrant. These are preferences I can accommodate. However, participants' preference for the D quadrant's holistic, creative and conceptual activities would be a challenge for me. As a facilitator, I needed to move beyond my comfort zone to facilitate participants' expectations in the D quadrant.

- *Profile 2*: indicates a double dominant, which means the participant is strongly represented in two quadrants, with the primaries being in the upper left A and lower left B. The processing modes in quadrants C and D are secondary but at the same time functional. The most satisfying types of work for this participant would include: data analysis, doing hands-on practicals, establishing order and details (Herrmann Iinternational, 2016). I would possibly accommodate this participant based on my descriptors in the A quadrant as *logical* and *mathematical*. However, I would need to put some effort into the process as they do not reflect my natural preferences.

- *Profile 3*: indicates a triple dominant profile with strong preferences in the B, C and D quadrants, which I would most likely accommodate based on my personal preferences. However, I tend towards the A quadrant, which is less preferred by this participant. We both need to be cognisant of the fact that neither of us prefer to think in the A quadrant.

- *Profile 4*: indicates a multi dominant profile with strong preferences in all four quadrants, though all preferences are not equal. This supposes that the participant can cope well with a diverse kind of activities in the classroom. Situationally, he/she would be able to move back and forth despite conflicts from within. I would possibly cope well with quadrants B and C. However, the other two strong preferences (A and D), which differ from mine, would prove challenging.



I: Double dominant profile 2121

II: Double dominant profile 1122

III: Triple dominance 2111

IV: Quadruple dominance 1111

Figure 5.5: Examples of students' profiles

Source: Herrmann International (2016)

In summary, Figure 5.6 represents a schematic composite group of all the preferences in the four quadrants of the brain, as displayed by the 13 students in the programming classroom. They show a strong preference for quadrants B, C and D whilst quadrant A is sparsely represented. This figure is *not* representative of all the profiles in the programming classroom. A complete class profile might give a better representation.

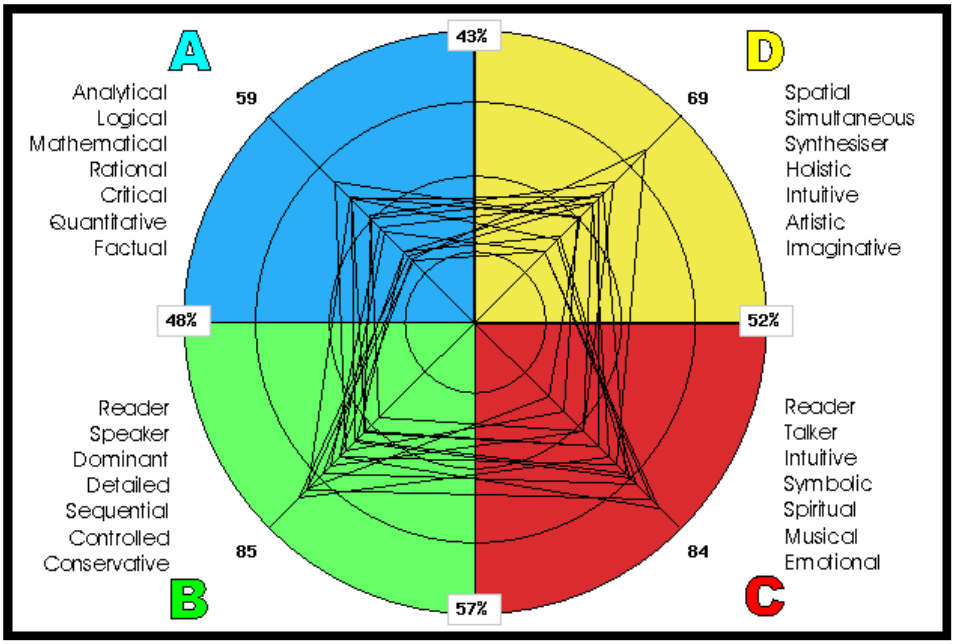


Figure 5.6: Composite group of participants' preferences
Source: Herrmann International (2016)

The understanding gained from this result revealed that participants' strong preferences (B and C), which were well represented in the class, correspond with my own strong preference. It was necessary to be flexible, as far as instructional planning was concerned, in an effort to address *all* quadrants of the brain and *not only* the well-represented preferences. The instructions were therefore designed to include quadrants A and D which involve the logical, analytical, mathematical, quantitative and rational as well as the holistic, artistic, imaginative and synthesising of mental processes, respectively. These quadrants (A and D) are necessary for the development of programming skills in the learners.

Therefore, I became aware that I had to adapt my teaching and learning to be more situated in quadrants A and D through exposing my learners to series of programming activities during class. It is expected that participants, who are *not* dominant in A and D, will experience conflicts (Herrmann, 2009). Therefore, I supported the planning and organisation of work elements from the B quadrant with cooperative and expressive work elements extracted from the C quadrant. In this way all four quadrants of the brain were involved in the learning process. Instruction and learning opportunities were designed to cater for the four quadrants of the brain, as described in section 5.3.

5.2.2 Designing instruction to facilitate learning

The designing of instruction commenced as soon as my learning approach was determined. I kept in mind that the programming classroom would contain diverse learners whose

preferences would *not* match my own. I ensured that teaching resources, including computers and projector which are readily available in the computer laboratory, were in good condition and that they had been serviced by a computer laboratory technician who had also installed the Scratch software on the computer systems. With permission from the head of the department (computer science), all non-functional computers were removed from the laboratory and we jointly prepared the room for the practical. While gathering and checking resources, I realised that the whiteboard in the computer laboratory was not suitable for teaching. I liaised with the head of the home economics department who provided the laboratory with a new whiteboard.

The textbooks used for teaching and learning were the "IT is gr8! @ Grade 10" provided by my supervisor, and a QBASIC module jointly prepared by myself and other computer science lecturers. The National Commission for Colleges of Education (NCCE) curriculum guideline for teaching computer science (NCCE, 2012) was also used in planning for teaching. As part of my preparation I learned Scratch programming. This process took some months and was affected through personal self-study and the solving of series of problems from the text. I had to familiarise myself with Scratch programming as it had been newly introduced to me by my supervisor. With these resources in place, a constructive alignment of topics (in both procedural and visual programming) was planned in line with their intended learning outcomes, assessment activities and allocated time (Appendix B 1 and Appendix B 2). I planned the instruction for each lesson in such a way that it provided a balanced set of learning experiences to all students. I asked my colleagues questions if I did not understand and I valued each piece of advice given. Their advice helped me to grow professionally. I also studied other textbooks where practical examples were used. For QBASIC programming, the intended learning outcome was designed as stipulated in the NCCE curriculum. The cognitive terms of the revised Bloom's and SOLO taxonomies were used to design programming assessment tasks with guidance from additional writings by the authors of the taxonomies (Clear et al., 2008b; Thompson et al., 2008; Whalley and Kasto, 2013). The outcomes and assessment tasks for Scratch programming have already been stipulated in the 'IT is gr8! @ Grade 10' teacher's guide. No modifications were made.

The profile results of the participants, as depicted in Figure 5.5, were further contextualised within instructional planning and based on the whole brain approach and constructivist method of teaching and learning. I first looked at *how* instructions within the whole brain will be planned by searching for practical examples from authors such as De Boer et al. (2013) who had used it in their own context. The knowledge I gained from them led me to produce the whole brain instructional planning for programming (*q.v*. Figure 5.7).

For A quadrant, all students were exposed to fact based lectures, research findings from the internet and school library and the use of presentation slides to deliver instruction. For the B quadrant, students were exposed to different programming problems accompanied by steps to follow as sourced from the learning manual, and sometimes verbally communicated by the teacher.

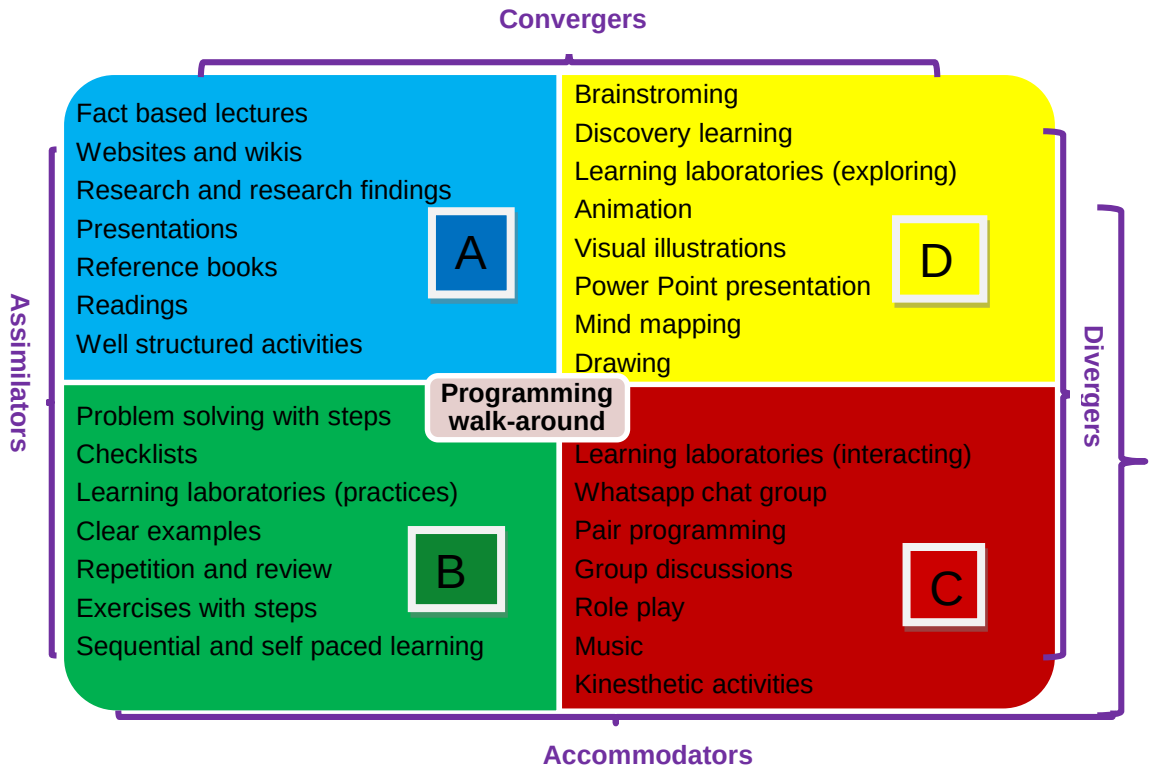


Figure 5.7: Whole brain teaching and learning walk around for programming

They used observation checklists to navigate their cooperative and teamwork abilities during project work. At the end of each lesson, I ensured that the lesson was reviewed so that students could learn according to their own pace in situations where they could be alone. Activities from quadrant C, incorporated into instructional planning, ensured interaction amongst groups during programming exercises and practise. I also created a WhatsApp group which presented a forum for feedback, questions and general interaction between teacher and students and amongst the studies themselves. They set about solving programming problems in pairs and discussed issues and problems in groups. In this way the students produced musical notes for programming projects as well as engaging in kinesthetic activities when I observed that they were too tired to concentrate. Group D activities used in planning instruction included: brainstorming in the classroom, learning through discovery during programming exercises, exploring the programming environment, designing animations and visual illustrations of programming problems using flowcharts. Other activities included drawing and mind mapping.

Additionally, the participants were placed into four homogeneous groups, except for one converging learner who joined the assimilating learners. The use of KLSI in this study guided the placement of participants into groups. After the HBDI profiles had been obtained, students were regrouped and placed in groups with other students whose profiles were based on the KLSI (see Figure 5.7). Potgieter (1999) established that the Kolb's learning approach does not strictly address the left *or* right brain but combines both brain hemispheres (*q.v.* Chapter 3). Hence, diverging learners were grouped with either similar quadrants in each group. The assimilators were regrouped with students whose quadrants were either B or C and D quadrants. There were no accommodators in the programming classroom. Understanding gained from the students' profiles helped in the design and facilitation of instructions within a whole brain and constructivist learning environment. This was designed using the whole brain walk around (*q.v.* section 3.2.3) for programming. The walk around was continuously adjusted based on student views, contextual issues and my reflections at the end of each lesson.

Using the whole brain method of facilitating learning, the programming instructions were planned by combining different learning activities, situated within each instruction, as the topic required. Table 5.1 gives a summary of the planned instruction with the dates, programming language, duration, number of students present in the class and the whole brain approach used. I see the outcome of this planning as my own living educational theory which can be useful in another setting. The constructivist learning activities which were used during each classroom session were also included. In addition, I prepared PowerPoint slides with the corresponding authentic assessments students were required to do for each programming class. The following sections (5.2.2.1 and 5.2.2.3) present feedback received from students and a non-participant observer on how I have facilitated learning and how the students contributed to learning (section 5.2.2.2).

Table 5.1: Instructional plan for programming during the first cycle action research

Date and Lessons (L)	Programming Language	Topic taught	Duration	No of students	Whole brain approach	Quadrants	Constructivist learning activities in the classroom
20/01/2016	Nil	Consent letters and familiarisation.	2 hrs	16	Nil		Learner control, social interaction, teacher as facilitator.
27/01/2016 (L1)	Nil	Basic concepts of computing.	2 hrs	18	Powerpoint presentation, visual illustrations, group discussions, clear examples, websites and wikis.	A, B, C and D	Collaborative learning, problem-solving, teacher as facilitator, social interaction, meaning construction, authentic activities, multiple perspectives, situated learning, consideration of errors.
3/02/2016 (L2)	Scratch	Solving algorithm with problems.	2 hrs	20	Powerpoint presentation, lecture, group work, poem, group presentation, exercises with steps.	B, C and D	Collaborative learning, authentic activities, teacher as facilitator, social interaction, meaning construction, multiple perspectives, problem-solving, knowledge construction, cooperative learning (think pair share).
10/02/2016 (L3)	Scratch	Algorithm with calculation.	2 hrs	23	Brainstorming, self-paced, group work, exercises with steps.	B, C and D	Collaborative learning, authentic activities, teacher as facilitator, social interaction, meaning construction, multiple perspectives, problem-solving, knowledge construction, cooperative learning (jigsaw).
17/02/2016 (L4)	Scratch	Introduction to Scratch programming environment.	2 hrs	23	Powerpoint presentation, animations, practical (exploring and interacting), group work, visual illustrations, role play.	C and D	Collaborative learning, authentic activities, teacher as facilitator, social interaction, meaning construction, multiple perspectives, problem-solving, knowledge construction, scaffolding, team teaching, learner control, group learning.
24/02/2016 (L5)	Nil	Free lecture / filling of HBDI form.	1 hr	13	Nil		Nil
2/03/2016 (L6)	Scratch	Algorithms (decision and repetition).	2 hrs	20	Powerpoint presentation, Lecture, brainstorming, animations, practical (practice), projector, lesson review, WhatsApp, group work.	B, C and D	*As above (L4) with rubric, cognitive conflict, learner control, disequilibrium, assimilation.

9/03/2016 (L7)	Scratch	Repetition	2 hrs	22	Powerpoint presentation, practical practice (interacting), group work, projector, WhatsApp, animations.	B, C and D	*As above (L4).
23/03/2016 (L8)	QBASIC	Project presentation (data types), variables, looping and Test of individual concepts.	2 hrs	22	Group presentation, music, mind mapping, research, wikis and websites, drawing, WhatsApp, reference book, lecture, practical (practice).	A, B, C and D	*As above (L4) with cooperative learning.
30/03/2016 (L9)	Scratch QBASIC	Operators and expressions. Introduction to QBASIC programming, operators, keywords, data types and running simple programs.	2 hrs	15	Lecture, practical (practice), self-paced, projector, sequential exercises, WhatsApp.	A, B and C	*As above (L4) with cooperative learning, pair programming.
1/4/2016 (L10)	QBASIC	Expressions and test (Interim Test 1).	2 hrs	18	Powerpoint presentation, self-paced practical (practice and interacting), problem-solving with exercises.	B, C and D	*As above (L4).
2/4/2016 (L11)	QBASIC	Working with variables, looping, running of programs and tests (Interim Test 2 and 3).	6 hrs	21	Practical (practice), self-paced exercises, problem-solving with exercises.	A, B, C and D	*As above (L4).
7/4/2016 (L12)	QBASIC	Control structure and subprograms (Final Test).	4 hrs	19	Lecture, practical (practice), self-paced, powerpoint presentation, sequential exercises, kinesthetic activities, group work.	A, B, C and D	*As above (L3).

* *As above* means the constructivist activities were repeated in the classes where the (*) appears in the column.

5.2.2.1 Feedback on teaching and learning

At the end of the teaching and learning, participants assessed the way in which I facilitated learning using the whole brain feedback survey (*q.v.* Appendix A 16). Their responses to the survey are illustrated in bar charts which shows how learning was facilitated, initiated and maintained. As shown in Figures 5.8 to 5.10, the x-axis features letters a, b, c or d which represent the information the participants responded to in the questionnaire. Each discussion followed the progression of the letters on the chart. The Y-axis represents the participants' responses in percentages. This section commences with the description of how learning was facilitated.

Learning facilitation

As illustrated in Figure 5.8, 'item a' shows that 50% almost always and 25% frequently noted that the teacher facilitated learning by showing a strong interest in programming and learning tasks while 13% occasionally noted. In 'item b', 63% almost always and 38% frequently preferred the teacher's audible expression during teaching. In addition, 38% of the participants almost always and frequently noticed the teacher's method of teaching by encouraging insight into the importance of programming problems and related problems while 25% said they hardly ever saw the teacher doing this. In like manner, 88% of the participants always and frequently (13%) saw the teacher providing programming sessions that are encouraging and lively in 'item c'.

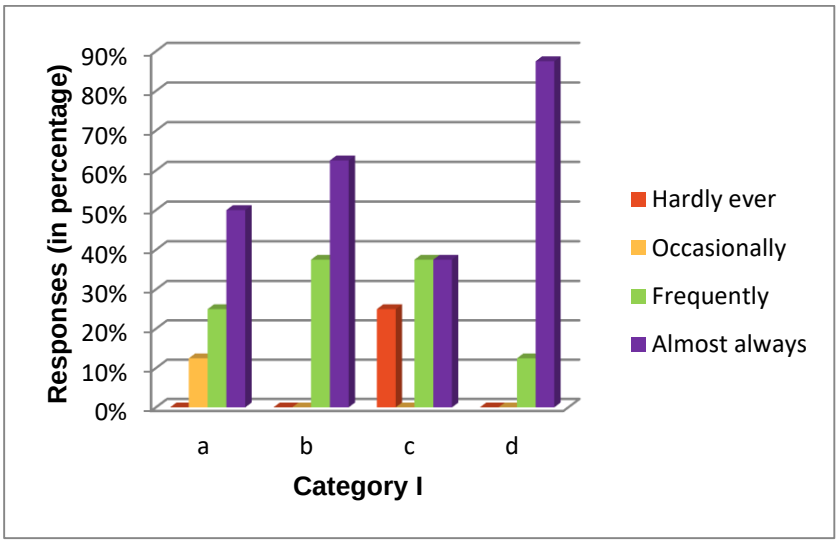


Figure 5.8: Learning facilitation feedback

Learning initiation

In Figure 5.9, 'item a' shows that the participants supported that the teacher almost always (50%) and frequently (38%) initiated learning by creating a climate that encourages learning. Likewise, for 'item b' 50% indicated that the teacher frequently stated the objectives and learning outcomes of each classroom session while 25% said the teacher did this either almost always or occasionally. Similarly, 'item c' shows that the teacher frequently (38%) and almost always (25%) linked programming problems to real life. The remaining 25% noted that they occasionally, or hardly ever (13%), saw the teacher doing such.

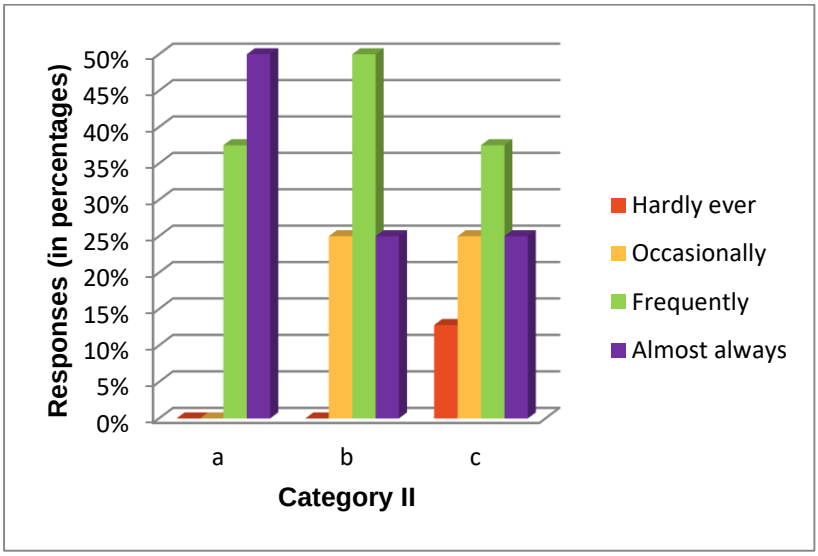


Figure 5.9: Learning initiation feedback

How did the lecturer maintain learning?

Results from the bar chart in Figure 5.10 shows that 50% of the participants almost always and frequently (50%) noted that the teacher maintained learning by encouraging lecturer-student discussion which enabled them to always ask challenging questions regarding programming in 'item a'. They also said the teacher almost always (75%) and frequently (25%) encouraged them to construct their own understanding during programming in 'item b'. Accordingly, in 'item c', 50% of the students said the teacher almost always, frequently (25%), and occasionally (13%) helped them learn by using other means and not solely based on their comfort zones. In contrast, 13% hardly ever left their comfort zones. On whether teacher encouraged students to freely express themselves in 'item d', 50% attested that the teacher almost always did this, while 38% frequently and 13% occasionally saw the teacher allowing this to occur in the programming classroom. Similarly, 50% noted that the teacher, in an effort to ensure self-regulated learning, almost always and frequently encouraged them to engage in critical thinking and reflect on their learning in 'item e'. Also,

the teacher always (88%) and frequently (13%) created opportunities for the students to program cooperatively in a group in 'item f'.

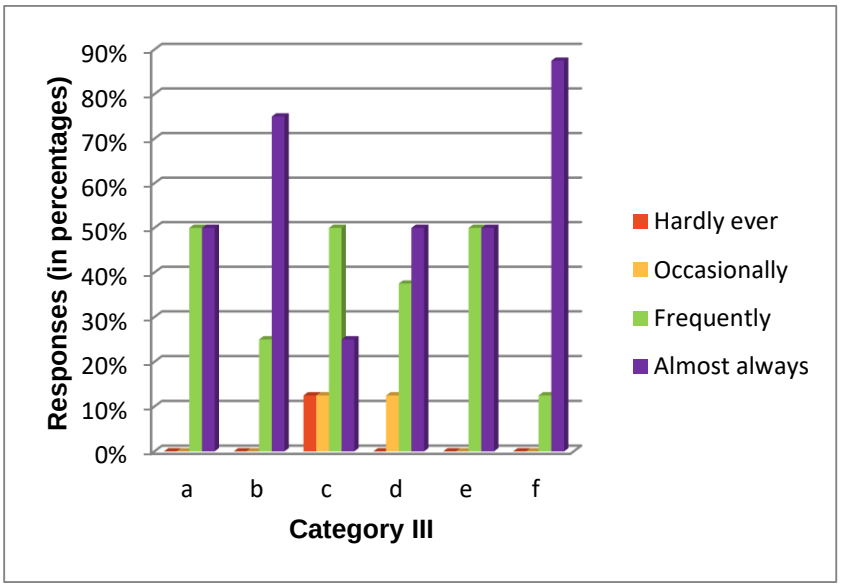


Figure 5.10: How the lecturer maintained learning

5.2.2.2 Students' perception regarding their learning contribution

The participants assessed themselves. The results presented in this section depict their contribution to their own, and others' learning of programming during the semester, as well as their learning application and strategy. As shown in Charts 4 to 6, the x-axis features letters a, b, c or d which represent the information the participants responded to in the questionnaire. The y-axis represents participants' responses in percentages. This section commences with students' contribution to their own, and others' learning.

Students' contribution to their own and others' learning

Figure 5.11 depicts students' responses on how they contributed to their own and other students' learning in the programming classroom. The result in 'item a' shows that the majority of students almost always (63%) and frequently (25%) showed strong interest towards the learning of programming concepts in the classroom while 13% said they occasionally showed their interest. In addition, they contributed to their learning in 'item b' by almost always (50%), frequently (38%) and occasionally (13%) expressing themselves well in the programming classroom. Similarly, in 'item c', 50% frequently, 25% almost always and the remaining 25% occasionally gained insight into the importance of each programming concept and its related problems. The students also contributed to their own, and others' learning, by ensuring almost always (75%), frequently and occasionally, 13%

respectively, that they participated in the programming classroom in such a manner that each lesson was lively and encouraging in 'item d'.

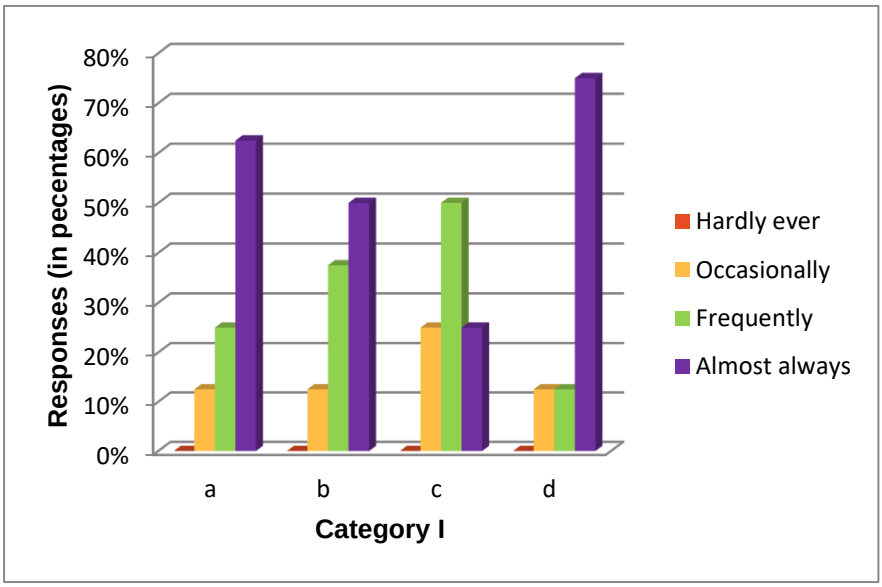


Figure 5.11: Students' contribution to learning feedback

Students' learning application

Figure 5.12 represents the way in which students applied programming learning to real-life situations to encourage deep learning. Firstly, their response shows that 63%, 25% and 13% respectively responded that they occasionally, frequently and almost always, co-created a climate conducive to the deep learning of programming in 'item a'. Secondly, the students almost always (50%), frequently (25%) and occasionally (25%) linked their learning programming to real-life experiences in 'item b'. Thirdly, in 'item c' only 50% frequently, almost always (13%) and occasionally (13%) formed a mental representation of the multidimensional nature of each programming topic studied while 25% indicated that they hardly ever formed a mental representation.

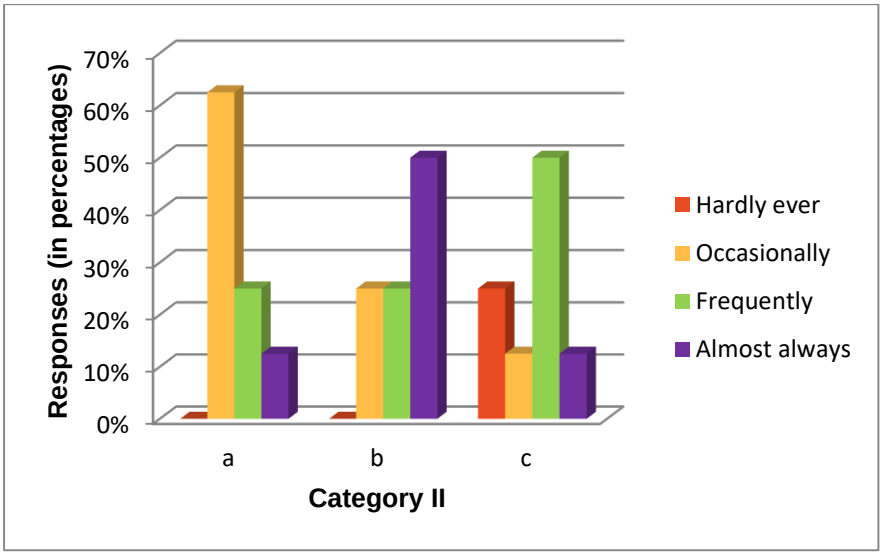


Figure 5.12: Students' learning application feedback

Students' learning strategy

Figure 5.13 illustrates students' learning strategies in the programming classroom. The figure shows, in 'item a', that students frequently (75%) and occasionally (25%) developed an enquiring mind in the programming classroom through lecturer-student discussions. As a result, 38% frequently, 25% almost always and 25% occasionally constructed an understanding of programming concepts and activities done in the class in 'item b'. While seeking opportunities for improving their learning approach, 38% frequently, 25% occasionally and 25% almost always moved out of their comfort zones in 'item c'. Similarly, in 'item d', most students noted that they frequently (63%) and almost always (38%) expressed themselves freely and openly in the classroom. This, however, changed their attitude to learning programming such that 38% almost always, 25% frequently and 25% occasionally reconsidered their former attitude to learning in 'item e'. The remaining 10% noted that their attitude to programming never changed. This, however, may have explained in 'item e', why 63% almost always and 38% frequently had the opportunity to gain a better understanding of themselves. Discussing responses 'g' and 'h' together, 75% almost always and 25% occasionally developed a better sense of responsibility towards the learning of programming in 'item f'. They, therefore, exhibited this responsibility further by noting that 38% almost always and 38% occasionally contributed to their peers' learning by helping them find solutions to programming problems while 13% hardly ever did. Also, there was a community of practice while learning programming because 25% frequently, and 38% almost always and occasionally took part in cooperative learning activities in 'item g'.

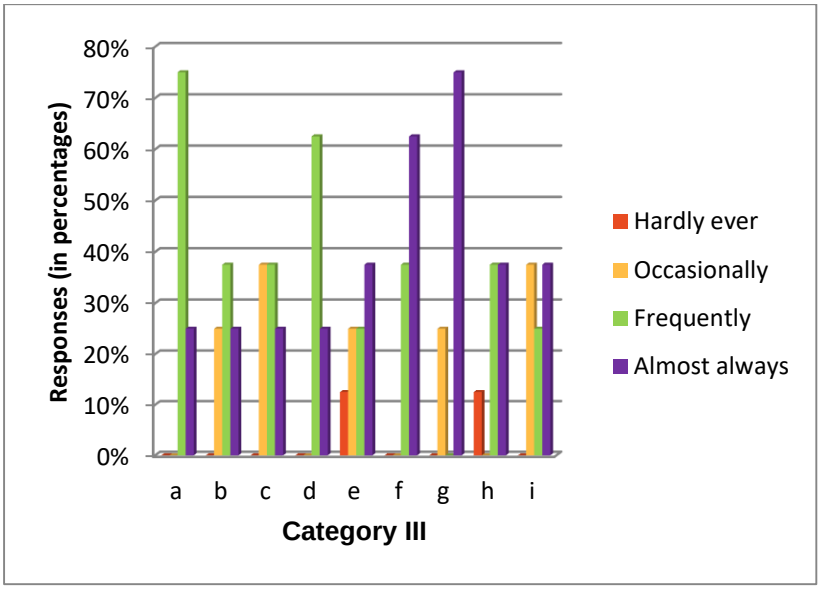


Figure 5.13: Students' learning strategy feedback

5.2.2.3 Feedback from non-participant observer

During the research process a non-participant observer (Appendix A 18), who also teaches computer science in an affiliate university to the college, observed my teaching and student learning in programming. He completed a feedback questionnaire, similar to the one given to the participants. I modified the questionnaire slightly to include *supported*, *not supported* and *not observed*. From the feedback, he noted that I inspired my students by showing passion for the teaching of programming. He supported that I promoted learning by stressing the significance of each programming concept taught as well as providing active and inspiring learning opportunities. He stressed that I initiated programming teaching by first stating the purpose and learning outcomes of each lesson. Therefore, he concluded that the programming classroom created a climate that promoted deep learning where learning was linked to real-life situations. He also stressed that I encouraged a form of warm relationship between myself and the students on the discourse of learning. This, in the long run, allowed the students to develop an inquisitive mind. Other areas where he noted that I maintained learning are: through the fostering of critical thinking, encouraging self-reflection and cooperative learning. Providing scaffolds to the students and providing opportunities for learning flexibility during the teaching and learning were supported. To conclude, he wrote a descriptive summary of his observation and stated, *"in her first contact with the students, the content of a card tagged 'Classroom Rules' was discussed with the students which she later posted on classroom doors. She encouraged the students to ask questions during the classes. Lateness to the classes by the students was discouraged. She encouraged the students to interact with one another during the lessons to share knowledge using peer*

group tutoring. She supported the classes with both class work and assignments. She was totally in control of the classes".

5.2.3 Observation of classroom processes

The previous section reported on how the instruction was planned based on the four quadrants of the brain to suit the facilitation of programming in my classroom. This section discusses the observation of classroom processes that took place while facilitating learning. The observations took place from 20 January to 7 April 2016 on every Wednesday of the week with a maximum of 2 hours per observation. The observations were conducted in the computer science laboratory (*q.v.* Figure 5.14) which is situated in the school of science building. The laboratory comprised 50 computers of which 30 were non-functional. There were also two non-functional air conditioners, four functional but noisy standing fans, four ceiling fans, two fluorescent bulbs and six halogen bulbs. There was a big pillar, supporting the building, at the centre of the laboratory which prevented participants sitting at the back of the laboratory as they could not view the whiteboard. There was a projector slide at the front of the computer laboratory while the projector itself was placed on one of the computers in the middle row. In front of the laboratory were two whiteboards, one old and one new, which had been borrowed from another department for the research. The flooring of the laboratory was neither rugged nor tiled but painted.

The distance between the computer tables and the supporting chairs was very limited and participants could hardly move freely. The seating arrangement of the participants is represented in Figure 5.14. Two laboratory technologists, who manage the laboratory, also attend to technical issues. These two individuals compute the results of students for each department and check result for students. This task lead to a constant movement of people *in* and *out* of the laboratory. I controlled this influx of people by affixing a laminated notice to the door which read 'Teaching in progress, check back by 2pm'.

Data for the observation were collected using video, anecdotal notes and structured observations. Eleven open-ended video observations recorded classroom process, but in consideration of space, I present a summary of only three observations with an interim analysis of each observation. A comprehensive observation codes of all lessons in the two cycles can be found in Appendix C 2 and Appendix C 3. The observations focused on the participants because in the development of programming skills, the learners are required to construct understanding regarding a broad range of programming topics. Attention was paid to students' social interactions, shared meanings and behaviours *as lived* in the classroom.

The participants were observed individually, and in groups, during classroom processes. I will refer to individual students by referring to their groups, for example student A1, B2, C1, D1, E3 and F2. I also reviewed my teaching strategies, especially the facilitation of learning based on the whole brain. This was done to monitor my professional development.

Figure 5.14: Seating arrangement of participants in the computer laboratory

I conducted an interim analysis of the data collected, firstly, by watching the recorded video at the end of every class. This was done in an effort to capture and reflect upon: classroom processes, student interactions in groups, participants' behaviours and concepts learnt. The structured observations and anecdotal notes were validated with the video to check for connections. This helped me to make necessary changes to planned instruction and teaching strategies for the next lesson.

5.2.3.1 Classroom observation and analysis of Lesson 3

The concept of the algorithm was taught in the third lesson. From the teaching activity of Lesson 3, fourteen codes were identified. These codes are: positive emotional climate (PEC), seeking help (SH), regulation of cognition (RC), student behaviour (SB), lateness (LT), no learning material (NLM), knowledge of flowchart algorithm (KOF), electricity problem (EP), meaning construction (MC), cooperative learning (CL), Assignment (AS),

group learning (GRPL), learning challenge (LC) and whole brain teaching (WBT). Table 5.2 presents the breakdown of the analysis, with its and corresponding categories.

Table 5.2: Interim analysis of Lesson 3

Subcategories	Codes	Categories
Positive emotional climate Seeking help Regulation of cognition	PEC SH RC	Self-regulation
Student behaviour Lateness No learning material	SB LT NLM	Student behaviour
Knowledge of flowchart algorithm	KF	Scratch and QBASIC knowledge gained
Electricity problem	EP	Contextual problem
Meaning construction	MC	Constructivist learning strategy
Cooperative learning Assignment Group learning	CL AS GRPL	Constructivist teaching strategy
Learning challenge	LC	Learning challenge
Whole brain teaching	WBT	Whole brain teaching strategy

Some of the students arrived late (LT). The concepts of the abstract and concrete algorithms were extensively explained and group work was given to the whole class (GRPL). They were asked to create a poem showcasing the steps to solving problems (WBT). Students in groups A and E jointly produced the poems on algorithm steps (CL), they danced and were pleased about what they were doing (PEC). Students F1 and F2 were confused (LC) and looked for help when their group did not understand the class activity (SH). Following this, they were able to participate in the class activities toward the end of the class (RC). Students in groups C, D and F were observed playing during the creation of their poem (SB) on algorithm steps. Because the groups did not take the learning seriously, students D1, D2 and C1 could not present the song well. From the poems presented, students could list the steps in solving an algorithm from the poem (MC). In addition, only five students (15%) had the learning material (NLM). During the class, there was a power cut and the projector switched off (EP). As a result, the teacher asked the students to share their handouts to visualise the explanation of concepts of flowcharts and algorithm. Students initially showed an incomplete knowledge of flowcharts, but when the teacher revised the concepts, students could link actions to the flowchart symbols after solving series of problems (KOF). Students were given an assignment to be completed for the following week (AS).

5.2.3.2 Classroom observation and interim analysis of Lesson 8

Lesson 8 focused on the teaching of repetition in Scratch programming. From the lesson observation discussed below, sixteen codes were identified. The codes are: Lateness (LT),

student behaviour (SB), student enrollment (SEMT), regulation of cognition (RCG), valuing learning (VL), seeking help (SH), positive emotional climate (PEK), meaning construction (MC), student engagement (SE), individual learning (IL), revision (REV), group cooperation (GRC), no group cooperation (NGRC), scaffolding by the teacher (SCT), scaffolding by the student (SCS) and brainstorming (BRST). Table 5.3 gives a summary of the analysis with the corresponding categories.

Table 5.3: Interim analysis of Lesson 8

Codes	Acronyms	Categories
Lateness Student behaviour	LT STB	Student behaviour
Student enrollment	SEMT	Contextual problem
Positive emotional climate Seeking help Valuing learning Regulation of cognition	PEC SH VL RGC	Self-regulation
Meaning construction	MC	Constructivist learning strategy
Student engagement Individual learning Revision	CL AS REV	Constructivist teaching strategy
Group cooperation No group cooperation	GRPC NGRC	Group benefits and negative impact
Scaffolding by teacher Scaffolding by student	SCT SCS	Scaffolding
Brainstorming	BRST	Problem-solving

The topic of repetition was revised and students were then given individual class activities (practical) to do for 10 minutes (IL). Some students were late (LT). I also noticed another new student who had just joined that day (SEMT). Students' problem-solved programming problems from Scratch and constructed meaning on repetition (MC). While working on practical on repetition, I guided the students (SCT). Student B1 also guided student B2 during the practical (SCS). Group A students asked me for help (SH) and student E2 also stood up from his seat to ask for clarification concerning the class activity from student D1 (RCG). Student B1 was confused, he could not solve the problem (LC). I ascribed this to his truant behaviour (SB). Then his partner, student B2, guided him in the activity (SCS). Even though I said they should work in groups, students in group D and C still worked in pairs (NRGC). I then noticed that group E students were also struggling with the problem, and I asked the class to pause for a little brainstorming (BRST) so that other groups could be helped. Students were guided where needed. After the students had completed numbers 1 to 6 in groups, I solved queries together with the students (SE). After completion of the practical, students in group A applauded when they finally arrived at the correct answer after many trials (PEC). Students F1, F2 and C4 stayed after class to complete their class work

(VL). Figure 5.15 presents samples of artefacts produced by the students during class practical.

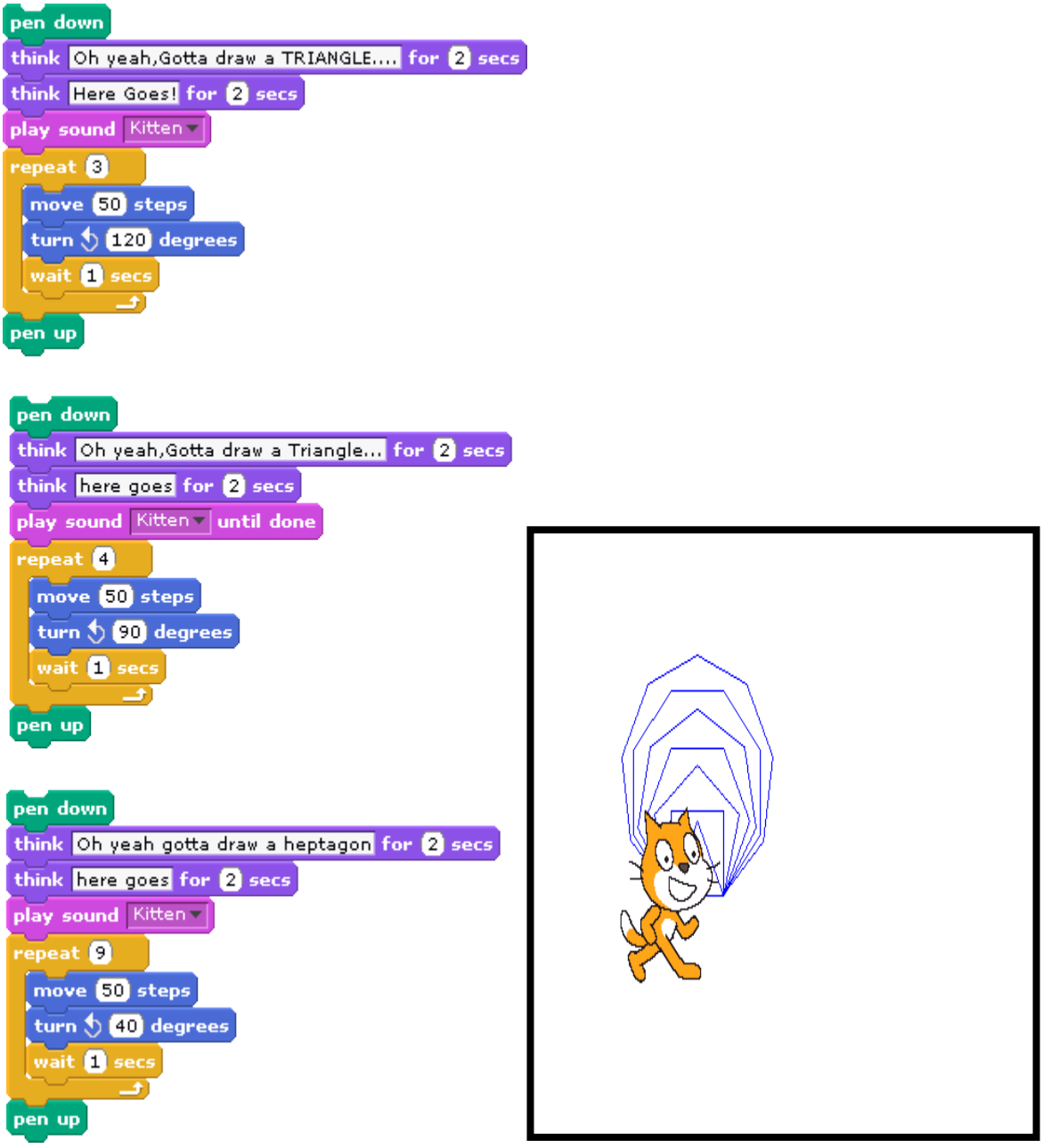


Figure 5.15: Samples of students' artefacts on repetition

5.2.3.3 Classroom observation and interim analysis of Lesson 12

Lesson 12 was the last class observed and it focused on writing simple programs in QBASIC. From the lesson observation, eight codes were identified. These codes are: knowledge of flowchart algorithm (KOFA), knowledge of expression (KOE), student behaviour (SB), lateness (LT), student engagement (SE), whiteboard (WB), electricity problem (EP) and scaffolding by the teacher (SCT). Table 5.4 gives a summary of the analysis with the corresponding categories.

Table 5.4: Interim data analysis of Lesson 12

Codes	Acronyms	Categories
Knowledge of flowchart algorithm Knowledge of expressions	KOFA KOE	Programming knowledge gained Knowledge of expression
Student behaviour Lateness	SB LT	Student behaviour Lateness
Student engagement	SE	Student engagement
Whiteboard Electricity problem	WB EP	Contextual factor
Scaffolding by teacher	SCT	Scaffolding

Lesson 12 was a weekend class which was jointly arranged by the teacher and students. There was power cut on campus (EP) and so students used my laptop to practise. Since the lesson could not be projected, the students complained that they could not see the whiteboard from the back (WB) and this resulted in many of them wanting to sit in the front of the class to see the program being run. Before the practical, a class activity where students represent a Scratch program in a QBASIC expression was given. It was observed that students applied the knowledge gained from Scratch to QBASIC, and the majority scored above average (KOE). After running a sample program on the laptop, I asked students to do class work and write a program manually (SE). After the solution was solved by the students, I asked them to represent the solution in a flowchart (KOFA). The class activity was disturbed by student F4 who was not concentrating (SB). At 15:06 another student (C4) made a nasty comment regarding one of the students (D2) saying “you will meet yourself as you sit in the class.” Her statement caused the whole class to laugh. The last activity involved a group class activity in program writing. Students were called at intervals to write and run their programs on the laptop (SE). Students were scaffolded along the line (SC). Figure 5.16 is a sample of a student’s program in QBASIC.

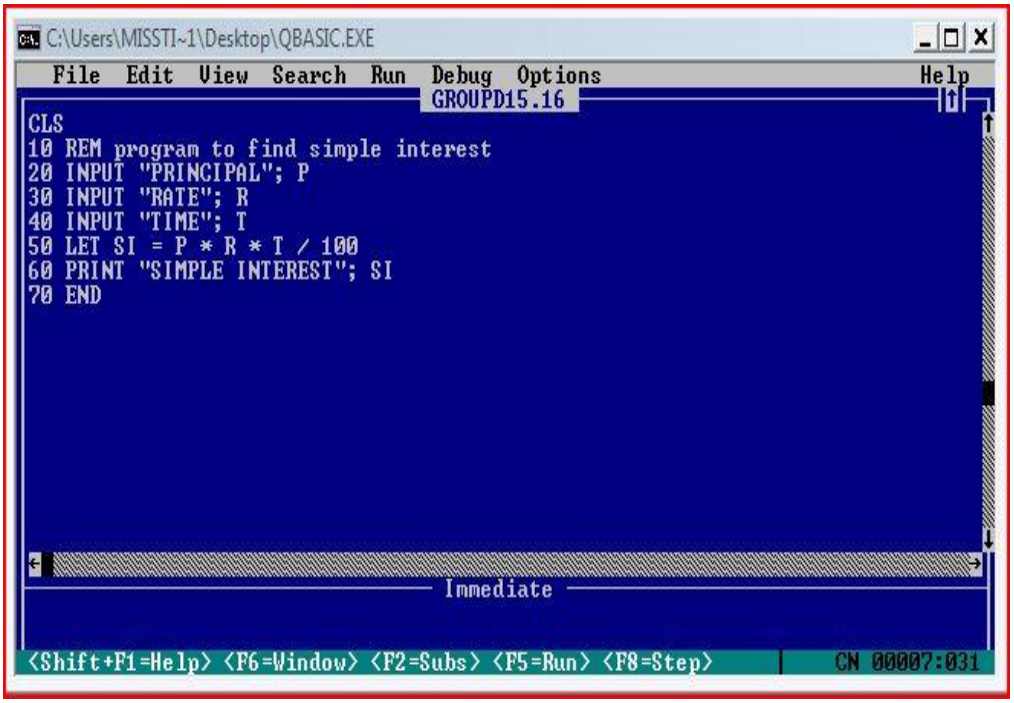


Figure 5.16: Screenshot of a program written by group D

The way in which categories become a sub theme, and finally themes, is further discussed in section 5.3.

5.2.3.7 Structured observation

Structured observation has been discussed in section 4.3.3. Here, results of different observations on students' behaviour in the programming classroom are presented. Two structured observations used during the semester are *observation checklist* and *tally sheet* They are hereby discussed below.

Observation checklists

The checklist presents results of students' behaviour during group work activities on programming (Appendix A 17). All the groups were observed at different times during the semester. Figure 5.17 presents the result of the collaborative activities within each group. Groups A, D and E brainstormed five times while groups B, C and F brainstormed four times during group work. All the groups stayed on task during the five observations, except for group F who stayed on task three times only. The groups always respected each other, except for group F members who only showed respect towards each other three times out of five observations. Groups A and E actively participated four times, groups B, D and G actively participated three times while group C participated actively only once.

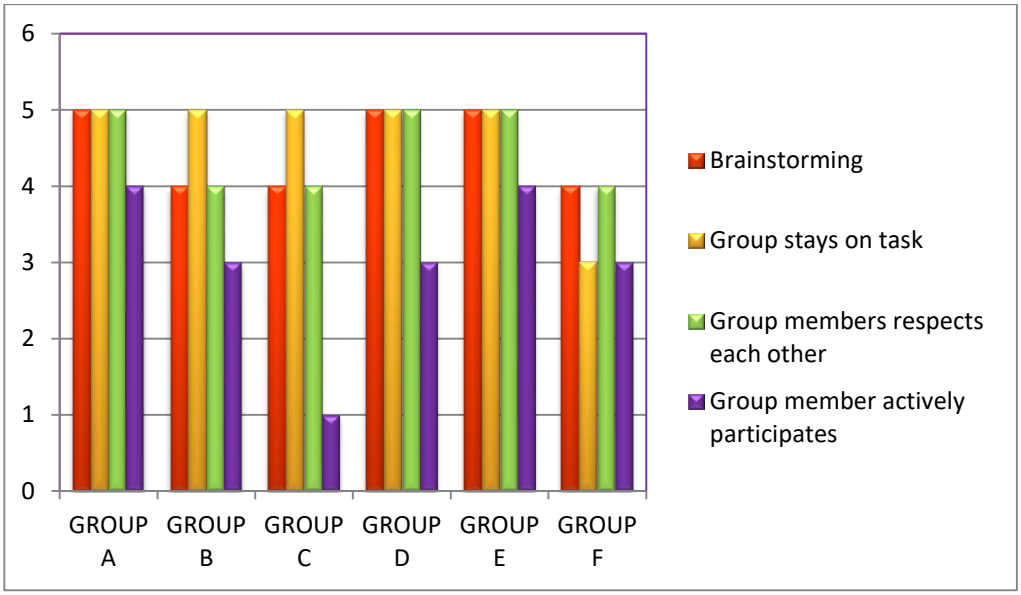


Figure 5.17: Collaborative group checklist

Tally sheet

The tally sheet in Table 5.5 shows the behaviour of five students observed during the session. Student A2 exhibited the desired behaviour when observed on two occasions but his behaviour was termed *undesirable* when he talked in class and kept his head on the table while the class activities were proceeding. Student D1, in one of the classes on Scratch programming, followed direction four times, raised his hand for attention and responded when I asked him questions. However, he shouted out twice and mumbled once during the first Scratch practice. Student F2 followed directions once, raised a hand for attention during Scratch practical and responded when spoken to. Student E3 followed direction and responded when spoken to. However, she exhibited undesirable behaviour by walking around when the class was in progress.

Table 5.5: Tally sheet of students' behaviour

Desired behaviour	**3/2/2016**	**9/32016**	**17/2/2016**		
Asking for help	A2	A2	D1	F3	E3
Following direction	√	√	√ √√√	√	√
Raising hand for attention	√	√	√	√ √	
Copying down solution	√	√			
Responding when spoken to	√	√	√	√	√
Undesirable behaviour					
Shouting out in class	√	√	√√		
Keeping head on the table	√	√			
Mumbling			√		
Not interacting					
Moving around					√

5.2.4 Artefacts

The artefacts were collected alongside observations regarding classroom teaching and learning, however, I decided to describe the results separately. The type of artefact collected were student-generated tests. The teacher created tests (formative) which were based on the programming constructs participants had been taught in the classroom. The questions were classified based on combined Bloom's and SOLO taxonomies (Appendix B 3). Each test questions was further mapped to the Block model (Whalley and Kasto, 2013). The essence was to determine the knowledge dimension and students' level of comprehension in programming. A synthesis of the classification can be found in
Appendix B 3. Five different types of tests (*q.v.* section 4.3.4) were administered, as discussed. These tests were: a test on individual concepts, interim test 1, interim test 2, interim test 3 and a final test. The following discussions further detail the test characteristics and students' performances in each test.

5.2.4.1 Test of individual concepts

The first test the students took on 23 March 2016 examined individual concepts (Appendix A 6), thus determining students' understanding of solution development constructs such as algorithm, flowcharts, repetition and the solving of simple programming problems. Twenty-two students attempted the test. Out of 25 obtainable marks, the result showed that only one participant scored above average while the remaining 21 participants scored below average. From test observations I could deduct that some of the participants could identify variables, define a stage and list the repetition structures in Scratch while some could not. These deficient students could not draw a flowchart for the algorithm nor identify repetition structure in small codes. Also, they were unable to list the program blocks the user could use to communicate with the user in Scratch programming. There was no correct identification of an abstract and concrete algorithm. I was not pleased with the test scores, but the results did, however, showed that I needed to rethink my teaching methods. Figure 5.18 gives a summary of students' test scores.

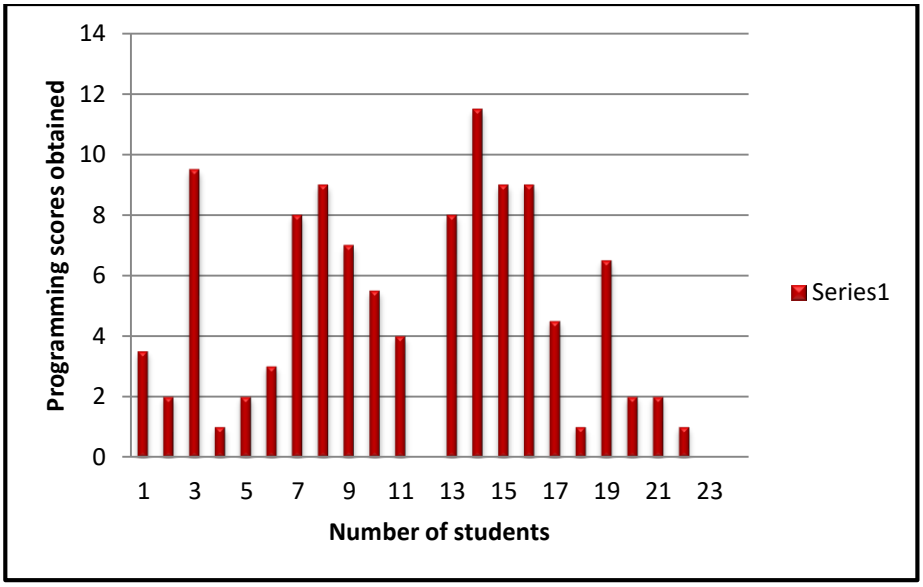


Figure 5.18: Students' scores in test on individual concepts

Students' program comprehension skills in the written test were also determined by finding the percentage correct answers of questions relating to tracing, writing and explaining skills type (Appendix B 3). Results showed that students developed 60% explaining skills, 32% tracing skills and 8% writing skills in programming (see Figure 5.19). The cumulative marks attached to the question differ. This may explain why the percentage score for explaining skills is higher than the tracing and writing skills.

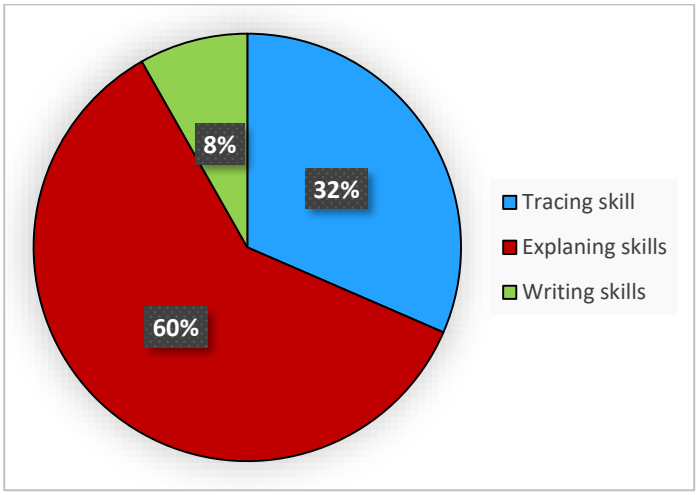


Figure 5.19: Students' programming skills on test of individual concept

5.2.4.2 Interim test I

The participants were exposed to the second test, the interim test I (Appendix A 7) which lasted 30 minutes on 1 April 2016. The test was conducted to determine students'

understanding of solution development, data types and program writing in QBASIC programming. Of the 24 students in the class, half scored above average and the other half scored below average. The test results showed that there was an improvement in their performance on the interim test, though some of the participants still performed low. Students developed 46% tracing skills, 34% reading and 20% writing skills (see Figure 5.20). The obtainable marks attached to tracing skills were double that of reading and writing skills. Simplifying the result further, one can say that students developed more reading skills than writing and tracing skills.

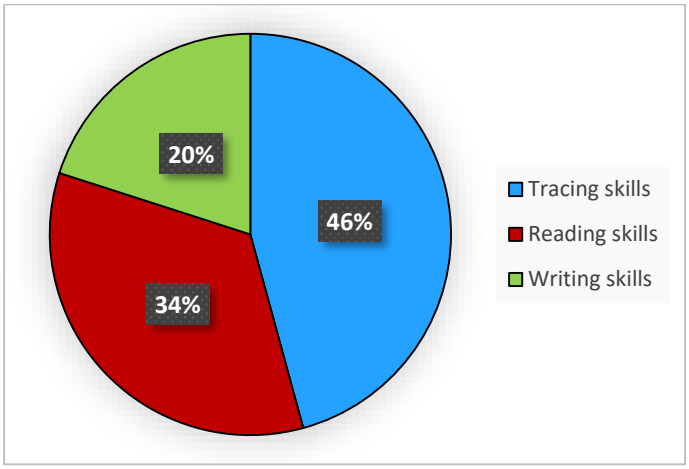


Figure 5.20: Students' programming skills on interim test I

5.2.4.3 Interim test II

Interim test 2 (Appendix A 8), was meant to determine students' understanding of expressions in QBASIC programming, and it lasted for 20 minutes. Twenty-two students attempted the test on 2 April 2016, and the result showed that only three students scored above average while the remaining 19 students scored below average. Figure 5.21 provides a summary of students' scores on programming expressions.

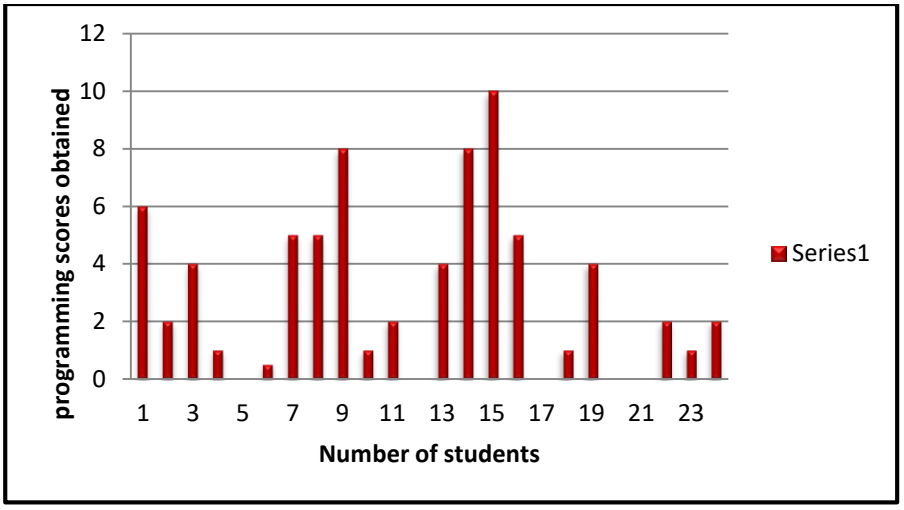


Figure 5.21: Students' programming scores on interim test II

Program comprehension was also determined. The results, as depicted in Figure 5.22, show that students developed 35% writing and explaining skills and 21% tracing skills. Test questions on writing skills, with its attached obtainable marks, were double that of tracing and explaining skills. Therefore, students developed more explaining skills than tracing and writing skills.

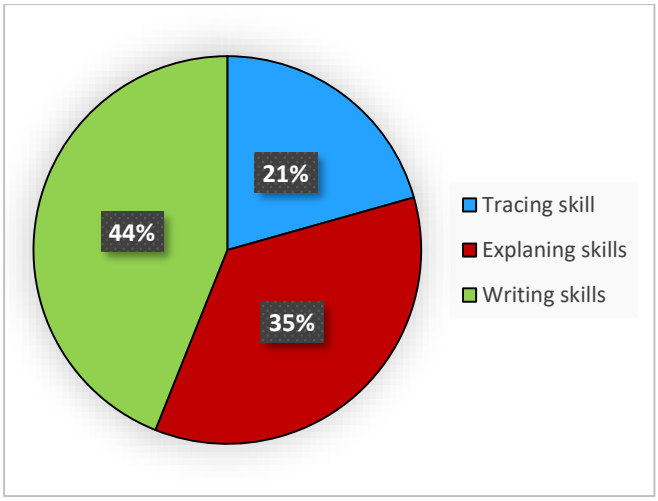


Figure 5.22: Students' programming skills in interim test II

5.2.4.4 Interim test III

Interim test 3 (Appendix A 8) was meant to determine students' program writing skills. Figure 5.24 presents a sample of a program written by students. Nineteen students attempted the test on 2 April 2016, and the result showed that out of 10 marks obtainable, only two students scored above average while the remaining 17 students scored below average. The bar chart in Figure 5.23 gives a summary of students' scores on programming writing.

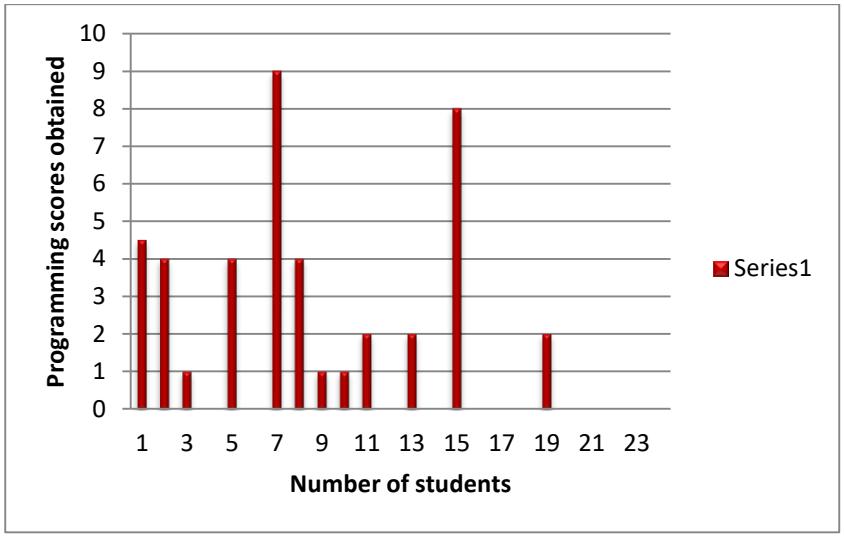


Figure 5.23: Students' scores on interim test 3III

A sample of the students' written program in interim test III is represented in Figure 5.24. The participant solved problems 1 and 2 of the question given. This figure shows that the student can write short programs in QBASIC. However, most of the participants attempted questions 1 to 3, while none of them attempted question 4. It seems as if they viewed question 4 as more difficult than the other questions. Another reason could be that question 4 involves conditions and the use of control structure commands which they had not mastered. In addition, a student might feel that he/she might not be able to solve the question within the limited time.

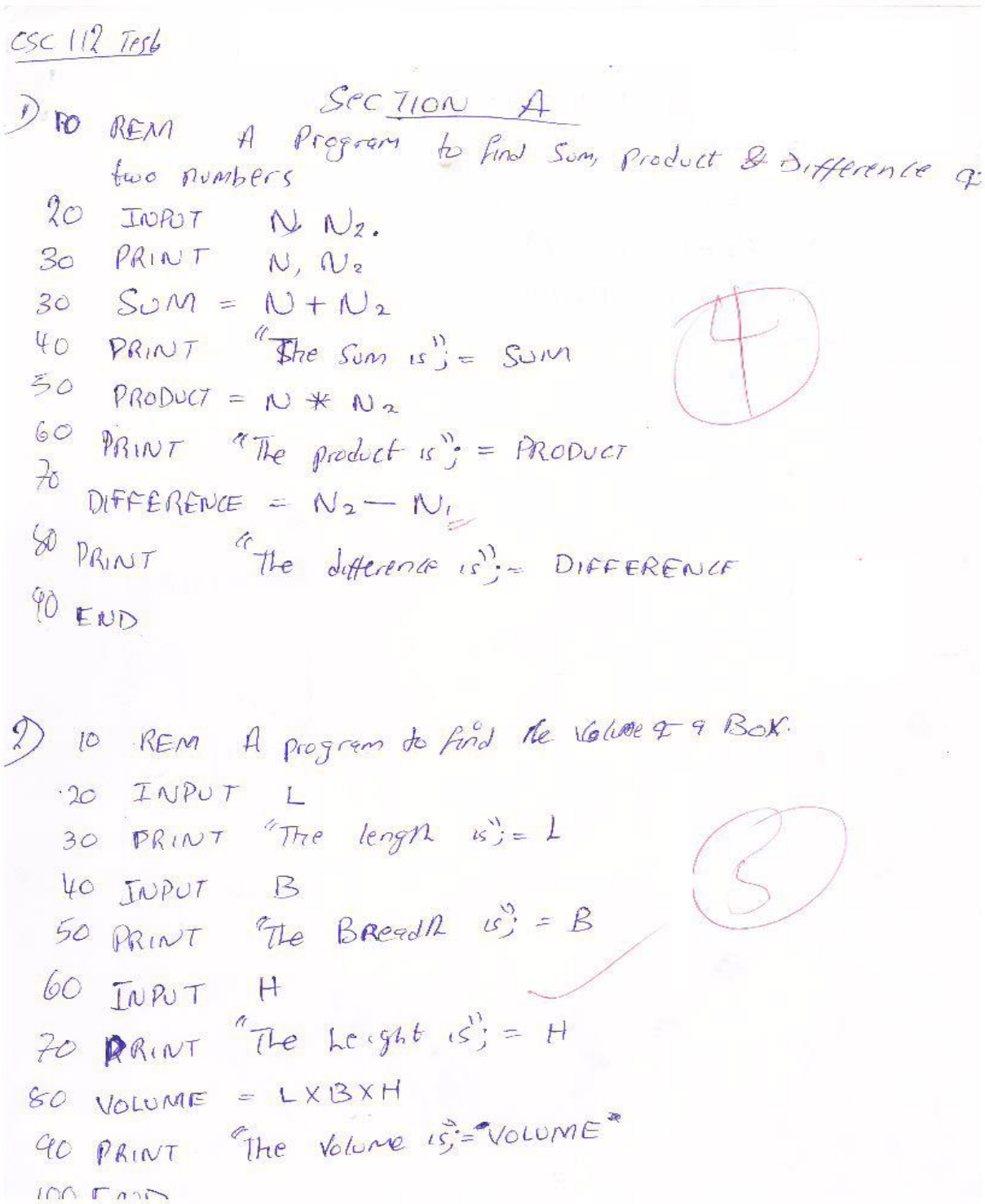


Figure 5.24: A sample of a program written in interim test III

5.2.4.5 Final test

The final test (Appendix A 9) comprised of an assessment of solution development, data types, program writing and explanation of concepts, and was done on 7 April 2016. Students did well on solution development and data types while they performed low in explanation of concepts and program writing. The pie chart (see Figure 5.25) shows the program comprehension skills of students in the final test. Students had 44% tracing skill, 26% explaining skill, 20% reading skill and 10% writing skill.

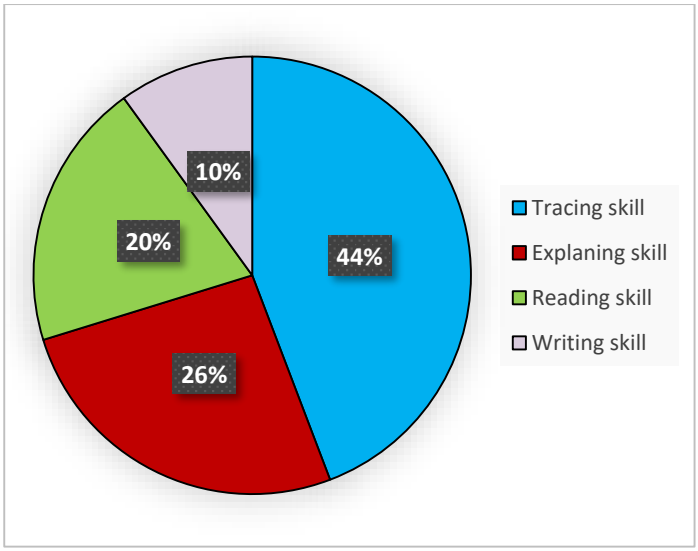


Figure 5.25: Students' programming scores on final test

Feedback on formative tests

At the end of the semester, a further attempt was made to determine the reliability of the test scores. The internal consistency of the tests was measured using Cronbach alpha which provides the correlation of each test item in relation to all the other tests, measuring the same concept (Cohen et al., 2011). An internal reliability test was conducted on the tests and all the items in each test were categorised as data types, program writing and solution development. The tests were analysed by a statistician at the University of Pretoria. The Cronbach alpha (*q.v.* Table 5.6) for data type questions was .808 which means the test items were reliable. Program writing has a reliability coefficient of .803 which means the test items were reliable and solution development, with a reliability coefficient of .340, shows the test items were not reliable (Pietersen and Maree, 2016) . This may explain why students did not perform well in the test of individual concept questions.

Table 5.6: Reliability statistics

	Cronbach's Alpha	N of Items
Data Types	.808	3
Program Writing	.803	3
Solution Development	.340	3

A further investigation was conducted into the test items. From the analysis in Table 5.7 below, there was a moderate positive correlation (.465) between interim test 1 - solution development and test of individual concepts. No significant correlation (.097) between final test - solution development and interim test 1. A moderate positive correlation (.402) exists between final test - solution development and test of individual concepts. None of these values is high, so even after deleting one of the items, the Cronbach alpha is still not close to 1.

Table 5.7: Correlations

		Test of individual concepts - Solution Development	Interim Test 1 - Solution Development	Final Test - Solution Development
Test of individual concepts - Solution Development	Pearson Correlation	1	.465*	.402
	Sig. (2-tailed)		.039	.109
	N	21	20	17
Interim Test 1 - Solution Development	Pearson Correlation	.465*	1	.097
	Sig. (2-tailed)	.039		.700
	N	20	22	18
Final Test - Solution Development	Pearson Correlation	.402	.097	1
	Sig. (2-tailed)	.109	.700	
	N	17	18	19

*. Correlation is significant at the 0.05 level (2-tailed).

I decided to re-design the test in the next cycle of the study to determine its reliability. The overall assessment results of participants were used as a point of departure for the interview. The participants interviewed were chosen based on the result of their performances in programming.

5.2.5 Interview with participants

The next section in the data collection was an interview with participants who had a *lived experience* in the learning of both visual and procedural programming. The selected participants were high, medium and low achievers. In effect, six participants were interviewed. The interview with the participants, as a data collection method, has been

discussed (*q.v.* section 4.3.5). Two different interviews were conducted: a retrospective think-aloud interview followed by the semi-structured interview. The results described in this section is a summary of participants' responses to the interview protocol (*q.v.* Appendix A 4).

5.2.5.1 Retrospective think aloud interview

The retrospective think-aloud interview as a data collection method was discussed in section 4.3.5. The essence of this interview was to understand participants' mental representations in small sets of program comprehension codes. The interview was based on sets of questions (Appendix A 5) obtained from course material (QBASIC) and mapped to the Block model. The participants were given a variety of problems but the interview focused on variables in a code, operation of a block, operation of the statement and the goal of a program. The questions were further mapped to the Block model, as well as Bloom's and SOLO taxonomies. An overview of the classification is presented in Table 5.8.

Table 5.8: Classification of retrospective questions

Question	Type	Revised Bloom's	Block model		SOLO
			Level	Comprehension dimension	
1	Tracing	Remember	Atoms	Text surface	Uni-structural
2	Tracing	Understanding	Block	Functions	Multi-structural
3	Explaining	Understanding	Atoms	Functions	Uni-structural
4	Explaining	Understand	Atoms	Program execution	Uni-structural
5	Code intent	Understand	Macrostructure	Text surface	Relational
6.	Writing	Creating	-	-	Multi-structural

The guidelines used by Thompson et al. (2008) and Whalley and Kasto (2013) were used to classify these questions for the revised Bloom's. For the SOLO taxonomy, the guidelines of Biggs and Collis (1982) and Clear et al. (2008b) were used. In the Blocks model, an *atom* was considered a single element within a code e.g. a variable. A *block* in the given code represents a loop, *relations* represents a reference between sub-goals or blocks, and an atom, while *macro structure* represents the goal of the program. An explanation of the mapping of the Block model to SOLO and revised Bloom's taxonomies is explained in section 5.3.6. The essence of this classification was to determine the level at which the participants operated during the think-aloud interview. A discussion of students' mental representation of each program comprehension question in relation to the model will now follow.

Question 1: Code Tracing

Question 1 required the students to identify the variables in the set of code. Tracing questions, as noted by Whalley and Kasto (2013), involve a line by line tracking of data. The question asks that students retrieve required knowledge from memory, and it is therefore classified as *remember.* The answer given by the students comprised one or two constructs and were therefore classified as *unistructural*. On the Block model, the question was classified as *atoms* at the *text surface* level because this construct helps them to understand *only* the rules and not the flow, or the purpose, of the program. All students listed "*Sum*", "*Area*", "*BioScore*", "*TotalScore*" as variables in the code, except for one that listed "*Counter = Counter + 1*" and "*Sum = Sum + 1*" as variables, even though she could define what a variable is.

Question 2: Code Tracing

This question was posed in three different forms but the central idea was for students to explain their representation of the code. The question was classified as *understanding* because it required the students to explain the meaning of the line of code. The question was classified at the Block model level to be *blocks* and in the comprehension dimension to *function* because the line of code requires that they link their representation to the whole program. In code 1 (Appendix A 5), the understanding of the two participants was IF…THEN, GOTO 15 will make the control move to line 15. In code 2 (Appendix A 5), the participant related the condition to the whole program by saying *"the counter will be counting the counter + 1 until it reaches 100"*. Another representation was *the "calculation of the SUM begins at IF Counter < 100, and if the Counter <= 100, the control goes to 100 but when it reached 100, the program ends"*.

Question 3: Explanation of a line of code

This question required the student to express their explaining skills through an understanding of the program code. It was classified to be at the *atom* level with the comprehension dimension at *functions* because it required that the students explain the QBASIC statement's goal in the context it was used. It is unistructural because they gave a description of one portion of the code. The participant's representation of this code was *"the INPUT asks for the company's name…the computer expects the user to INPUT the expected value"*.

Question 4: Explanation of a line of code

Question 4 possesses the qualities of question 3 but the comprehension level of the Block model is *program execution.* It was classified at this level because it elicited the students' mental representation on the execution of the statement. Participants who responded to codes 2 and 3 noted that the statement means "*there is no number to count", "calculating for the first data", "when we go for the other one, the counter will now be equal to counter + 2.* For code 1, they noted that Count = Count + 1 is *"an addition of a value to the counter".*

Question 5: Code intent

This question required the students to explain and give representations of the purpose of the code. Since they were to explain the overall goal of the program, the classification of the Block model falls at the *text surface* level while the comprehension falls at the *macrostructure*. The code intent question had been formally classified by Thompson et al. (2008) as *understand* at the revised Bloom's and *relational* at the SOLO classification (Clear et al., 2008b). The participants responded to this question with *"to calculate gains of 20 or perhaps 21 company salesmen*" for code 1. For code 2 they stated that "*program to compute numbers less than or equal to 100"* and *"to calculate three scores and giving us the exponent of it"* for code 3*.*

Question 6: Writing

This question required students to re-write the program codes using the FOR…NEXT statement. It was classified as *creating* at the revised Bloom's because it requires that the students produce the program in another way by using knowledge gained in the programming classroom to construct their own program codes. It was difficult to classify the code at the Block model because the model was only meant for reading and not for program writing (Schulte, 2008). It was classified in SOLO as multi-structural as it required students' construct understanding of each line of code before writing it. Program codes written by participants differed based on their understanding of the FOR…NEXT statement. A sample of the written program will be explained in section 5.3.2.

Question 7: What they were thinking when solving the problem

When asked what they were thinking when solving the problem, they responded that ideas like: "*How am I going to solve it?*" "*Will I get the answer?*" "*Why is this computer telling me unexpected?*" were going through their minds. Some said that they did not understand the problem while others indicated that they used ideas gained in class to solve the problem, even though they did not write a correct program.

5.2.5.2 Semi-structured interview

The semi-structured interview, as a data collection method, was discussed in section 4.3.5. The interview protocol (Appendix A 4) was used to elicit responses from the participants. A summary of students' responses is forthwith discussed.

How did the teaching intervention help students to learn?

When asked *how the intervention helped students to learn,* the responses included comments that the intervention assisted them to remember the topics learned. For example, the drama and singing aspect of the learning enabled them to remember concepts in the exam. They noted that the intervention encouraged them to mix with their friends whilst showing them the way they *cannot* learn in the music for presentation. It encouraged them to learn cooperatively whilst allowing less used quadrant to be developed by learning through different means, planning of time and seeking help from peers.

Perception of Scratch programming?

When asked what their *perception of Scratch programming* was, they said Scratch was: 'okay', a stepping stone and strong foundation for programmers, a better way of learning programming, an aid to program writing, helpful in understanding QBASIC, a motivation for 60% of learners as well as a programming knowledge and understanding enabler. Scratch also helped to solve programming problems and aided them in learning how to program. The use of Scratch made the class enjoyable because it practically *showed* what had been *learnt* in procedural programming. One participant emphatically stated that Scratch supplied a solid foundation and that any student who wanted to be a programmer should start with Scratch and *not* procedural programming as the first mentioned would expose one to how programs are written and which steps to follow.

Enjoyment of Scratch?

When the participants were asked whether they *enjoyed using Scratch,* they noted that they enjoyed the: typing aspect, designing of costumes, repeat...until, forever, sensing, sprite animations, making of variables and repetition. They also said that working with animal animations and making them do specific actions was fun. They noted that *without* Scratch, QBASIC would not have been effective. One of the participants said that when the class started with Scratch it was strange to him. However, he developed an interest in it because it was obligatory. When the class started using QBASIC he gained a new appreciation for what Scratch had done. One of the participants said that she enjoyed it a bit, but that she

was always happy when asked to work on Scratch because she would have the opportunity to work on the computer.

Usefulness of Scratch in relation to QBASIC?

When the students were asked to rate the *usefulness of Scratch in relation to QBASIC*, they said that Scratch had helped them to transit to QBASIC. Scratch makes programming with QBASIC easy and they developed the ability to arrange program codes before they moved to QBASIC. They also noted that Scratch helped in the learning of variables, operators and repetitions because it was also found in QBASIC. When problems had to be solved in QBASIC, knowledge of how this was done in Scratch was used. This background knowledge helped to solve problems like REPEAT...UNTIL, FOREVER...IF.

Relationship between Scratch and QBASIC?

Participants noted the relationship between Scratch and QBASIC identifying the variable and data definitions common to both programming environments. They added that Scratch was much like QBASIC, especially the programming aspect. They found that in Scratch some of the scripts had already been written while in QBASIC the user would have to write his/her own. One participant pointed out that if one did not know how to arrange a program in Scratch, it would be difficult to arrange codes in QBASIC.

Most frustrating thing about Scratch?

The participants explained their perceptions of Scratch programming and how it had helped them to better understand QBASIC. They pointed out the frustrations as well. Some of their frustrations included: the changing background, looking for costumes, the location of program blocks, variable declaration, searching for concepts and looking for dragon fire and inhuman things. One participant said that Scratch is complex because although his program was very long, his output was small. Another participant noted that the disappearance of the program scripts posed a challenge because when the power failure occurred, he had to start from the beginning because he was unable to ‘save’.

Friends’ feeling about Scratch?

When asked what their *friends’ feelings about Scratch* were, they noted that some of the students liked Scratch and that they were very good at running programs and working on expressions. Scratch motivated them to always come to class because it was fun and creative. A negative response was that some of them did not like Scratch because they felt the teacher should just jump to QBASIC and forget Scratch since it was not the major course that was to be learnt.

Participants' feelings about programming?

When asked what their *feelings about programming* were at the end of the semester, they said that fellow students, friends and seniors who had done programming before told them that it was hard. They noted that had it not been for Scratch, they would have changed their course. One participant said that he really liked programming and that he had developed a passion for it because of the way Scratch had broken down the programming process. He added that he wanted to become a programmer.

Describe your overall experience in the procedural programming classroom

When asked to describe what their *experience was in the procedural programming classroom*, they said that they had gained experience in the running of programs, thinking far and how to save work done on the computer. They could write programs and do designs using Scratch programming. They had gained knowledge of flowcharts and pseudocode, learnt the use of INPUT statement, operators, keywords and QBASIC meaning. Programming also made them realise that problematic situations in real life must be tackled in a step-by-step fashion because, in programming, problems are given without a clue to the answers. They commented that programming related to many aspects of life and that it fostered the creation of a plan and determination. Participants also noted that the procedural programming classroom was always interactive, exciting, filled with questions and the class they enjoyed most during the semester. One of the participants said that she liked simple calculations on flowcharts and pseudocode, but did not understand Scratch programming well.

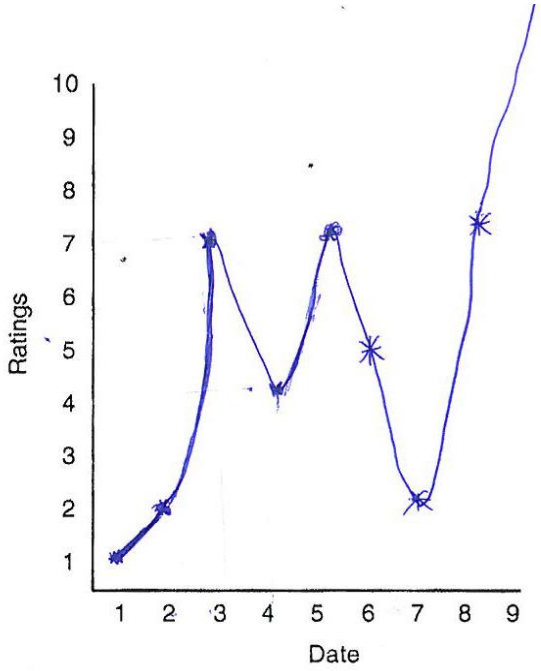
Recommendation towards the design of a new learning framework

When the participants were asked as to their *recommendations towards the design of a new teaching and learning framework* for programming in Nigeria, they said that Scratch should be included in the syllabus as a foundation course before the teaching and learning of programming. Therefore, lecture time should be *increased* as the course time was very limited and, as such, they were unable to learn all they needed. The teaching strategies which they wanted the teacher to use in the classroom included: allowing students to do individual research, teaching by differentiation, presentation of projects in the class, deep explanation of concepts, group learning and allowing for social interaction amongst groups. They also noted the provision of constant electricity and functional computers.

5.2.6 Reflective learning journals

The participants' summative reflections on what they had learnt in the programming course during the semester were detailed in their journal (*q.v.* Appendix A 22). They noted that they enjoyed the first class on flowchart and algorithm very much, while one participant noted that he understood a bit. They wrote that the topic on repetition was interactive, but some indicated a low self-efficacy for repetition. They understood Scratch programming well and commented that it aided their understanding of QBASIC. However, some students said that they had a limited understanding of QBASIC because it was complex programming while other participants said that it was easy for them because they had regularly practiced on their own. Students also noted that they had enjoyed the presentation of the projects which they had done in class. As evidence of this account is a sample (see Figure 5.26) of one of the participant's progression in programming. He rated himself according to his weekly performances with explanations.

Scale 1



Explain?

Date

① My first time in class, I enjoyed the class but I couldnt add up what programing had to do with drama presentation

② I was catching up and scratch was introduced as a concept of basic programing

③ Everything fits together. now I knew what the drama presentation was for. we had a test which I thought I did well on/till.

④ Test result came out and I didnt do so well so I felt bad through out the class

⑤ A presentation class; I did my best and I enjoyed the class so much

⑥ Basic programing not so very basic for me. More like

⑦ Complex programing

⑦ Another test: did poorly and it ruined my mood

⑧ did another test & did wonderfully - im proud of my scores.

Figure 5.26: Sample of reflection on learning and achievement in programming

5.3 RESEARCH FINDINGS – AR CYCLE 1

This section presents a *phenomenological reflection* on data with the exact structures of the students' lived experiences in programming. It involves both *phenomenological description* and *interpretation* of the participants' lived experiences in programming. The description of *experiences* cannot be captured abstractly because it is based on lived experience (Van Manen, 2016). Therefore, the section commences with the participants' background information, followed by the construction of themes using thematic analysis of data (*q.v.* section 4.4.2). Further discussion focuses on the description of empirical themes which emerged from the data.

5.3.1 Background information of participants

Percy is 17 years old and was born in Lagos state. He attended Lagos state model nursery and primary schools. He furthered his education at Ilekere junior high school and Ifesowapo comprehensive college where he attained his junior and senior school certificates. He was motivated to study computer science because of his previous exposure to gaming and different software packages on the computer system when he was young. Although he had intended to study chemistry, he chose computer science (his second choice) as chemistry was not offered as a subject at the college. He had wanted to study at university but his parents could not afford his tuition so he was left with no choice but to enrol at the college. He has no programming background.

Ramsey is 22 years old and studied at St. Michael Anglican primary school and Pobuna secondary school Epe where he finished his primary and secondary education, respectively. He joined the college because he had not been enrolled for his desired course at the university. He was motivated to study computer science because he is interested in the domain and he wanted the opportunity to work on a computer. He has no programming experience.

Bassey is 18 years old. He attended St. Mary primary school, Ojo, Ijanikin, Lagos state. He then attended Ilogbo Elegba and Araromi Ilogbo secondary schools where he completed his junior and senior secondary education, respectively. He wanted to study mechanical engineering but while awaiting his admission into the tertiary institution, his parents advised him to enrol for a computer engineering apprenticeship programme at Computer Village, Lagos, Nigeria. This motivated him to choose computer science as his course of discipline in the college. He has limited knowledge regarding computer repairs and no programming knowledge background.

Vanessa is 19 years old and attended Winners nursery and primary schools, Oriokuta, Ikorodu, Lagos, Nigeria. She then proceeded to True Vine college and Kingsway comprehensive college, both in Ikorodu, Lagos. She sees joining the college as a stepping stone after which she intends to secure a degree as soon as she completes her three year programme at the college. She was exposed to programming in senior secondary level but she was not taught how to write the program. Even though she was taught programming she had no understanding of the concepts taught.

Nancy is 18 years old. She attended Iwe-Nla nursery and primary schools. Because her parents could not afford to pay her school fees at the private school, they moved her to a public school, Lupetoro primary school, where she completed her primary school education. She then furthered her education at Ogunmodede secondary high school, Epe, Lagos, Nigeria. She has no interest in studying computer science, but because her course of choice is not offered at the college, she was left with no choice. Although her decision to study computer science at the college was guided by her previous exposure to computer science education at secondary school level. She has no programming experience. Nancy later left the college after the first semester exam to study biochemistry, her career choice, at university.

Mercy is 19 years old. She attended Excel nursery and primary schools, Iragushin, Epe, Lagos where she obtained her primary school certificate. She later attended St. Patrick grammar school from 2008 to 2013. She became motivated to study computer science because she wanted to know more about the course. She claimed that she had never operated a computer before and had no programming background.

5.3.2 Theme construction

Presenting an extensive phenomenological description involves determining the essential themes on which the phenomenological description will be based (Van Manen, 2016). The empirical data evidence was generated from observation transcripts, interview transcripts, feedback, reflective learning journals and the researcher's journal. The interim analysis of classroom observation (*q.v*. section 5.2.3) has been discussed. The processes of constructing themes involved the thematic analysis of data (Braun and Clarke, 2006; Creswell, 2014), as discussed in Chapter 4 (*q.v.* section 4.4.2). Using this method, I transcribed the recorded audio interview data and video observations of classroom processes. All textual data were reread on two different occasions to facilitate full immersion. During the final data analysis of the observations, I re-watched the videos on

two different occasions to attain a complete grasp of the content before corrections were made. I did the transcribing myself, but as a novice in data coding, I found the coding quite difficult. I sought help from my colleagues and other literature sources like Braun and Clarke (2006) and Saldaña (2015) which I read on many occasions to ameliorate my understanding. The relevant aspects of the videos, as they related to my research questions, were transcribed and coded, though not perfectly the first time. I recorded the transcripts a second time, based on each lesson. The coded interviews were returned to the participants for member checking and corrections were then made. Approval was given for the transcribed data. The participants' feedback and their learning journals had already been written. From each transcript, important sections were highlighted with different coloured markers to identify related ideas. I initiated the coding process by using a first cycle descriptive and *'in vivo'* coding (Saldaña, 2015) with 337 initial codes. During the second cycle coding, patterned codes were clustered together to form 39 categories (Miles, Huberman, & Saldana, 2014). These categories were reduced to 25 themes (Braun and Clarke, 2006). In determining the essential themes, some themes were discarded whilst others were compressed within another theme based on *latent content*. A critical investigation into these codes revealed that I still did not understand coding. I read more textbooks and sought assistance from other researchers and my supervisor. The transcripts were then reread and re-coded. Using a template (Appendix C 1) jointly designed by my supervisor and myself, a neat and comprehensive result was obtained. The template included: themes, subthemes, categories, acronyms, sources of data and number of occurrences. I used the 'navigation function' on the Microsoft Word 2018 software package to easily locate and determine the number of occurrences of each code. In the second phase of the coding I used one definite name to code data that depicts the same meaning, unlike the first phase where I coded every data with different names. As a result, 96 codes were generated. The *word cloud* presented in Figure 5.27 is a visual representation of the prominence of each worded code.

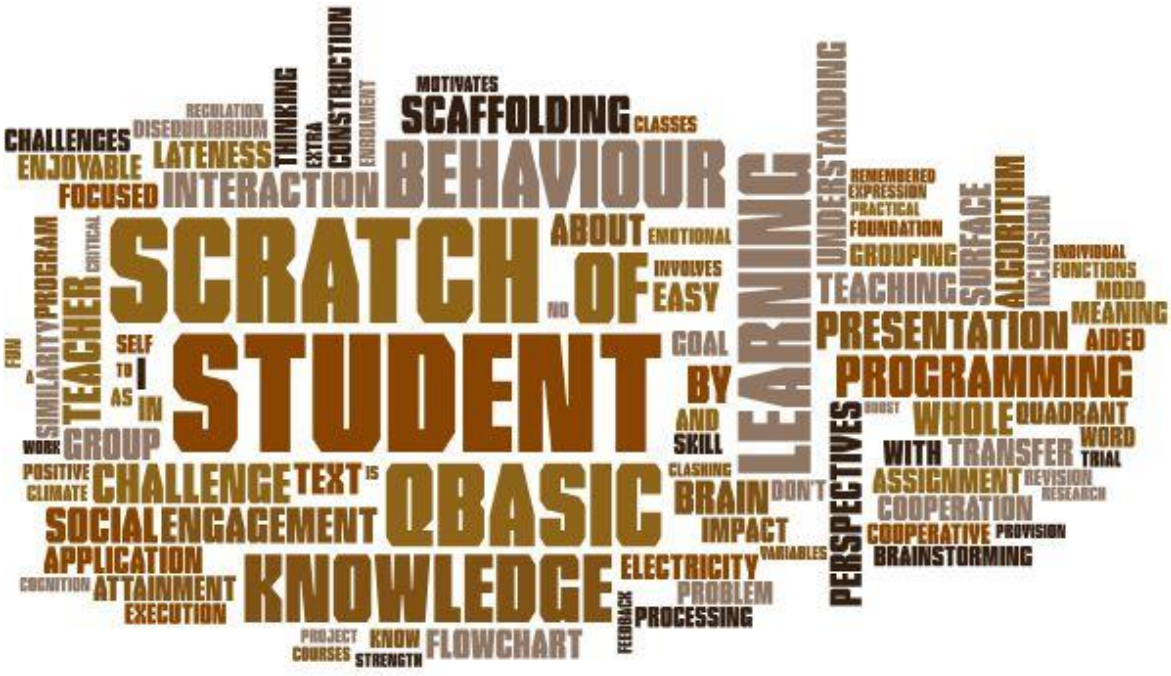


Figure 5.27: Word cloud representing the codes

The related codes were patterned into 24 *categories*. The categories were further reduced to *8 subthemes* and *3 major themes* (Attride-Stirling, 2001). These themes are the *phenomenological themes* which truly describe the structures of the participants' lived experiences in programming (Van Manen, 2016). Table 5.7 presents a summary of the analysis. A comprehensive analysis is presented in Appendix C 1.

Table 5.9: Summary of themes, subthemes and categories

Themes	Subthemes	Categories
Theme 1 Constructivist strategies for teaching and learning programming.	**Subtheme 1.1** Teaching strategies.	Whole brain teaching strategies. Constructivist teaching strategies. Grouping benefits and negative impact on learning. Teacher's personality.
	Subtheme 1.2 Strategies students used for the learning of programming.	Cognitive learning strategy. Self-regulation of learning. Social interaction during learning. Problem-solving.
Theme 2 Programming knowledge gained by students and impact of contextual factors on learning.	**Subtheme 2.1** Programming knowledge gained by students.	Perspectives about Scratch and QBASIC programming. Knowledge gained in Scratch and QBASIC programming. Word processing skills.
	Subtheme 2.2 Impact of programming knowledge on students.	Programming aids critical thinking Programming impact on students Scratch builds program writing skills Scratch inclusion to curriculum
	Subtheme 2.3 Impact of student factors on the programming knowledge gained.	Students' strengths in programming and challenges faced. Student behaviour. Student well-being.
	Subtheme 2.4 Contextual problem as impediment to the teaching and learning of programming.	Contextual problem. Suggested solution to contextual problem.
Theme 3 Mental representation during programming.	**Subtheme 3.1** Mental representation of the block model.	Atom Blocks Macro-structure.
	Subtheme 3.2 Affective and behavioural states during programming.	Affective states.
		Behavioural states.

5.3.2 Description of Themes

This section presents a phenomenological description of the main themes, subthemes and categories. These themes are used to describe participants' lived experiences in the programming classroom. The empirical evidence is based on *interview transcripts*, *observation transcripts*, *structured observation*, *written reflective learning journals* and *feedback from six participants* and *one non-participant observer*. The themes are supported with verbatim quotes from the participants. I acknowledge that the quotes are the participants' own words and therefore contain some grammatical errors which cannot be changed. The participants' pseudonyms, as used in the description, are Percy, Ramsey, Bassey, Vanessa, Nancy and Mercy. To facilitate easy understanding of the data evidence supporting the findings, the names replace the earlier symbolic figures of the participants: F3, C1, D1, A2, A3 and D2, respectively.

5.3.2.1 Theme 1: Constructivist strategies for teaching and learning programming

The discussion of constructivist strategies for teaching and learning programming is all-encompassing. The thematic map (see Fig. 5.28) illustrates Theme 1. Theme 1 comprises two subthemes: teaching strategies and strategies students used for learning programming. Subtheme 1 shows that the teaching strategies used in the programming classroom include: whole brain teaching and constructivist strategies, grouping benefits and negative impacts as well as teacher's personality. The second subtheme shows that students learned programming using a constructivist learning strategy, self-regulation of learning, social interaction and problem-solving. The following discussion focusses on the description of Theme 1, based on the subthemes and categories.

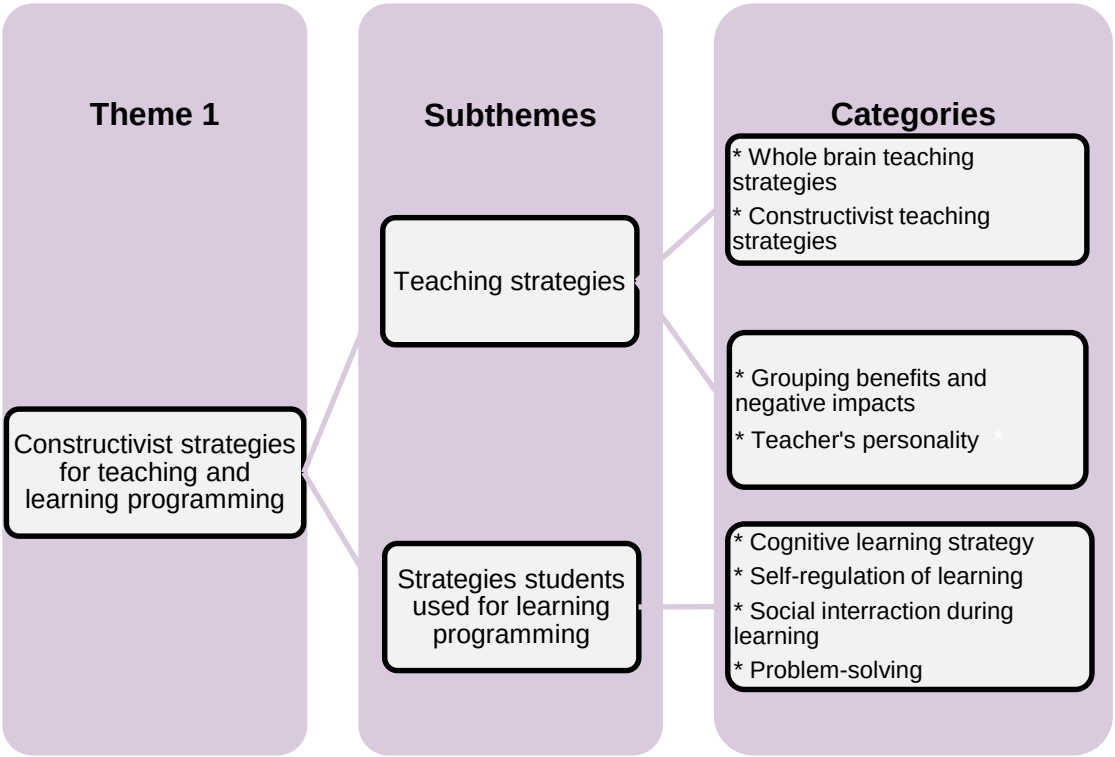


Figure 5.28: Thematic network of Theme 1

Subtheme 1.1: Teaching strategies

Teaching strategies relate to the different teaching methods which are supported by underlying learning theories used by the teacher to facilitate the teaching of programming. The strategies are: whole brain teaching and constructivist teaching strategies. Other findings focus on grouping benefits and negative impacts, as well as how the teacher's personality impacted on the teaching of programming. Each category will now be discussed.

Category 1.1.1 – Whole brain teaching strategies

The participants discussed the influence of whole brain teaching (WBT) strategies used in the classroom on the learning of programming. These include "*grouping the students to develop their own learning*" (Mercy, whole brain feedback) and the use of "*different ways of learning to make us understand helped us very well in her course and others*" (Ramsey, whole brain feedback). Teaching, through the act of dramatising, encourages schema activation, *"the exam I can remember, oh so, so person acted this…like it will be ringing in your mind"* (Nancy, interview excerpt). Other whole brain strategies used included poem "*from the poem presented, students were able to list the steps to solving an algorithm",* role play "*from the role play students were able to construct meaning on variables"*, kinaesthetic activities, "*allow them to stand up and move around to remove fatigue due to prolonged sitting*", lesson summary "*the lesson was summarised and students' understanding were established"* (classroom observation, Lessons 3, 4, 5 and 10 respectively, 2016).

The use of class presentation, "*presentation of group projects using the whole brain method was done*" (classroom observation, 23 March 2016) ensured that students were actively involved in the learning of programming. One participant, revealing his experience said, *"it has been somehow interesting. It has made the class lively"* (Ramsey, interview excerpt). Supporting the findings, another participant quipped, "*a presentation class, I did my best and I enjoyed the class so much"* (Percy, reflective journal). Apart from the presentation making the class interesting and lively, one participant also explained its benefit to the understanding of programming concepts. He said, *"it will make you to always remember…like that QBASIC some people brought cardboard out…so even in the exam if you don't remember it inside the book, once you flashback that we did the presentation… you know that you will remember*" (Bassey, interview excerpt). The whole brain quadrant grouping aided the learning of programming. Through quadrant grouping, "*I was able to mix with my friend, mix with my friends, even the way I can't learn I was able to learn in that way…but me mixing with them has helped me to be able to sing and write some notes of music and that has boosted my thinking preference"* (Vanessa, interview excerpt). Whole brain teaching strategies made the participants "*know more about yourself*" (Percy, interview excerpt). Vanessa, talking about her friends, said the whole brain teaching, "*finds out and searches about them" and "they feel it has helped them to boost their way of thinking*". This was why participants' responses from the feedback questionnaire revealed that half of the class supported that the teacher almost always (50%), frequently (25%) and occasionally (13%) helped them to learn using other means which are not solely based on their comfort zones (*q.v.* section 5.2.2).

Category 1.1.2 – Constructivist teaching strategies

This category discusses different constructivist teaching strategies used by the teacher to facilitate the teaching of programming during the semester. Strategies, such as group learning, were used at different times. One of the participants said teaching was done by "*grouping the students to develop their own learning"* (Mercy, whole brain feedback). Through scaffolding by teacher and student, "*students in each group explored the Scratch environment...some students in Group A, B, D and G could get it with little guidance. They further assisted each other in their groups without calling me for guidance. Group C, E and F had problems working with the Scratch environment and I visited them at intervals to put them through"* (classroom observation, 17 February 2016). Therefore, Mercy seeing the importance of scaffolding, stated that she needs to be guided on *"how to run programs"* for her to be confident in program writing and testing.

Other constructivist teaching strategies used to facilitate learning of programming included: the use of assignments, individual self-paced learning, "*they were asked to do no 7 individually based on the experience gathered in the group activity"* (classroom observation, 9 March 2016), revision of learning to further students' engagement, learn on their own, as well as determining students' level of understanding from their explanations regarding the last lesson. Classroom observations, supporting the findings, shows that, "a*t the beginning of the lesson, students were asked to summarise the last lesson taught. Student A2 talked about concrete algorithm but she couldn't explain abstract algorithm… talked on the steps in solving an algorithm and could list them from the poem sang in the last lesson in her group* (classroom observation, 10 February 2016). Using real-life application for teaching helped one of the participants to always "*link learning to real life situation"* (Percy, whole brain feedback). Likewise, findings from the feedback questionnaire showed that the teacher frequently (38%), almost always (25%) and occasionally (25%) linked programming problems to real life. Only 13% of the students hardly ever experienced the teacher doing this.

With the use of cooperative learning, one participant noted, "*like when you scattered us on, like that the presentation. So, everybody just mixed up together. We combined and we make the thing work. So, we put ideas together so that we can do the presentation".* Through extensive practical use in the classroom, we were "*able to understand the topic".* The teaching of programming to strengthen critical thinking developed the participants' critical thinking. One participant said, *"she teaches us in a way that involves critical thinking and helps us to understand the course"* (Vanessa, whole brain feedback). Another teaching strategy the teacher used, and upon which the participants lay much emphasis, was doing

research work. Research was a way to explore the world and life from their perspective. Through this, different experiences were brought into the classroom. The concepts were discussed towards the construction of knowledge. According to the participants, *"when we did research it was interesting, it was inviting"* (Percy, interview excerpt). Another participant added, *"like make research, go to the net, how to do this…It's okay that we should like encourage students. Like not just like they should be spoon-fed by teachers. Let them go out also and make research on their own. Let them see what they can get, bring it to the class, interact and discuss together*" (Nancy, interview excerpt). In summary, facilitation of programming through these strategies promoted student engagement where students were actively involved and committed to the learning of programming. A classroom observation supporting the findings shows that *"after the students have done numbers 1 to 6 in groups, I solved the questions following together with the students*" (classroom observations, 9 March 2016).

Category 1.1.3 – Grouping benefits and negative impacts

This category explains the teacher's classroom observations on the use of grouping in the classroom. Quadrant grouping was discussed in category 1.1.1 and observations focus on both the benefits and negative impacts on the learning of programming. Findings reveal that some of the groups cooperated, while others did not, whenever projects were assigned. In support of the findings, a classroom observation reveals that "*students shared tasks; two female members checked the internet for the information while the other two male members of the group produced the notes for the song of the guitar and practised with it. They waited in school to practice together*". However, in another group this was not the case as "*student (B1) refused to join them"* during the group assignment. As a result, when there is no cooperation, the learning aim will not be achieved. Another negative impact on learning was observed when "*I was explaining the content and work to do to the class, group E members were discussing and arguing. Student E2's voice dominated the group*" (Classroom observation, 17 February 2016). This could affect the assimilation of knowledge by students in other groups.

Category 1.1.4 – Teacher's personality

Teacher's personality relates to the qualities found in the teacher which the students embraced as well as other expected qualities they long to see in programming teachers. The teacher's use of different teaching strategies, combined with her great passion and interest for teaching, created an environment conducive to students' understanding programming. In support of the findings, one participant said, *"the lecturer inspires the students by always showing great interest or passion for the subject matter and learning*

tasks...tries as much as possible to create a conducive atmosphere for learning" (Percy, whole brain feedback). This was supported by feedback from both the non-participant observer as well as the participants' survey in which 50% almost always, and 25% frequently noted that the teacher facilitated learning by showing a strong interest in programming and learning tasks. The findings also revealed that not only teaching methods and learning environment encouraged the learning of programming, but also the teacher's qualities and attitude to the subject matter. Percy's statements in an interview excerpt support the findings. He said "*on Saturday you come to school and the lecturer is there waiting for you. It's something that you should be happy about and our lecturer has tried for us in CSC 112. I won't lie. You can't find so many lecturers like that in this institution".*

However, the teacher was also expected to facilitate the learning of introverts in the class by re-explaining programming concepts to those students who did not understand. The focus should thus not only be on students who *do* understand the topic, but the teacher should also identify non-interactive students and ascertain whether they understand the topic. The participants also expect a detailed explanation of concepts. An interview except from one of the participants supports this finding, *"Like the lecturers should interpret more questions for the students, and they should not like focus on people that know it better…the teacher should be able to ask from the students even from the people that they think that may not know it"* (Vanessa, interview excerpt). In addition, time management was an aspect of teaching the participants frowned upon. They expected the teaching and learning to end when the lesson was finished. Although they did not formally explain their views, during one classroom observation I noted *that "students can feel bored and not ready to learn when I try to use the extra time".* They were very sensitive and aware of the teacher using extra time.

Subtheme 1.2: Strategies students used for learning programming

This subtheme explains different methods used by the participants in accomplishing the learning of programming. These include constructivist learning strategy, social interaction, self-regulation of learning and problem-solving. The findings are discussed below in categories 1.2.1 to 1.2.4.

Category 1.2.1 – Cognitive learning strategy

This category describes cognitive strategies used by students in learning programming. Classroom observations enabled students to construct the meaning of programming concepts. The equilibrium was disrupted when new information was added to the schema. This became evident during classroom learning, through role play centred on variables and

poem formation algorithms. For example, "*students could list the steps to solving an algorithm from the poem*" (classroom observation, 3 February 2016). However, there were arguments regarding students' understanding of variables, *"when students were asked to identify variables, they were at disequilibrium when they saw 'DisplayCounterAverage' and 'Counter + 1', but after many explanations, they were able to accommodate"* the meaning of variables (classroom observation, 2 March 2016).

Category 1.2.2 – Self-regulation of learning

Self-regulation of learning relates to the processes the students used to plan, set goals and monitor their learning of programming. They self-regulated their learning of programming using different means including seeking for help from the teacher and their peers. For example, in group E, participants *" called on me for help"* and "*according to the students they had some challenges in presenting their group work. They couldn't play the saxophone and at the same time, they couldn't get one. Because the students need to give a touch of the instrument to their song and none of them could do it, they, therefore, invited a student from the other department, who helped them to play the saxophone using his mouth"* (observation transcript, week 8, 9 March 2016). Another way in which the students self-regulated their learning was through valuing of learning. This they exhibited by raising a hand to ask for help when they did not understand a concept, waiting at the end of the class to complete a class activity and even contributing money to complete an assignment. An observation excerpt supports the finding, "*some students could not finish the class activity during the lesson. They waited to do it after the class*" (classroom observation, 9 March 2016). Corroborating the findings from the classroom, one of the participants said, *"the lesson need thinking, so I have to go home and read and understand what I have learnt in school"* (Nancy, reflective learning journal). Students also learned programming by maintaining a positive emotional climate which they exhibited by "*clapping their hands when they got the answer*" and using a multidimensional view of learning, as one of the participants said "*I constructed a multidimensional view in my mind about the lesson taught*" (Percy, whole brain feedback) and regulation of cognition whereby "s*tudent E2 stood up from her seat and walk up to Bassey for clarification on the class activity*" (classroom observation, 9 March 2016). Students' survey responses, in section 5.2.2.1, further detail the ways in which students self-regulated their own learning.

Category 1.2.3 – Social interaction during learning

In this category, I discuss *how* programming was learnt through classroom interaction. Findings revealed that participants socially interacted within groups during programming activities. A classroom observation reveals that "*towards the end of the class another*

cooperative work was given to the students in their groups and they all interacted well without any argument" (classroom observation, 27 January 2016). In support of this finding, the participants said *"I learnt and understand that knowing the kind of...and relate with others help in learning greatly" (Bassey, whole brain feedback),* and *"I gain a lot about this five-repetition structure in flowchart because the class was interactive"* (Mercy, reflective learning journal). Supporting the findings, feedback survey responses show that the students frequently (75%), and occasionally (25%), developed an enquiring mind in the programming classroom through lecturer-student discussions. However, not all the participants fully participated in the classroom. One participant said *"I partially involved myself in the class activities"* (Ramsey, whole brain feedback). Findings from the collaborative group checklist show that *"Groups (A and E) participated 4 times, groups (B, D and G) participated three times while group C only participated once".* This confirms Ramsey's statement because he was in group C.

Category 1.2.4 – Problem-solving

This category describes the mental processes used by the students in solving a problem towards the learning of programming concepts. Such processes included brainstorming. This finding was supported by observations made by the non-participant observer. The participants problem solved during programming activities and especially when they were given class work on the multiplication table, *"group E students were also struggling with the problem then I asked the class to pause for some brainstorming so that other groups could be helped"* (classroom observation, 9 March 2016). In another classroom observation, the following discussions took place:

> *"Student A4: Me, I've not seen any….90 degrees*
> *Student A3: How many sides*
> *Student A4: You, its 3 steps in 2 so we now*
> *Student A2: (pointing to the computer) This is where the problem is*
> *Student A4: Maybe we should not do it like that. Maybe we should use 80 and 120*
> *Student A2: It is drawing a straight line for us*
> *Student A3: Let us use 180*
> *Student A4: It supposed to be 90 X 2…"* (classroom observation, 2 March 2016).

5.3.2.2 Theme 2 – Programming knowledge gained by students

This theme describes the series of programming knowledge students gained when exposed to programming. Included in this discussion is the impact of student and contextual factors on the teaching and learning of programming. The thematic map (see Figure 5.29) illustrates

Theme 2. Therefore, Theme 2 comprises four subthemes and eleven categories. The following discussion focuses on the description of Theme 2 based on subthemes and categories.

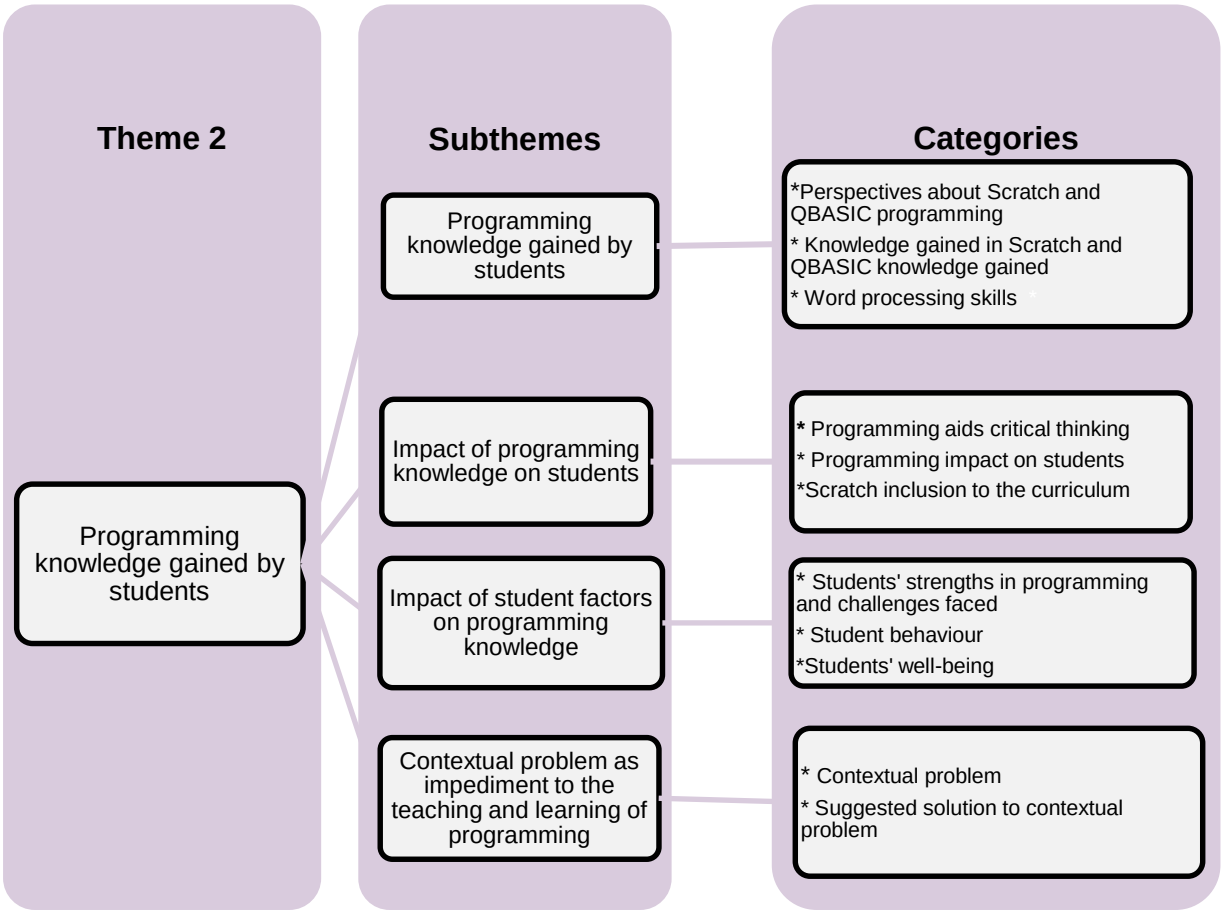


Figure 5.29: Thematic network of Theme 2

Subtheme 2.1 – Programming knowledge gained by students

In this subtheme, I discuss the different programming knowledge gained by the students in the programming classroom. The discussion focuses on knowledge gained and students' perspectives on both Scratch and QBASIC programming, knowledge of word processing skills and previous experience as it impacted on the programming knowledge gained. These findings are discussed in categories 2.1.1 to 2.1.3.

Category 2.1.1 – Perspectives about Scratch and QBASIC programming

This category describes the participants' own views about Scratch and QBASIC programming. They perceived Scratch programming language as fun, easy to learn and enjoyable. The aspects of Scratch which the participants found enjoyable include: *"the blocks, those diagrams, the fantasy, the transportation"* (Mercy, interview excerpt) and *"the*

balls, the diagrams of animals…placing animals and making them do some specific action" (Ramsey, interview excerpt). In support of the findings, Vanessa added that "*it was practical and easy to understand"* (reflective learning journal). Scratch also motivates *"almost, I can say 60% among all of us…to always come for lectures"* (Bassey, interview excerpt). All the participants saw similarities between Scratch and QBASIC programming. Similarities are found in the running of programs and the use of operators and variables. This finding is supported by one of the participants saying "*like a means to learn programming, just using the operators and the variables because as you have variables in Scratch we also have it in the programming*" (Vanessa, interview excerpt). Other participants relate the similarities to the interface of each of the programming languages. Scratch visually represents sprites which are picked from the blocks pallet, while QBASIC programming involves the typing and memorising of commands. Percy, while sharing his views said *"the relationship between Scratch and QBASIC is the programming aspect. In Scratch we have scripts and in QBASIC there are programs, and in Scratch, the scripts are already written for you. You just have to pick them and arrange them. But in QBASIC you have to write it yourself. So, if you do not know how to arrange programs in Scratch even to write a program in QBASIC will be difficult".* Because of these commonalities between them, Scratch could be applied in the learning of QBASIC programming concepts such as repetition structures, variables and operators. This finding is supported by the following interview except, *"Scratch is useful to QBASIC because of those operators and variables"* (Vanessa, interview excerpt) and *"there was something we did in Scratch that we were talking about instead of repeating we use just REPEAT…UNTIL…So, when we move to QBASIC instead of me to repeat I can just say let 5 equal to so, so and so. If not for the knowledge of Scratch, I won't be able to understand"* (Nancy, interview excerpt).

Therefore, QBASIC was easy with Scratch because "*it was the Scratch that makes us learn QBASIC faster"* (Nancy, interview excerpt) and "*the program writing was quite easy, but not very easy to get and understand, but the knowledge of the Scratch helped me to get it better"* (Vanessa, reflective learning journal). Since the participants experienced Scratch as beneficial to the building of programming knowledge and development of skills needed for comprehending QBASIC programming concepts, the participants concluded that Scratch is a foundation to the learning of programming. The finding shows that *"Scratch is just like a foundation for someone to…of having at least a little knowledge about what QBASIC programming is all about"* (Ramsey, interview excerpt) and *"I see Scratch as a stepping stone to QBASIC...Scratch helped me actually as an individual to understand QBASIC better…it will expose you to how programs are written and the steps you should follow when you are writing program"* (Percy, interview excerpt). Participants did not really feel this

impact until they *"move to QBASIC, that is when I can now say that, oh! thank God for the knowledge of Scratch"* (Nancy, interview excerpt).

The participants also discussed their perception of QBASIC programming and programming in general. Supporting this finding with the following interview excerpts, they noted that QBASIC *"is just for fresher's like us, for beginners, it will make us to know that level of programming before jumping into the real aspect"* (Bassey), but *"the QBASIC programming is not easy to understand"* (Mercy) and *"QBASIC programming not so very basic for me. More like complex programming"* (Percy, reflective learning journal). However, *"QBASIC programming relates to many aspects of life because life because in life, everything is just about segments After you have done something you look at what next. What haven't I done? What should I do? What am doing and what should I have done. QBASIC programming is an opener for any student that knows what he is doing"* (Percy).

While sharing general views on programming, the participants noted that they had a premonition that programming would be hard and this was fueled by prior information from friends, non-computer science majors and even their seniors (computer science major). Participants expressed the caution they received from their peers. They noted *"most of this our seniors have been telling us that QBASIC is hard…*" (Bassey) and *"so, I was like I want to change my course…even my course mate they were telling me somebody told me not to do this course, that people don't really get programming*" (Nancy). The students claimed their preconceived ideas about programming later changed due to the experience they had in Scratch programming, because "*when you said we are going to make use of the Scratch programming…then when we entered programming, I said what is even there that everybody is shouting about. It was okay, maybe I think because of the knowledge we had in Scratch*…" (Nancy, interview excerpt).

Category 2.1.2 – Knowledge gained in Scratch and QBASIC

This category describes the programming knowledge gained by students in Scratch and QBASIC programming. The participants noted that understanding gained in QBASIC programming included the use of statements and commands. Vanessa and Nancy respectively said, "*I don't even know about programming before. So, the knowledge I gained in class about QBASIC has made me know a lot of things on programming and how to write programs using different commands and statements"* and *"the QBASIC keywords".* However, during the sixth week of the semester, Ramsey noted that QBASIC *"is new to me and I did not get much out of it*" when it was introduced.

Explaining this category further, the participants described the general knowledge gained on variables, repetition, control structures, expressions and flowchart algorithm. Learning

algorithm and pseudocode was an interesting topic and Ramsey and Bassey claimed that they understood the concept well because *"I now understand that pseudocode deals with normal word and flowchart deal with using a diagram, as in using plan shapes to solve an algorithm"* (Bassey, interview excerpt). Others said it took them a long time before they understood the concept, especially in the first class. The first class here represents the different times the students joined the class. For example, *"difference between algorithms and a flowchart because it was the first class, so I couldn't understand better"* (Ramsey, reflective journal). The possibility that the students did not resume classes during the second and third class, where foundation topics were taught, may have attributed to this. Supporting the findings, evidence from the third lesson for the semester (the day the concept of algorithms was introduced which Ramsey termed the 'first day') reveals that *"from her solution, it was obvious that she has not mastered the differences between the symbols of flowcharts. She used input/output symbols throughout the solution. I revised flowchart symbols again, and after the explanation…more problems were given on algorithms, flowcharts and pseudocode. Students can now link action to each flowchart symbols after solving series of problems"* (classroom observation, 3 February 2016). However, the students' knowledge of flowchart algorithm improved and this was seen in the twelfth lesson where the *"majority of the students could solve an algorithmic representation of a Scratch program, except for four of the students who are still developing as their performance on the activity were below average"* (classroom observation, 2 April 2016).

On control structures, findings show that not all the students have a complete understanding. Supporting the findings, evidence from one participant shows *"I understood the control structure but I don't know how to write and solve some questions under it"* (Vanessa, reflective journal). With repetition, during the seventh lesson when the topic was introduced, Vanessa said *"I do not really understand the repetition structure but I was able to have an idea of it"*. On variables, findings show that though students were taught variables and its application in solving programming problems, they did not always declare variables for programming problems. Supporting the findings, evidence from three classroom observations respectively reveals that "*groups (C, D, E, F and G) also presented their results too without making use of variables in their solution…when I looked round at their work, I noticed the students did not all make use variables in their group work"* and *"in the problem given, some of them used the 'number of hours' without declaring the number of hours as a variable"* (classroom observations, 27 March, 2 and 4 April 2016). On expressions, the participants claimed that they understood expressions. Classroom observation showed that *"after subsequent teaching, the participants understanding of expression improved, while others still struggle*" though the participants did not explain this. I, therefore, noted in my

journal, *"I can deduce that group C and F students can still not construct understanding on expressions. This may impact negatively on the way these students apply expressions in the running of programs"* (researcher's journal, 2 March 2016). However, during the eleventh week of the semester, a further classroom observation showed that students performed better than they had done in previous exercises. Supporting the findings, "*comparing previous classwork (1/4/20i6) on expression and today's activity two on expression, the performance of students A2, A4, B3 and D1; and students D2, E3, F3 who performed below average in the previous class has improved. However, Student A3 and B1 score dropped while students C1, C2, C4, E4, D3, F1, and F4 still performed low in this activity as well*" (classroom observation, 1 April 2016). One of the participant's statements further supported the findings by explaining how they understood solving problems with expression. Bassey said *"that BEDMAS is relevant to BODMAS. So, I quickly understand it that the O we remove it replace it with exponential. I will follow the procedure. And the QBASIC, you just know how to convert it when they give you the words in the normal mathematical expression you just know how to take it. Like if you are given 4 x* X^2*, you know that it will be 4*X^2*".

On Scratch programming, the knowledge gained in the Scratch programming classroom helped the participants in writing small sets of codes. They cognitively attributed the skill of writing a correct program to the arrangement of blocks and understanding of program writing on repetition structure in QBASIC programming. For example, *"like when you are supposed to put something in a LOOP… and you don't put it in the looping statement, the output will be very wrong. So, in QBASIC also if you are to face a problem and you do not arrange it in step by step. You do not arrange it in order then you will not get what you want to get"* (Percy, interview excerpt). Ramsey added that *"repetition in Scratch, it helped us to know that if we are to do something, there is a way you can do it instead of repeating it 100 times"* (Ramsey, interview excerpt).

Category 2.1.3 – Word processing skills

Apart from programming knowledge gained, students also gained word processing skills, such as typing. They related this knowledge to the constant use of the computer system in the programming classroom. A participant who cannot type and has limited knowledge of computer engineering said "*I don't know how to type, but since I've been going for that Scratch programming I was now using the opportunity to learn the way to type"* (Bassey, interview excerpt). Another participant, who was exposed to word processing, gained additional knowledge while learning. He said, *"it has made me develop another aspect of operating of system apart from just the normal way of way of just typing to another level"*

(Percy, interview excerpt). However, while some students claimed they could move to another level of word processing because of the frequent usage of the computer system, some of the students struggled with it. Although word processing skills are not deemed to be that important to programming, when students struggle to use some keyboard keys, it leads to frustration when programming. This finding is supported by an interview excerpt from Mercy "*I want to press delete and also to…assuming am writing like this…to come to the next line to…I did not get the…."* Corroborating the findings, it was observed that *"some students are not familiar with the computer keyboards, they can't locate some keys which also affect how they write their program...."*

Subtheme 2.2 – Impact of programming knowledge on students

This subtheme describes the effect which programming has on students including the fact that programming aids critical thinking, Scratch builds programming writing skills and creates interest in programming. These findings are discussed in the following sections.

Category 2.2.1 – Programming aids critical thinking

Participants' experiences in the programming classroom awakened them to the fact that programming involves thinking and it aided their critical thinking as well. Without critical thinking, problem solving in programming might be difficult. Therefore, students must *"reason…think a lot before you can come up with a solution"* (Percy, interview excerpt), "*QBASIC involves thinking. Because if you can't think then that means you can't solve anything under QBASIC… then it needs an application, going back to it repeatedly for you to understand and get what you are doing"* (Nancy, interview excerpt) and *"don't… jump to conclusion, just mere looking at what is on ground…must take a critical look at the thing and think deeply"* (Ramsey, interview excerpt).

Category 2.2.2 – Programming impact on students

The learning of programming did not just stop in the class, but was also applied to their daily lives. They attributed the programming knowledge gained as helping them in their personal planning. Percy noted *"[I] plan myself very well and when we got into Scratch and QBASIC, I realize that it is the same everywhere. Before you do anything, you should be able to think about the outcome. You should be able to picture what the outcome will be as you are starting immediately…."* Another participant said *"it makes me schedule my time, because schedule my time, program my time very well just like in QBASIC programming"* (Ramsey). These findings are also supported by excerpts from two participants. They noted *"I can now apply what I have learnt to anything at any time"* (Nancy, whole brain feedback) and

"whatever is being taught and I also apply it to other lessons and courses" (Vanessa, reflective journal). The impact was also felt in relation to dealing with life situations through problem-solving because *"the QBASIC programming classroom…has made me realise something that in this world when you are faced with situations you have to tackle it step by step. Because in QBASIC programming you will be given a problem without giving any clues on how you are to solve it".* The support afforded by Scratch awakened one participants' further interest in programming. This was stressed in the following interview excerpt *"well, I like programming with the way Scratch has helped break down programming for me. I feel like I want to be a programmer because it is the very interesting…and, I like programming a lot. I develop a passion for it"* (Percy, interview excerpt).

Category 2.4.3 – Scratch inclusion to the curriculum

The participants believed that due to the benefits received from Scratch, the teaching of Scratch programming should be included in the computer science curriculum. They stated that *"Scratch has helped me as a person to understand QBASIC…then I will advise the school to add Scratch to the syllabus"* (Percy, interview excerpt). Another participant gave reasons why he felt it should be added to the curriculum. He said "*like that Scratch you hardly see someone that will not, even though it is five years ago now and they now tell you to come, input cat as costume one no matter how the person will still remember since all those blocks are there*" (Bassey, interview excerpt). Also, Scratch should be required for the teaching of any programming course from the first year to the final year. Therefore, "*I want you to continue running that programming language…Scratch should be continually used as a foundation for other courses in any….just like using Scratch in the foundation of QBASIC programming. The using of Scratch also for other programming courses*" (Ramsey, interview excerpt). Vanessa also added *"we can continue doing the Scratch in other programming courses*". She stressed that other visual programming languages can support the learning of text-based programming languages and not necessarily Scratch programming, *"like if we are going to do any programming in either second semester or 200 level like we should, the school should like bring like an example, it might not even be Scratch"*.

Subtheme 2.3 – Impact of student factors on the programming knowledge

In this subtheme, different factors which impacted on the programming knowledge students gained are discussed. These factors include: strengths and challenges students faced in programming, student behaviour and student health. These findings are discussed in the categories below.

Category 2.3.1 – Students' strengths in programming and challenges faced

This category describes the strengths and challenges students experienced in programming which either *enabled* their understanding or *hindered* their progress in programming. Students claimed that the knowledge gained in Scratch aided their understanding of programming. Vanessa said *"when we are talking about the sprite, talking about the variables, the control and everything on how to write, and using the operators. These things also helped me to know more, is just the practical way of learning programming and the Scratch has also helped me to know more"*. In support of Vanessa's statements, during a classroom teaching on repetition and decision in Scratch programming, it was observed that *"in group B, they presented their assignment with probing questions from the teacher and students to ascertain whether they understood what they have done; and which they were able to defend"* (classroom observation, 2 March 2016).

An interesting finding highlighted by all the participants focused on the challenges encountered during QBASIC and Scratch programming. Challenges included students not being able to solve some programming problems in the classroom which they were supposed to have mastered in the ninth week into the semester. For example, *"they explained looping, string constants, numeric constants very well with examples. Students cannot boldly give examples of a variable but can define it and state its functions. Even the whole class could not give examples of a string variable"* (classroom observation, 23 March 2016). Another challenge relates to students' inability to locate costumes for assignment, *"they explained that they couldn't find the cat and so they could not complete the assignment"* (classroom observation, 23 March 2016). Another participant's challenge was the disappearance of assignments from the system, *"they did the assignment on the system, but they couldn't find the assignment as at the time I want to assess them"* (classroom observation, 2 March 2016). The reason for this was unclear, but there was a possibility that the students did not save the assignment correctly. It has already been established that students lack word processing skills which affected the way they program. Some factors, however, contributed to the students' challenges in programming. Supporting the findings, Mercy recorded in her reflective log during the first and fifth week of the semester respectively, *"because am still a stranger to the computer and am not good in the calculation, the…my system was not okay and I can't meet up, so I didn't understand"*. To corroborate the findings further, evidence from another participant during a retrospective interview was "*I don't really know how to combine eeeh, like IF, like INPUT, they put double quotation mark, enter the name and you put that quotation marks and, and you put that dot and comma and you put N and dollar sign. So that kind of thing now if I want to use it in that other"* (Bassey, interview excerpt). A final view was given by Mercy, as she shared her

perceptions as to the challenges her friends faced in programming, *"…on QBASIC, the QBASIC aspect, the data types, the mind mapping, I think they don't know it at all…"* (Mercy, interview excerpt).

Challenges faced in Scratch programming, which include the location of program blocks and costumes as well as searching for concepts to be use when writing a program, were discussed. This was described as *"the only place I was frustrated is just for me to know all those program blocks...and to know how to use them…the ones that the data is not given"* (Bassey) and *"the changing the background of sprite…then how to locate some variables, maybe like some barriers …those edge…how to find maybe a baby or a baby doll. They are so frustrating. It took me a lot of time to get that"* (Mercy). As well as, *"looking for sprites, looking for the girl working, all those kind of irritating thing…something that I will be looking for like the dragon with fire… Something that I will be looking for that is not human…going to the palette looking for NEXT, WAIT…UNTIL all those kinds of thing. It's frustrating but you learnt out of it"* (Nancy). The complex structure of the Scratch environment required the writing of longer scripts which produced a small result, *"you have to complete the whole program before you can get your result. And the result was just a simple result…when I was writing my program, it was very, very long"* (Percy). Other participants struggled with code vanishing, *"when I was writing the program on Scratch and all my programs left, without me pressing any command…then I wasn't able to get back the program"* (Vanessa). However, one participant discovered the reason behind such experience He said, *"eeeh, yes, the blocks of scripts, the block is...before I get to know about the blocks, I placed the wrong blocks, I carry the wrong blocks to the wrong area, so that…Scratch programming is just about just getting aright code to use. So, for me being, I mean using the wrong blocks at the wrong place is just"* (Ramsey). Two participants referred to the challenges faced with extra teaching hours. Percy initially said "They said…*in CSC 112, to be frank, it is the only course that, at least, I am satisfied. Because we came for Saturday classes and many lecturers won't want to come for Saturday classes*". His statement was supported by Mercy who said, "*maybe by creating a time different from the lecture time in order to educate more to the students*" while considering the limited time used for the teaching of Scratch and QBASIC programming. Percy finally stressed that although he liked extra classes he did not "*like coming to school on Saturdays"* (Percy, interview excerpt). One of the participants said, referring to his friends who are also computer science majors and Scratch programming, *"most of them don't like it. Because they feel that we should just skip into QBASIC and forget about Scratch. You know when most of them learnt that QBASIC is our real course, why then are we doing Scratch. But it has helped me, I don't know if it has helped them. It has helped me"* (Percy).

Category 2.3.2 - Student behaviour

Student behaviour describes desired behaviours exhibited by the participants which encouraged the learning of programming during the classroom sessions. For example, this desired behaviour was found during Lesson 5, when the participants were exposed to costume design in Scratch programming, *"at 12:50pm, Ramsey mumbled and banged the table showing he was totally lost and couldn't solve the problem of costume design. At 12:57pm he could still not solve the problem, but he beckoned to the teacher for assistance. At 1:04, Bassey, scaffold him, and he* (Ramsey) *reciprocated by thanking him with a shake of hands"* (classroom observation, 9 March 2016). In support of this, he raised his hand four times to ask assistance (see Table 5.5*).* The undesired behaviour exhibited by the students included truancy, "*student B1 felt confused, he cannot solve the problem…he doesn't come to class",* lack of seriousness regarding class activities*, "in group D, the members of the group were playing and not being serious about the class work",* lack of seriousness regarding assignments *"one of the students (E1) did not participate in the assignment and group discussions. She only contributed money. She complained that the time fixed was not favourable*" and grumbling towards peers, *"immediately student (F2) was called upon to represent group A to C, the students grumbled. Student A4 said, "better do it correctly".* This could be because they did not have confidence in her. Also, Nancy demonstrated undesirable behaviour during the fourth lesson on algorithms for calculation, *"at 12:38, she grumbled when I said the class will do a classwork. At 12:29, she placed her head on the table and acting restless, but she was participating in the class while resting her head on the table. At 12:52pm, a classwork was given and she said I don't understand, then I guided her. At 1:09, she placed her head on the table again. At 1:11 she said, 'you better do it well o' to a student that want to solve a problem on the whiteboard. At 1:35, she grumbled again when they were given classwork. She was acting as if she just wants to leave the class"* (classroom observations, 3 and 10 February, 23 March 2016).

Other behaviours exhibited by participants, which impacted on their learning of programming included lateness, which resulted in some programming concepts having been taught before other students arrived. For example, *"some students came late to the class and joined the class*" (classroom observation, 3 February 2016). Other causes of lateness included extended lecture time by other lecturers. In one of the programming classes, *"students were fifty-six minutes late to the class…they reported that a lecturer which they had his class earlier delayed them. And because it was a test the students couldn't leave his class".* These instances of lateness were not the fault of the students but displayed a lack respect, by other lecturers, for the stipulated lecture times. The students wrote an apology letter explaining when and why they had arrived late. However, *"lateness*

to the classes by the students was discouraged" as observed by the non-participant observer. Negligence to complete assignments can be attributed to students not having the required learning material, for example, *"I have asked the students to read about operators before coming to class but they did not read it…I noticed some students still don't have a photocopy of the Scratch programming handout…the other handout on QBASIC gave to them, they did not also make the photocopies"* (classroom observation, 30 March 2016).

Category 2.3.3 – Student well-being

The last category in this subtheme describes the participants' well-being as it impacted on the learning of programming. Findings show that "*student A1 has been sick, though she is in the class today*" (classroom observation, 17 February 2016). She could not really participate in class. Also, the mood of the students affected their understanding of the topics learnt. Findings, supporting students' fluctuating moods, are reported in the students' reflective learning journal. For example, "*I understand a little because I wasn't in happy mood that day*" (Mercy, reflective journal) and *"test result came out and I didn't do so well, so I felt bad throughout the class...another test, I did poorly and it ruined my mood…did another test and I did wonderfully. I'm proud of my scores*" (Percy, reflective journal).

Subtheme 2.4 - Contextual problem as impediment to the teaching and learning of programming

This subtheme describes the influence of identified problems in the context of the teaching of programming. It also discusses suggested solution to these problems. The subtheme is unpacked in the categories discussed below.

Category 2.4.1 – Contextual problem

The problem relates to those identified within the context during the teaching and learning of programming. Negative and unanticipated occurrences, such as electricity failure, faulty and limited computers, student enrolment, practical issues, projector fault and clashing of courses during the learning of programming, were impediments to obtaining programming knowledge. The participants were unhappy when this occurred as they were unable to constantly run their programs on the computer, as anticipated. Teaching process was interrupted because of lack of electricity during the 3rd, 5th, 7th, 10th and 12th week of the semester. A classroom observation corroborates the above findings, *"the electricity started fluctuating, and some systems switched off. As a result, students can't work individually. They had to work in a group which was not the intention of the teacher for the day"* (observation transcript, Week 11, 30 March 2016). One participant showed his displeasure when the computer system was switched off while he was busy programming. Program

scripts disappeared necessitating re-programming due to electricity failure. In support of this finding, one participant said, *"so due to light failure and others so I couldn't save I have to start again"* (Ramsey, interview excerpt). With faulty computers, one participant complained thus, *"like at times, we will want to work on the computer, the computer won't work. We now have to be sharing with our friends or something, and is not meant to be so*". Another factor influencing the students' learning of programming was student enrollment. Because the college had low student enrollment for the session, admission process was open till almost end of the semester. Students who had thus not resumed between the first and third week of the semester, during which the foundations of programming were taught, experienced problems understanding the course. For example, during the fourth week "*three other students joined the class and they were warmly welcomed*" and in week eight "*then I noticed another new student just joined today*" (classroom observation, 10 February and 9 March 2016). Clashes between lecture times were evident causing one of the active participants in the class to miss programming classes for three weeks. He said *"I have been missing French class for some weeks and I needed to also attend French classes"* (classroom observation 30 March 2016). Due to the position of the whiteboard in the laboratory, *"some of the students complained they can't see from the back and some of them even came to the front to see the program that has been run"* (classroom observation, 2 April 2016).

Category 2.4.2 – Suggested solutions to contextual problem

Different solutions were identified by the participants towards improving the teaching and learning of programming in the college. These solutions centred on the provision of electricity, *"they should make the light to be constant in order for us to work"* (Vanessa, interview excerpt) and the provision of computers, *"the computers they should make it enough and for it to be working well…everybody should have a computer each in order to practice it. They should have more computers"* (Vanessa, interview excerpt). One participant suggested that lecture time be reduced. This suggestion stemmed from the prolonged time the lecturer used to teach programming and, most especially, the Saturday extra classes. She said, "*but with time like coming to school sometimes, like you won't eat to school when you come to school is like it's stressful. Although you are here to learn but if you can like use the time you use per day. But if you can like reduce the time we use per day it will be okay. Like students will like to come to class, but like just using three or four hours sitting down… students they want something that one hour, two hours let me go like interact even if am not taking any courses, let me just walk around, exercise. Then I can come back"* (Nancy, interview except).

5.3.2.3 Theme 3: Mental representation during programming

This theme describes the content of the participants' comprehension activity obtained from the programming codes provided (Appendix A 5) and their affective and behavioural states during programming. The subthemes atom, relations and macro levels, explain participants' representations. The thematic network for Theme 3 is depicted in Figure 5.30.

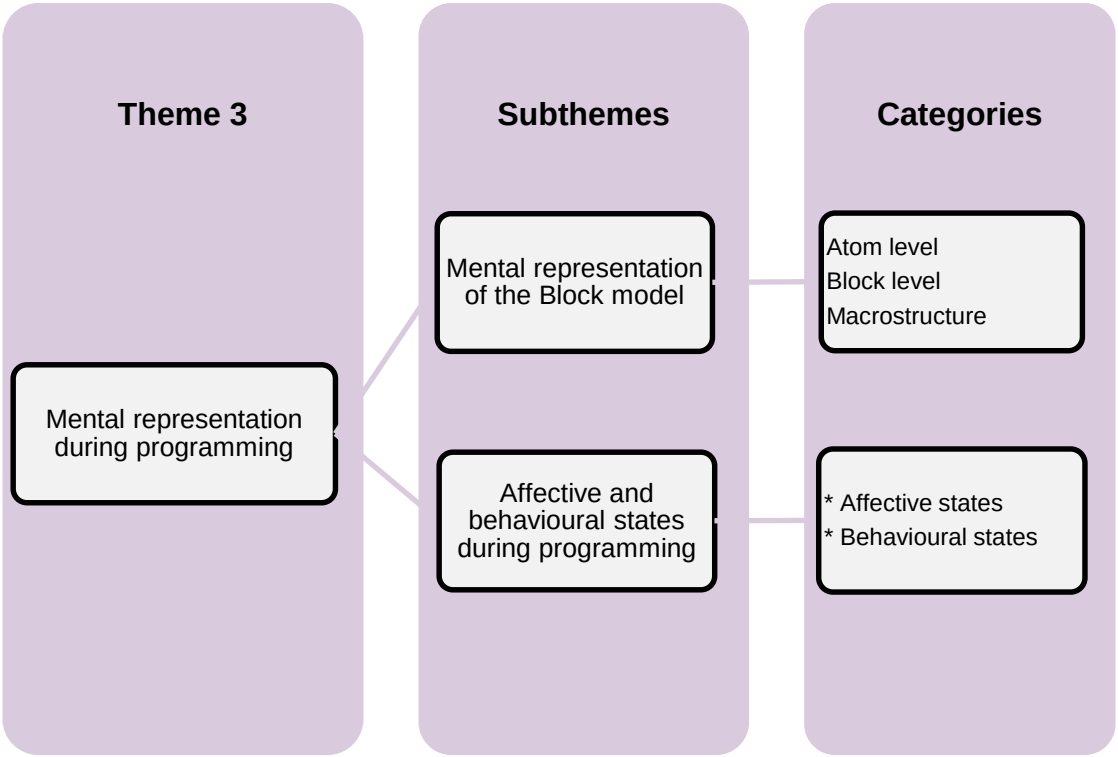


Figure 5.30: Thematic map of Theme 3

Subtheme 3.1: Mental representation of the Block model

This subtheme describes evidence of the participants' representation of the programming codes on the Block model given. Their representations are discussed according to each comprehension dimension of the Block model (*q.v.* *Table 2*.2) in categories 3.1.1 to 3.1.3. Evidence from the participants' interview excerpts supports the findings.

Category 3.1.1 – Atom level

This subtheme describes participants' mental representations on the lowest knowledge dimension of the Block model (see Section 2.4.2). This level is where participants first familiarise themselves with the programming concepts. The atom level comprises three comprehension dimensions with the respective questions - the text surface (question 1), the functions (question 3) and program execution (question 4). These questions can be found in Appendix A 5. On the *text surface*, participants could define what a variable is and identify

variables in a code. They explained a variable as "*a container that can house, ranging from numbers to symbols. It can contain numbers and when it contains numbers it is called numeric variables when it contains letters, we call it string variables"* (Percy). Bassey gave an explicit explanation with an example of a variable as *"if you are given a program to calculate the area of a circle or area of a square, Area = Length * Breadth. So, I understand that length and that breadth they are in the Area…so that Area is a variable".* However, in code 2, Mercy identified the variables as *"Counter = Counter + 1" and "Sum = Sum + 1".* From the answers given, Mercy's representation reveals that she could identify Counter and Sum as variables but when they are combined with a value, she still saw them as variables. She, therefore, had a misconception about variables. To query their representation further, they were asked to differentiate between a numeric, string variable and constants. Some of the participants could not confidently differentiate between a string variable and a string constant. Some of them said *"but am not sure…but I think…string constant… have a dollar sign instead of double quotation marks* (Bassey). This may impact on how they define variables when writing a program.

Along with *program execution*, operation of a statement involves determining students' representation of the execution of a single line of code in a program. Participants could explain the operation of a line of code presented to them in code 2. For example, while describing their representation of 'COUNTER = COUNTER + 2', Mercy said, *"it will be counting 2 times".* Here, Mercy could form a representation of the same code she identified as a variable, to meaning counting in multiples of two. Though she had a misconception that when used as execution, it is still a variable, she could identify the function of the statement in the program. However, in code 3, Bassey saw COUNTER = 0 as a form of initialisation whereby the computer at 0 will make the first count, and then make a second count as 1. His representation showed that he had not formed a correct representation of the execution of initialisation at 0 and at 1 during program execution on the notional machine. On the *function*s comprehension dimension, the participants' representation captured the essence of the INPUT command (code 1). For example, one participant noted that he understood the function of INPUT in the program to mean *"the INPUT asks for the company's name…the computer expects the user to INPUT the expected value"* (Bassey). His representation captured the meaning of the command as used in the program.

Category 3.1.2 – Block level

This category determines students' representation of sets of the statement within a program. In the line of code in code 2 below,

```
70 IF COUNTER <= 100 THEN GO TO 60
```

participants mentally represent the code to mean *"like if you are making a decision. If it's correct then you print but if not you back…like starting all over again"* (Nancy). She was able to form a representation that the code was testing a condition which requires the program to perform some operation if the condition is met, else the control moves to the top of the program. Another participant had a different representation of the code, as she said *"that means we will continue counting the counter + 1, another plus 1 until It reaches 100"* (Mercy). Her own representation of the block is to count in multiples of one. Her explanation shows she was able to relate the code to the whole program, but she was expected to mention (counter + 2). This may be because she could not remember the exact code since she was not allowed to go back to the program again. However, Bassey formed another representation of the code, *"that statement is the...conditional statement, that IF...THEN. I think is telling us that IF COUNTER <= 100 THEN GOTO 60, and that 60 is the place they start calculating the SUM. So IF COUNTER <= 100 you will go back to SUM and start but if it's not you have to END it*". He first identified IF…THEN as a conditional statement instead of an unconditional statement. He was able to form a representation of the flow of the program and the next line of code to be executed. His representation depicts that if the count is not up to 100, the control moves to 60 where the sum is calculated, else the program will end since the count has reached 100.

Category 3.1.3 – Macrostructure level

In this category, participants could form a representation of the program goal by using the REM statement. Some errors were also introduced into the program which they were expected to trace and then introduce the correct code as they comprehend the whole program. Even though the title of the program was not given, one participant was observed reading through the program to mentally determine the goal of the program. For example, in code 1, Ramsey said *"because in QBASIC program we have what is called REM, and that REM is just like stating what you want to work upon, what you want to achieve at the end of everything. So, I used that REM in answering the first question. Like in number one the REM was the program is to get a commission of a particular company".* Another participant, who answered the same question, did not mention that he used the REM statement but was able to explain that a program was done by first checking the condition the program was testing. He utilised his tracing skill to do this. Percy said *"when I look at the program I saw a statement there that IF sales or IF counter < 20 THEN go back to Counter. So, I realised that the Counter must be 20 and above or maybe 21 before the program will be able to end. So, I realise that the program was going to calculate the profit of 20 company salesman"*. His, (Percy's) representation directly captured the essence of the code. In code 2, Mercy, Nancy and Bassey's representations of the code were, *"program*

to compute set of numbers", "program to compute numbers less than or equal to 100", "average age of students starting from 10 to 100" which did not really capture the goal of the program. They were only able to form a representation that the program was calculating some sets of numbers between 1 and 100 whereas the program was calculating even numbers between 1 and 100. They did not explain how they formed the representation of the program goal. In addition, they determined the goal using different ways.

The findings from the participants' mental representations showed that there was no cross-referencing in their representation as formerly theorised by Schulte et al. (2010). One reason is that the retrospective questions were *not* designed to cater for cross-referencing (*the ability of a programmer to link elements and express his/her mental representation*) at the blocks and relation levels (see Table 5.8). Another reason could be the way teaching sequence was structured in a systematic order where students developed a detailed mental representation of each concept. Therefore, they were unable to cross-reference because there were not enough data for some of the blocks.

Table 5.10: Mapping of Block model to SOLO taxonomy

Macrostructure	Relational		
Relations			
Blocks		Cross-referencing	ulti-structural
Atoms	Uni-structural	Uni-structural	Uni-structural
	Text surface	**Program execution**	**Function**

When the questions were mapped to the SOLO classification and Block model, a relationship existed at the atoms and macrostructure levels (see Table 5.8). Findings revealed a multi-structural classification at the relations level. This supported the findings of Whalley and Kasto (2013). However, there was no classification at the relation level. I note here that the questions used by the authors were not the same as those used in this study.

Subtheme 3.2: Affective and Behavioral states

This subtheme describes the participants' affective states and behaviours when running some sets of program codes on the computer. They were all asked to rewrite their program using either a FOR…NEXT, or a subroutine statement. Follow-up questions were asked in the retrospective. They were also observed when running the program. The three categories that depict the findings are: confusion, figure out and behavioural states. The findings are discussed in categories 3.3.1 to 3.3.3.

Category 3.2.1 – Affective states (confusion)

In this category, the participant could not rewrite the problem in code 2. She did not understand how to use a FOR…NEXT statement. She was only able to write the following codes, as depicted in Figure 5.31.

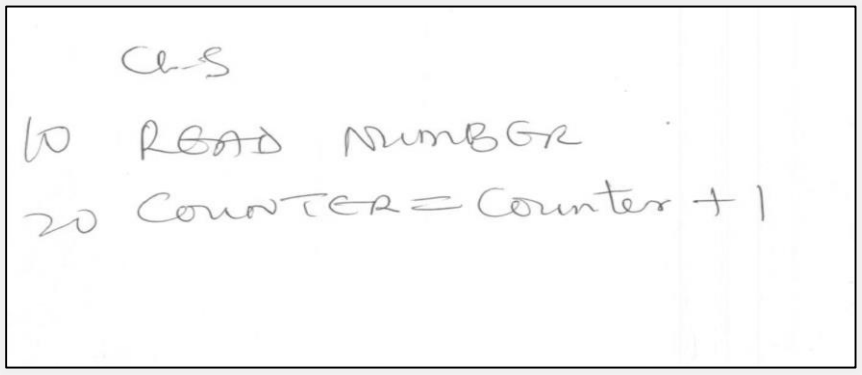


Figure 5.31: Sample of Mercy's program

In her program, she asked the compiler to read the variable NUMBER without specifying the corresponding data. She also made use of the counter. She did not complete the program and, as such, was unable to run it. There was no indication of a FOR…NEXT statement in the program codes. In a retrospective interview, the participant said,

"Ha! I don't even… I can't get it because I don't know it at all" (Mercy).

I also observed that Mercy was somewhat confused when trying to write her code on the computer system, as she remained stuck in one spot for about five minutes. She later cancelled the codes and started writing the program again. When asked what she was thinking, she said *"the INPUT number, you said am to use the FOR…NEXT statement, so I don't know how to use that INPUT number with the FOR…NEXT statement. Am confused maybe am going to remove the INPUT number and put the FOR…NEXT"*. From her statement it can be deduced that not only was she unable to understand the use of the FOR…NEXT statement, and the context in which it is used, but her inability to identify some keyboard functions also contributed to it. This was evident when she supplied her reason for studying computer science education at the college, *"I have never operated the computer and I want to know, to know as in what it does, its function and how to operate on it"*. The participant was thinking of two things, she was thinking about which key to press and at the same time how to use the FOR…NEXT statement in writing the required program.

In addition, participants' lack of confidence when writing their program codes also led to confusion. One participants said*, "will I really get the actual answer, that's what was, I was like when will I repeat it again. Like in my mind will you get this answer? Okay, let me go, like the last one I did, using the FOR…NEXT, I know am going to use the DIM, then how*

will I use it. What will I do?" (Nancy). Nancy's statement shows she was not sure of what to do and she was anxious about the exercise, especially when the program was giving errors. She felt she was not confident enough to solve the problem. The codes from her program are supplied in Figure 5.32.

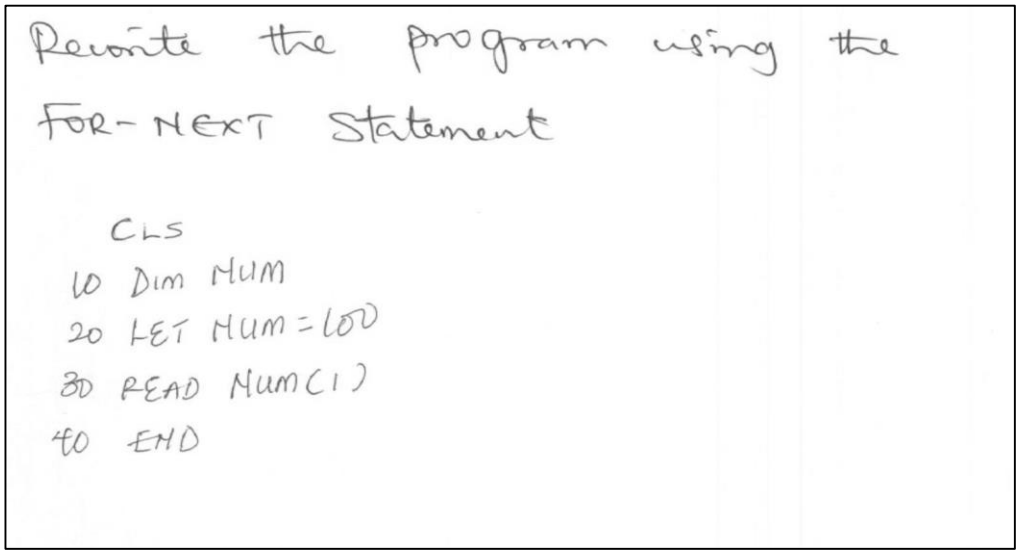


Figure 5.32: Sample of Nancy's program

Even when she was supposed to debug, she was unable to identify the specific error the compiler was referring to. She only tried to press some commands on the keyboard which removed the comment accidentally. Looking at her solution, she did not indicate FOR…NEXT statement as required but used the DIM statement instead. She must have seen, in class, where the DIM statement was used in conjunction with the FOR…NEXT statement. Here she had not really understood the context in which the FOR...NEXT statement could be used. Because of their lack of confidence and resulting confusion, they were not sure of which statement to use for an action. That was why another participant said, *"thinking should I use the INPUT, should I do it the way I use to do the INPUT or should I just use my understanding to do it"* (Vanessa). Another participant added, "*I was thinking that how am I going to put this FOR…NEXT and how am I going to use this FOR…NEXT in this type of program that I was doing*" (Bassey). This shows that they do not know the function of the FOR…NEXT statement, much less using it in writing a program. Another reason why they were not sure was that they did not really form a viable mental representation of the main program given to them. This may have affected how they used the FOR…NEXT statement. In addition, the FOR...NEXT statement was only taught once towards the end of the semester and, by implication, the participants may not have had a complete understanding of the use of the statement.

Category 3.2.2 – Affective states (figure it out)

Here the participants noted that while running the program, and still confused, they linked the program to what they had done in the previous class, hence their schema was activated. One of the participants said, *"I just remember some things, I remember that you…said the J will come before I and when we use READ, after READ we use NEXT before we put the DATA. That was just was coming to my mind. So, I was trying to use that idea but I did not get it"* (Bassey). Other participants said, *"when I was writing the FOR…NEXT, I remembered the way I was taught to write in class" (Vanessa) and "then later I just remembered okay like am supposed to like to put, when I remembered program that we write, when am, okay let me just try if It can get it"* (Nancy). From their explanation, the participants could only remember the exercises done in the class on when FOR…NEXT statement was used within an array and that was why their solution reflects DIM. However, they did not understand that the 'DIM' cannot be used in the context of their own solution. Their program was based on repetition. Therefore, they had a misconception on when and how the FOR…NEXT structure should be used in a program.

Category 3.2.3 – Behavioural states

Behavioural states describe participants' behaviour during programming. One participant was observed flicking her fingers while writing her program (video replayed), and she said *"I want to press delete and to…assuming am writing like this, so to come to the next line to…. I did not get them"* (Mercy).

5.4 META REFLECTION

This section focuses on my reflections about the first cycle action research, and how the research has contributed to my professional development and teaching practice. As stressed by Munn-Giddings and Winter (2001), one principle of action research is that the researcher learns about his/her own practice. The discussions here are guided by Schön's (1983) reflective practice for continuous professional development. The focus is on learning through self-awareness of my practice and personal experiences gained during the research process. These reflections came into focus through critical questioning which helped me improve my practice. I have learnt a lot about myself, my students and how engaging the teaching of programming skills, from visual to procedural, within a whole brain and constructivist teaching environment could be. I will first address my strengths and learning gained, then my weaknesses as well as areas where improvements are needed. This will be done in relation to my students' and the non-participant observer's feedback on teaching, as well as my personal reflection.

To facilitate first year students' programming skills, from visual to procedural, I moved away from the teacher-centred approach which I usually employed. I made the content more student centred, by focusing on *how* first year students learn programming skills within a holistic environment. The way I teach my students changed from asking the *what?* questions to the *why?* and *how?* questions. This encouraged critical thinking, meaning construction and social interactions between my students and myself as well as amongst students. I also realised that welcoming my students' opinion at every stage was very important as it imbued them with a sense of belonging in their studies. As part of my learning, I regarded my students as *co-researchers*, and I was aware that their collaboration during the project had a major impact on the study. By consulting my students, I regarded them as agents of change on issues that concern them. I encouraged my students to reflect on their learning by using a journal and completing a feedback survey which was a step forward in my teaching practice.

I realised that the new way of teaching programming to my students was interesting, but that it involved a lot of commitment from both the teacher and the students. It was *demanding* to facilitate programming based on the planned instruction. At some point I was so engrossed in different activities that I found it challenging to write my reflections, collate data and analyse it. Nevertheless, I see myself becoming a reflective teacher who is committed to knowing *how* her students understand programming concepts.
However, putting the constructivist learning theory into practice was not easy in the beginning. As the weeks passed, I gained understanding and became increasingly interested in the process. At first, I worried about finishing a planned topic for the day because in constructivism, the students are in control. But later, I realised that using a constructivist learning does not entail finishing a syllabus, but ensuring students construct meaning on a topic and relate it to other learning domains. Another constructivist learning aspect I struggled with was that student errors were treated as their own construction. I was tempted to supply answers without allowing students to construct meaning in the third week of the programming classes. I still see myself adjusting to the *meaning construction* aspect of constructivist teaching. I encouraged social interaction by allowing students to work in groups and discuss how to solve a problem (*q.v.* section 5.2.3).

I constantly checked my students' progress and difficulties regarding certain concepts. Such reflections necessitated the re-teaching of concepts such as algorithms, variables and expressions that students did not understand when it was first taught. The reflective aspect of teaching changed me into a teacher who understands the significance of constant reflections and who carefully thinks how my choices and actions can impact on my teaching

and my students. I have therefore learned that I need to always teach with my students' well-being in mind. The teaching of programming should reflect comprehension and foster the development of skills, not just teaching for the attainment of the grade. With my former teaching methods I usually followed the planned instruction in a bid to ensure the completion of the syllabus but without ensuring that my students fully understand a concept. One important thing I learned, which caused me to evaluate my past teaching practices, was my students' testimony regarding the warning they had received from their peers and relatives about their choice to study computer science, especially programming. With this information as background, I understand that my present actions, skills, interests, teaching strategies, attitude and commitment as a programming educator will either *encourage* or *discourage* my students, and other upcoming students, who want to study programming. I do not view this responsibility lightly and will thus remain committed to my professional development. I was encouraged when my students noted that they now have a passion for programming because of the learning they gained in Scratch programming. My learning journey as regards the design of tests using the SOLO and the revised Bloom's taxonomies has also improved. I designed tests using the Bloom's taxonomy before my professional development began. The use of the revised Bloom's and SOLO taxonomies for designing assessments have, however, taught me that I need to think critically and investigate my students' responses further in an effort to bolster my professional development.

Planning instructions based on the *whole brain*, I have learned that programming can be taught in different ways (see section 5.2.2). I have never planned programming instruction using different activities such as the one used in the whole brain. My supervisor introduced me to whole brain teaching and learning. This has changed the way I think when planning programming instructions. The process has been most interesting and enjoyable. However, planning on paper is easy but translating this planning into practice can be quite demanding. I learned that students sometimes want to explore, do research and be in control of their learning. They do not want the teacher to always tell them everything, contrary to the way I always thought when teaching. Reflective feedback from my students helped me to ascertain what kind of teacher I am. The students regarded me as a *caring teacher* who uses all possible means to ensure that they understand a concept. They noted that I inspired them by always showing great interest in programming and co-creating an environment conducive to teaching and learning. My students liked to learn in a group and do projects together, but they disliked it when some members of the group did not cooperate. They preferred to do practical work with technology. They became motivated to learn Scratch programming and most of them even waited at the end of the class, or came back during the week, to practice. However, they felt the teaching of QBASIC programming should be

the focus of the class since the exam would be based on QBASIC. Their reactions toward the use of technology (computer) for practical purposes and paper-based programming made me realise that they become motivated to learn programming when they practice on the computer.

Areas where I still need to improve, as noted by my students in their feedback, include linking learning to real life situation. Thereby, I did not live up to the values which I brought into the study. From my own assessment I also noted that I did not always link learning to real-life situations. Although this was not noted by the non-participant observer, I choose to take heed of my students' observations and I will improve upon this in the next cycle of the study. Other areas which the students pointed out, and which I also identified as a weakness, were the management of time and including introverted students in my teaching. The observer did not record these as weaknesses, although he did not observe all the class sessions. The use of Scratch programming handouts, as learning material, was challenging for the students because most of the photocopied handouts were not clear. Another weakness I noted related to the use of role-playing in facilitating the learning of programming. I asked my students to perform a role-play activity, without fully explaining the roles they were to play and, consequently, the presentations did not go to well. I will, therefore, learn more about the use role-play in teaching programming concept in the next cycle, because I see it as a teaching method that will encourage schema activation. The planning of programming assessments is another area that I need to focus on in the next cycle of the study. The idea *behind* the use of the Block model was to plan assessments, from the atom to the macro level, and to monitor the progression of the learning as per the blocks. However, I did not incorporate this *before* but rather *after* the assessment had been planned. This did not encourage cross-referencing in the students' mental representation (see Table 5.8). In the next cycle, I will focus on this area. Other areas which I do not personally view as weaknesses, but which occurred in context were: electrical failures, non-functional computers and clashing of classes.

Lastly, I discovered that the participants were not comfortable with thinking aloud during programming activities. Programming itself is a complex activity that requires a higher level of thinking. I realised that I needed to identify other means through which students' thinking could be recorded including the use of a video recording of participants' programming sessions which could be used as a pointer to the retrospective interview. During teaching and learning, I discovered that the $TLPF_1$ was not linear as initially designed (*q.v*. Figure 3.12). During teaching I still referred to the *analysis stage* to determine the learning approaches of new students who had enrolled after the course resumed. The design,

develop, implement and evaluate stages influenced each other in the sense that the use of authentic assessments required moving back and forth between these stages of the $TLPF_1$.

5.5 AREAS FOR FURTHER IMPROVEMENT

Based on the reflections gained in the first cycle, I highlighted important areas which needed to be improved upon in the second cycle of the study. Possible measures needed for changes to occur are also explained.

- Management of time: I will time myself to stop teaching 10 minutes before the end of the class.
- Increase programming time by splitting Scratch and QBASIC classes: I will liaise with the head of the department and explain the students' expectations. I will also welcome student ideas and their responses will determine its implementation.
- Clashing of classes: I will make the timetable committee aware of the problem and it is hoped that changes will be made to the timetable.
- Readable Scratch handouts: I will ensure that photocopies of the Scratch textbook are readable by suggesting to the student where they can obtain the service. I will, however, not force them to do so.
- Redesign test of individual concepts: I will redesign the test of individual concepts to capture the lower level of the taxonomies as well as mapping to the Block model before the test is administered.
- I will focus more on variables, flowcharts and algorithms and the FOR…NEXT structure since students struggle with these concepts.
- I will use a simulated HBDI questionnaire to replace Kolb's because it was designed to align with the validated HBDI items. This does not mean the Kolb's inventory was useless.
- The need for reliable power and functional computers will be discussed with the head of the department, the dean of science as well as the works and services department in an effort to focus their attention on these issues.
- Redesign the $TLPF_1$ to produce the $TLPF_2$ while incorporating the issues raised in the meta reflection.

5.6 CHAPTER SUMMARY

This chapter discussed research results, research findings, meta reflection and areas for possible improvement in the second cycle action research. The research results detailed the data gathered during the acting phase of the study. Observations made on the data were also detailed, with both positive and negative accounts during teaching and from the participants. The findings section reflected on the research results with supporting evidence from the data. I further discussed a meta reflection of the first phase of the teaching and learning of programming based on my personal learning as well as student and non-participant observer reflections. In addition, possible improvements to the second cycle of the study and their implementations, were explained.

CHAPTER 6 - RESULTS AND FINDINGS: ACTION RESEARCH CYCLE 2

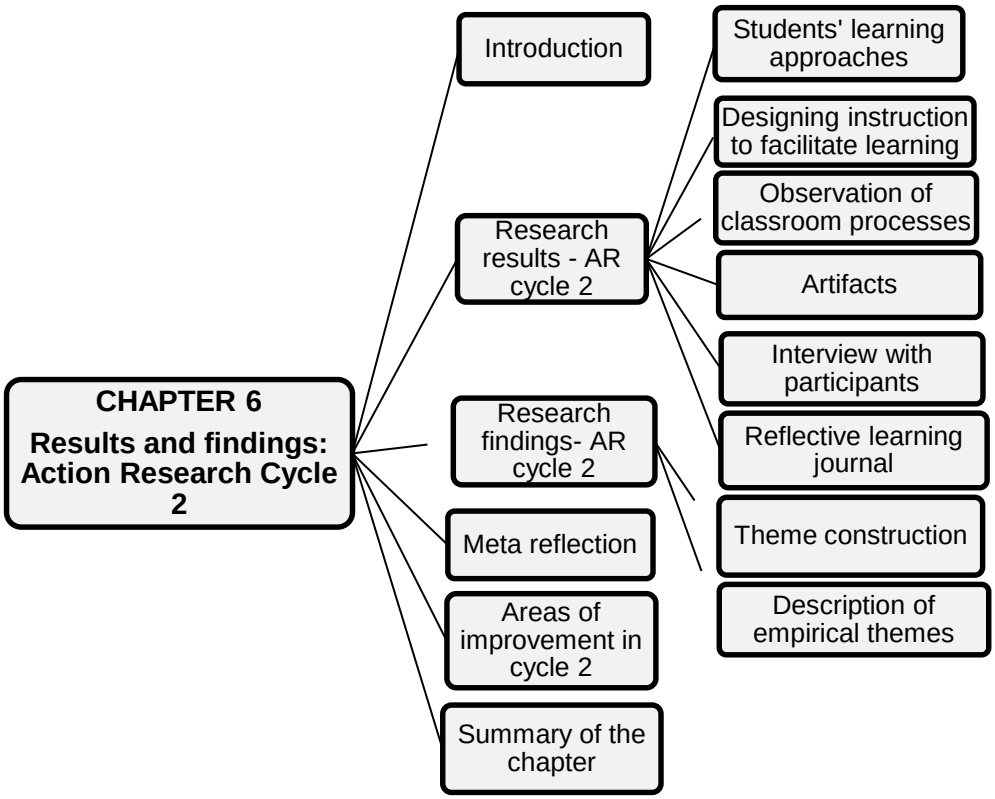


Figure 6.1: Diagrammatical representation of Chapter 6

6.1 INTRODUCTION

This section details the second cycle of the action research study. The graphical representation of the chapter is shown in Figure 6.1. The chapter is divided into five sections. The first section introduces the chapter. The second section, which is the research result, presents a discussion in six sub-sections dealing with: students' learning approaches, redesigning instruction to facilitate learning, observation of classroom processes, artefacts, interviews with participants and reflective learning journals. The third section discusses the findings gained from the result under three sub-sections namely: background information of participants, theme construction and description of empirical themes. Following this, the researcher's meta reflection on the second cycle with areas of improvement and further professional development highlighted, as discussed in the fourth section. Finally, the chapter concludes with a summary in the fifth section.

6.2 RESEARCH RESULTS – AR CYCLE 2

In this section, I present discussions on the data gathered. The section depicts *observing the effects of new action* in the Cycle 2 action research plan (*q.v*. Figure 4.10). In preparation

for this cycle, the areas for further improvement, as itemised in the first cycle (*q.v.* section 5.5), were taken into consideration. The timetable was first checked to ascertain the lecture time which was fixed from 16:00 to 18:00 on Fridays. This time slot did not bode well for electricity issues on campus and thus it would near impossible to facilitate practical programming sessions during this time. I approached the chairman of the time-table committee and requested a further amendment to the timetable. After many deliberations, the time was changed to 12:00 to 14:00 on Wednesday while the Friday time remained. The timetable change was communicated to the students. In addition, students' recommendation in the first cycle to extend class times was also discussed. The students supported the move after weighing different options. We, therefore, agreed on Wednesday and Friday for Scratch and procedural programming, respectively. Our decisions regarding the classes were communicated to the head of the department and informal approval was given.

Other issues such as ensuring that there were no clashes in class lecture times, redesigning the test on the individual concept and the use of simulated HBDI for determining students' learning approaches were also addressed. All these changes were incorporated into the teaching and learning process framework ($TLPF_2$) for the second cycle (see Figure 6.2). The framework was discussed in section 3.3. The motivation for redesigning the teaching and learning process framework was based on observations and reflections during the first cycle. I discovered that the components of the ADDIE were not linear in the real sense (see Figure 6.2). Teaching and learning decisions involve moving back and forth in the framework. For instance, after the *analysis phase*, I proceeded to the *design phase*, followed by the *develop phase*. During the implementation phase, teaching and learning activities were evaluated. Therefore, *evaluation* was not only done at the end of the semester, but most assessments were done *during* the semester. While designing tests, I determined whether the test questions (see Appendix B 1- Appendix B 3) designed constructively did in fact align with the course objectives. This however affected the *develop phase* where I reselected content, instructional strategies and supporting media, as the need arose.

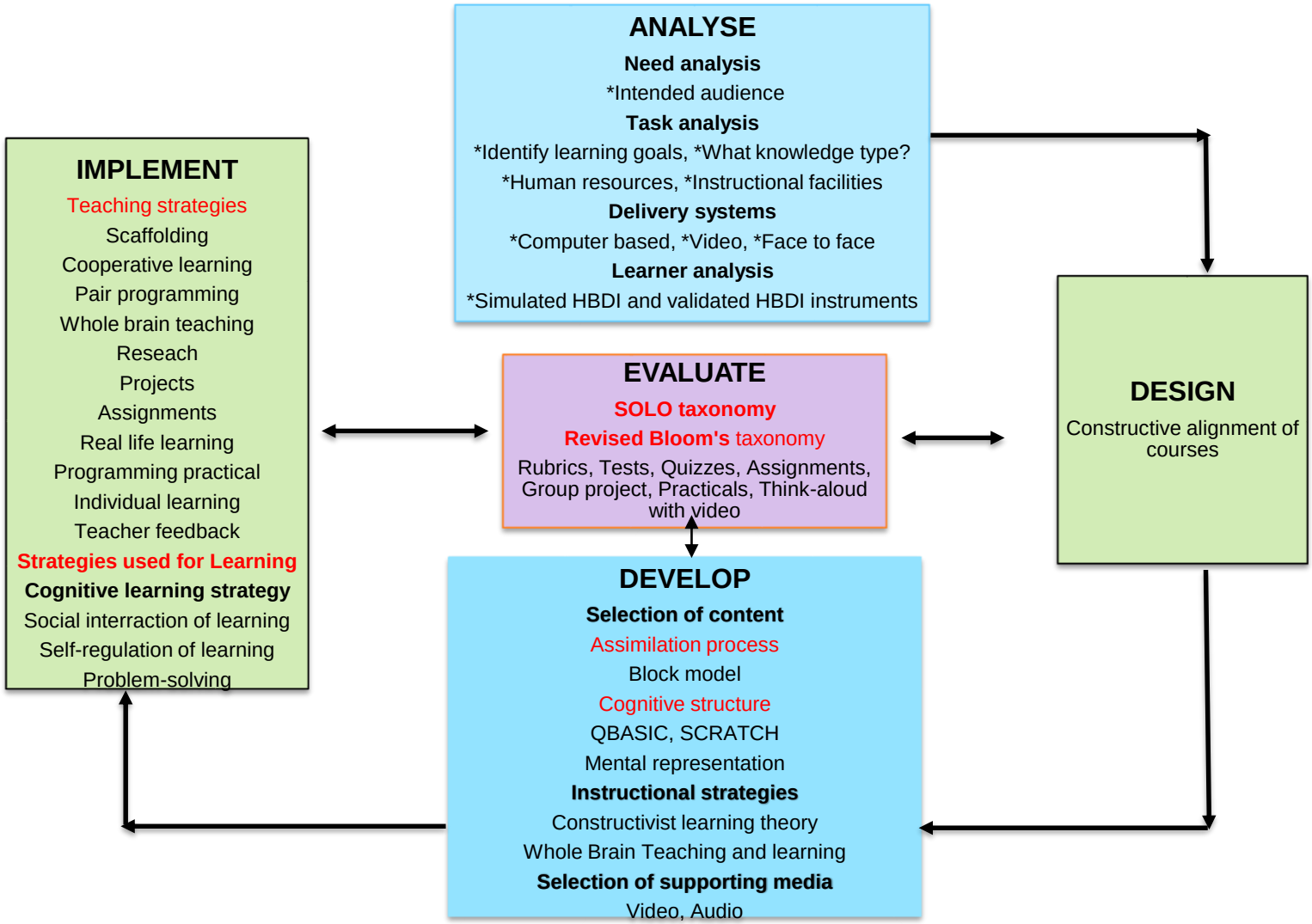


Figure 6.2: Improved teaching and learning process framework ($TLPF_2$)

I commence this section with a discussion of the students' learning approaches (*q.v.* section 6.2.1) which informed the design of instruction for the facilitation of learning (*q.v.* section 6.2.2). Following this, the observation of classroom processes (*q.v.* section 6.2.3) is explained with a focus on the artefact (*q.v.* section 6.2.4) of students' experiences in the programming classroom. The section concludes with a detailed discussion of the semi-structured interview (*q.v.* section 6.2.5) and reflective learning journal, as employed by the participants (*q.v.* section 6.2.6). Figure 6.3 illustrates a timeline of the data collected in the second cycle.

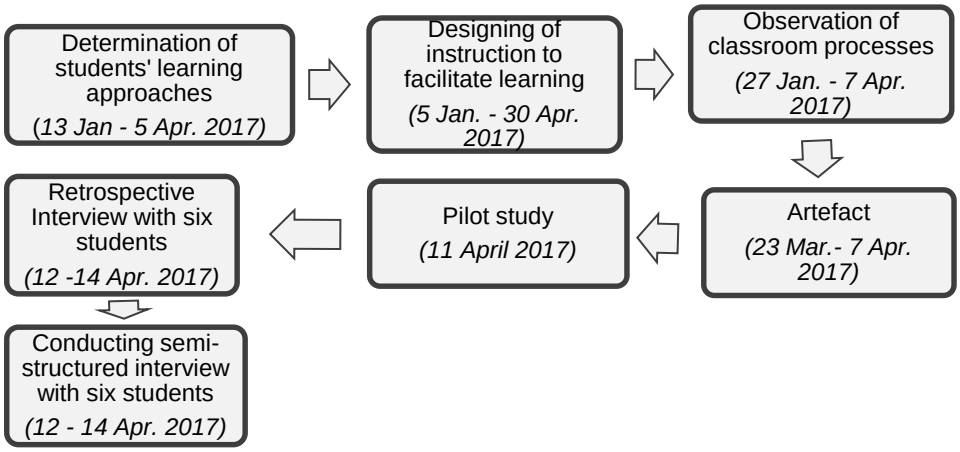


Figure 6.3: Timelines for data collection in the second cycle

6.2.1 Students' learning approaches

All the students were first exposed to a simulated HBDI, then the validated HBDI instrument was administered to nine students. The profiles of students generated from both instruments are discussed in sections 6.2.1.1 and 6.2.2.2, respectively. This process was conducted between 13 January and 5 April 2017.

6.2.1.1 Simulated HBDI profile scores of participants

The profile score of the participants was determined from the simulated HBDI (Appendix A 15) using the associated scoring key (Ngozo, 2012). As to the scoring process, the item number that matches each quadrant, as described in the instrument, were added together to determine the score for each quadrant. A high score represents a strong preference. Quadrants with the same high score represent a double dominance. The result shows that 30 of the participants had single dominance profiles with dominance in A quadrant (19%), B quadrant (28%), C quadrant (38%) and D quadrant (9%). The other participants had

double dominance, with dominant quadrants A and D (3%) as well as A and C (3%). Figure 6.4 presents a pie chart which summarises these profiles.

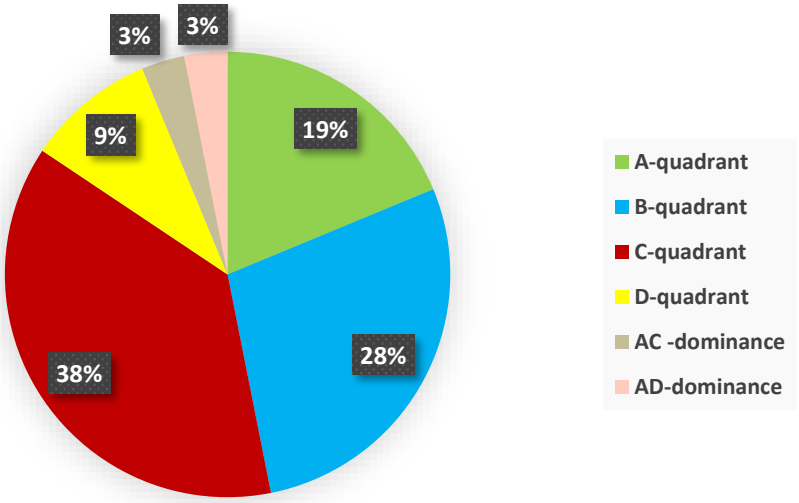


Figure 6.4: Students' simulated profile scores

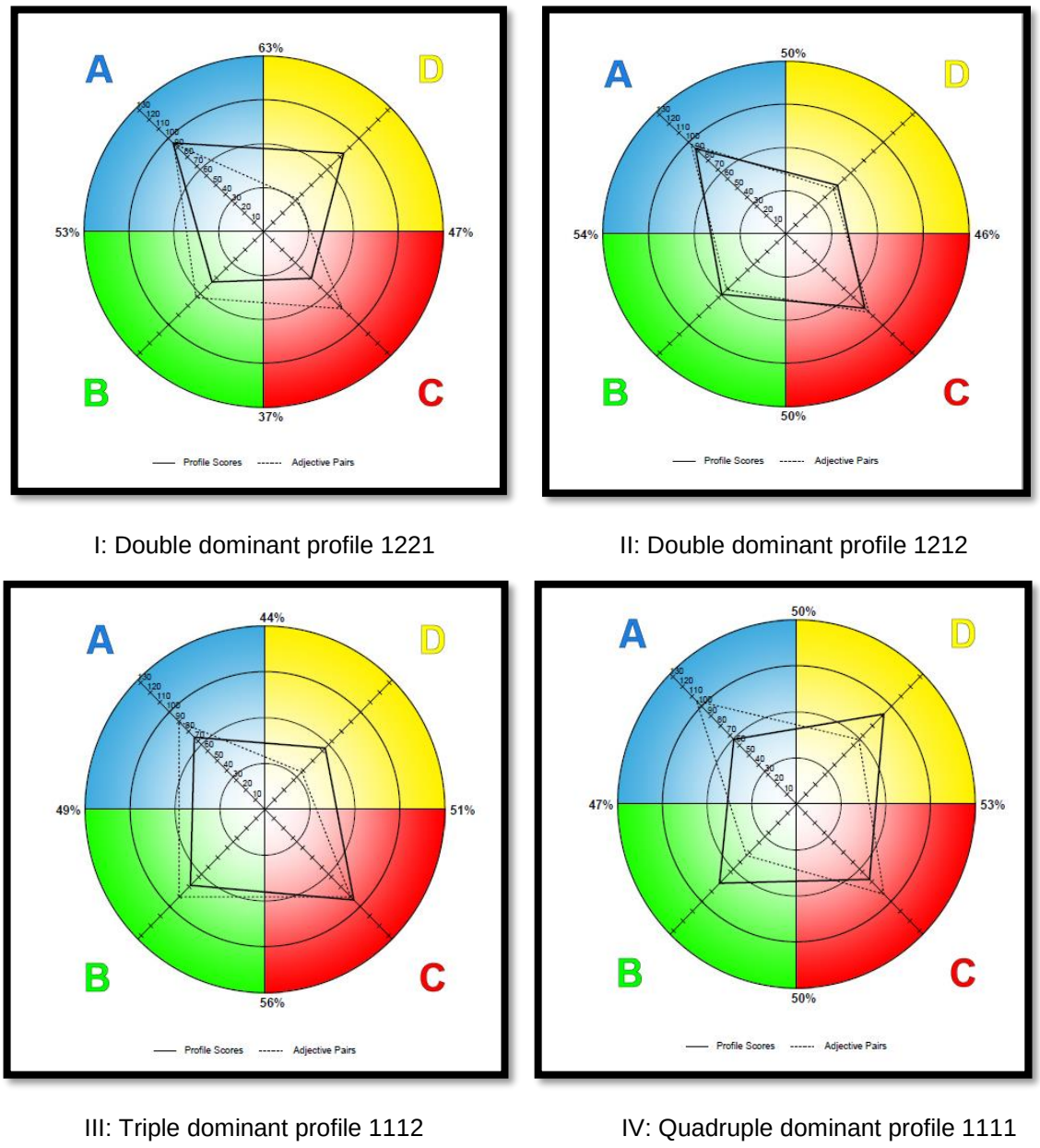
Understanding gained from the profile results suggested that I could conveniently accommodate participants who fall within the B and C quadrants. The pie chart shows that those quadrants are well represented in the classroom. Students, whose preferences are dominant in A and D or AD or AC, pose a challenge and I would consequently work harder at instructional planning and teaching designed to accommodate them. A further analysis of students' profiles is supplied in section 6.2.2.2.

6.2.2.2 HBDI profile of participants

In addition, a validated HBDI instrument was used to determine the brain dominance of the selected participants in the programming classroom during the sixth week of the semester. From the nine selected participants, two showed a strong preference for quadrant A, three for quadrant B, one for quadrant C and two for quadrant D. The HBDI profile overlay (see Figure 6.5) is the visual representation of the participants' profile scores which were read counter-clockwise, as ABCD, starting from the upper left. It was used to describe the participants' preference codes. The participants were either fully left brained or dominant in one quadrant of the left brain.

Profiles of the 9 participants are not discussed individually, but samples are shown to explain possible preferences in the programming classroom (*q.v.* Appendix A 21). Students' profiles, which are in contrast with mine, are discussed since my HBDI profile confirmed

that I can only accommodate students who share my profile. I therefore faced the challenge of adjusting my teaching and being flexible enough in the design of instruction as to accommodate all preferences present in my classroom.



I: Double dominant profile 1221 II: Double dominant profile 1212

III: Triple dominant profile 1112 IV: Quadruple dominant profile 1111

Figure 6.5: Samples of students' profiles
Source: Herrmann International (2017)

- *Profile I*: indicates a strong preference for quadrants A and D, while quadrants B and C are less preferred. First year students with this profile would desire rational, logical and quantitative approaches to thinking in quadrant A. Students who fall within the upper right D quadrant would be creative, holistic and integrative in their approach. However, a participant's preference for quadrants A and D would be challenging to

me and as a facilitator and I would need to move beyond my comfort zone to accommodate their expectations.

- *Profile II*: indicates a double dominant, diagonal axis between upper quadrant A and lower quadrant C. Participants with this profile show a strong preference for rational, logical and quantitative approaches to thinking in the A quadrant. They also show a preference for the emotional and interpersonal approach signified in the C quadrant. The processing modes in quadrants B and D are secondary, but at the same time functional. The most satisfying work for this type of participant includes making explanations, being involved in teamwork and doing challenging tasks (Herrmann International, 2016). I would possibly accommodate this participant based on my descriptors in the A quadrant as *logical* and *mathematical*. However, I need to consciously work at it since it is not my strongest preference.

- *Profile III*: is a triple dominant profile. The participants are strong in quadrants A, B and C while quadrant D is less preferred. I would most likely accommodate this profile, based on my preference. It is also expected that participants with these quadrants would perform well in programming because of their strong left-brain dominance. However, I would need to stretch myself to accommodate D quadrant students. Both the participants and I need to be aware that we do not prefer to think in the D quadrant.

- *Profile IV*: indicates a multi-dominant profile, having strong preferences in all four quadrants, though not all equal. A similar discussion of this quadrant was presented in Cycle 1 (see section 5.2.2.2)

6.2.2 Re-designing instruction to facilitate learning

The instructional design for the second cycle was based on the processes undertaken in Cycle 1 (*q.v.* section 5.2.2). The following discussions present a few exceptions from the first cycle. The participants were placed into homogenous groups of four, based on their simulated HBDI profiles. This was done so that students could learn with other students in their groups who have same dominance. A few weeks later, students were distributed into heterogeneous groups of four. This was done in an effort to support students to move *outside* their comfort zones (De Boer et al., 2013) and learn from other quadrants. This process could possibly help in the development of skills rooted in quadrants they less prefer. The whole brain way of facilitating learning was used to plan the programming instructions. The design was explained in Cycle 1 (*q.v*. section 5.2.2).

Different learning activities, associated with each quadrant, were combined within each instruction, as required by the topic. In this cycle, the topic *basic concepts of computing* was removed. Anecdotal evidence from students seem to suggest that the majority of them previously received instruction on this topic in secondary school. Table 6.1 summarises the re-designed instruction with: the dates, programming language taught, class duration, number of student present in each class, the whole brain approach and constructivist method used to facilitate learning. I focused more on *variables*, *flowchart*, *algorithm* and the *FOR…NEXT* concepts during the facilitation of learning. I informed students where they could make good quality copies of the learning material. Some of the students accepted my suggestion whilst others did not. The electricity issue remained a challenge in the beginning of the data collection. Electricity was only available from 12:00 to 13:00 which was *not* convenient for the class. Based on further deliberations with the staff in-charge, electricity was then made available from 12:00 to 14:00 daily. Other days of the week were used for assignments and personal practice. The following sections (6.2.2.1 and 6.2.2.2) present feedback from students, the non-participant observer (6.2.2.3) and a colleague (6.2.2.4) on my facilitation of learning.

Table 6.1: Instructional plan for programming during the second cycle action research

Date and Lessons (L)	Programming Language	Topic taught	Duration	No of students	Whole brain approach	Quadrant	Constructivist learning activities in the classroom
13/01/2017	Nil	Consent letters, familiarisation, filling of HBDI instrument, and deliberations regarding lecture time	2hrs	15	Nil	Nil	Learner control, social interaction.
17/01/2016 (L1)	Scratch	Thinking for computers (Algorithms)	2hrs	16	Brainstorming, clear examples, lecture and poem	A, B, C and D	Collaborative learning, brainstorming, teacher as facilitator, social interaction, meaning construction, authentic activities, multiple perspectives, situated learning, real-life examples, group learning.
24/01/2017 (L2)	QBASIC	Algorithms	2hrs	20	Lecture and group discussions	A and C	Group work, authentic activities, teacher as facilitator, social interaction, meaning construction, multiple perspectives, problem solving, previous knowledge construction.
25/012017 (L3)	Scratch	Algorithms	2hrs	22	Brainstorming, self-paced, group discussion exercises with steps	B, C and D	Collaborative learning, authentic activities, teacher as facilitator, social interaction, meaning construction, multiple perspectives, problem solving, previous, knowledge construction, cooperative learning (jigsaw).

01/02/2017 (L5)	Scratch	Algorithm with decision and repetition	2hrs	23	Presentation, brainstorming, self-paced, group discussion, exercises with steps, feedback on assignment	B, C and D	Collaborative learning, authentic activities, teacher as facilitator, social interaction, meaning construction, multiple perspectives, problem solving, previous knowledge construction, real-life problems, individual learning, group learning, rubric, cognitive conflict, learner control.
03/02/2017 (L6)	QBASIC	Project presentation and Test (test of individual concepts)	2hrs	24	Presentation, group discussions	A and C	Cooperative learning, rubric, teacher as facilitator, social interaction, meaning construction, multiple perspectives, authentic activities, rubric, group learning.
08/02/2017 (L7)	Scratch	Introduction to Scratch programming environment	2hrs	21	Powerpoint presentation, animations, practical (exploring and interacting), group discussions, visual illustrations, pair programming	C and D	Collaborative learning, authentic activities, teacher as facilitator, social interaction, meaning construction, multiple perspectives, problem solving, previous, knowledge construction, scaffolding, learner control, group learning.
10/2/2017 (L8)	QBASIC	Data types	2hrs	23	Reference book, lecture, group discussions, mind map	A and D	Authentic activities, teacher as facilitator, social interaction, meaning construction, multiple perspectives, problem solving, previous knowledge construction.
15/02/2017 (L9)	Scratch	Teach a Sprite to perform basic tasks	2hrs	25	Powerpoint presentation, pair programming, practical (exploration)	C and D	Collaborative learning, authentic activities, teacher as facilitator, social interaction, meaning construction, multiple perspectives, problem solving, previous knowledge construction, scaffolding, learner control, brainstorming.

17/02/2017 (L10)	QBASIC	Data types continued	2hrs	26	Poem formation, group work, group discussions	C	Cooperative learning, teacher as facilitator, social interaction, meaning construction, multiple perspectives, brainstorming, scaffolding, previous knowledge construction.
22/02/2017 (L11)	Scratch	Repetition	2hrs	25	Powerpoint presentation, practical practice (interacting), group discussions, pair programming, animations	B, C and D	Collaborative learning, authentic activities, teacher as facilitator, social interaction, meaning construction, multiple perspectives, problem solving, previous knowledge construction, meaning construction, reflective teaching.
24/02/2017 (L12)	QBASIC	BASIC Keywords	2hrs	15	Fact-based lectures, well-structured activities, problem solving with steps, practical (practices), group discussions	A, B and C	Cooperative learning, teacher as facilitator, social interaction, meaning construction, multiple perspectives, brainstorming, scaffolding, previous knowledge construction.
01/03/2017 (L13)	Scratch	Variables	2hs	17	Power point presentation, animations, practical (exploring and interacting), group discussions, visual illustrations, pair programming	C and D	*As L7 above* with error consideration.
03/03/2017	QBASIC	Test (test of individual concepts 2)	30mins	26	-		Teacher as facilitator.
08/03/2015 (L14)	Scratch	Presentation (repetition) and expression	2hs	26	Presentation, kinesthetic activities, checklist, learning laboratories (interacting), visual illustrations, brainstorming	C and D	* As L7 above with rubric.

10/03/2017 (L15)	QBASIC	Arithmetic Expressions	2hrs	22	Lecture, practical (practice), self-paced, sequential exercises	A and B	* As L5 above with Jigsaw
17/03/2017 (L16)	QBASIC	Writing of small program (control structures)	2hrs	29	Lecture, practical (practice), group work, sequential exercises	A and B	* As L5 above
18/03/2017 (L17)	QBASIC	Control structures and tests (interim test 1)	4hrs	29	Practical (practice), group discussions, problem solving with exercises	B and C	* As L5 above
24/03/2017 (L18)	QBASIC	Looping	2hrs	23	Lecture, practical (practice), sequential exercises, discussions	A, B and C	* As L5 above
29/03/2017 (L19)	QBASIC	Looping contd.	2hrs	26	Lecture, practical (practice), sequential exercises, discussions	A, B and C	* As L5 above
31/03/2017 (L20)	QBASIC	Initialising an array	2hrs	17	Lecture, practical (practice), sequential exercises, discussions	A B and C	* As L5 above
05/04/2017 (L21)	Scratch	Input/Output processing, decision making (Interim test 2)	2hrs	24	Powerpoint presentation, animations, practical (exploring and interacting), group discussions, visual illustrations, pair programming	C and D	*As L7 above

07/04/2017 (L22)	QBASIC	Array, subprogram, presentation and final test	4hrs	18	Presentation, lecture, practical (practice), sequential exercises, group discussions	B, C and D	* As L5 above

* As above, means the constructivist activities were repeated in the classes where the (*) appears in the column

6.2.2.1 Feedback on teaching and learning

The participants offered feedback on how I facilitated learning at the end of the teaching and learning process. Students' responses are illustrated in the figures presented below. As shown in Figures 6.6 to 6.8, the x-axis features letters a, b, c or d which represent the information participants responded to on the questionnaire (**Error! Reference source not found.**). The y-axis represents participants' responses in percentages. This section commences with a description of *how* learning was facilitated by the lecturer.

Learning facilitation

As shown in Figure 6.6, students' responses to 'item a', shows that 63% almost always and 33% frequently noted that I facilitated learning by showing a strong interest in programming and learning tasks. The remaining 4% said I did it occasionally. For 'item b', a large percentage (72%) of the participants noted that they liked my audible voice during teaching. The remaining 28% of the participants frequently support this. Additionally, responses from 'item c' show that 46% and 36% of the participants, respectively, noted that I frequently and almost always showed encouraging insight into the importance of programming and related problems. Other students occasionally (14%) and hardly ever (4%) saw me doing this. 'Item d' responses reveal that 66% of the participant noted that I almost always and frequently provided programming sessions that were encouraging and lively. The remaining 25% occasional and hardly ever supported the claim.

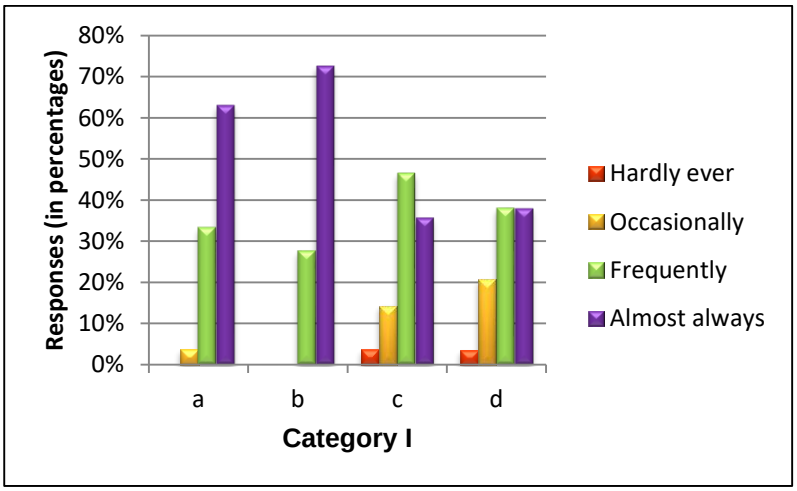


Figure 6.6: Learning facilitation feedback

Learning initiation

For 'item a', the participants noted that I frequently (55%) and almost always (17%) initiated learning by creating a climate that encouraged deep learning. The remaining 28% noted that it was either practiced occasionally or hardly ever. Likewise, for 'item b', the larger percentage noted that I almost always (48%) and frequently (35%) stated the objectives and learning outcomes of each classroom session. Thirteen and 4% respectively, said I either almost always or hardly ever practiced such. 'Item c' shows that 75% of the participants noted I frequently and almost always linked programming problems to real life situations. The remaining 25% said they occasionally or hardly ever saw me doing such. Figure 6.7 gives a summary of students' responses.

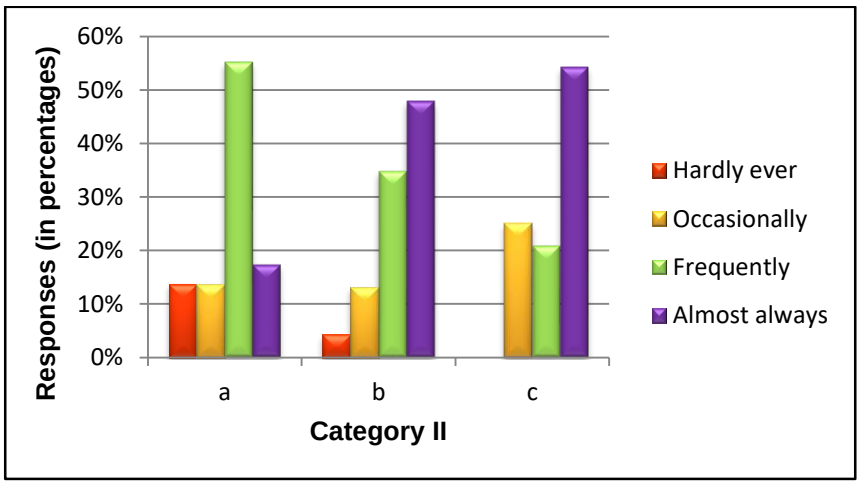


Figure 6.7: Learning initiation feedback

How the lecturer maintained learning

Figure 6.8 presents a bar chart on how the lecturer maintained programming learning during the semester. For 'item a', the larger percentage (86%) of students acknowledged that I frequently and almost always maintained an academic relationship that encouraged them to develop a questioning mind. However, 10% and 3% respectively, said I occasionally and hardly ever did. For 'item b', 64%, 32% and 4% respectively, supported that I almost always, frequently and occasionally encouraged them to construct their own understanding of programming concepts. Further explanation of 'item c' showed that 48%, 23% and 21% respectively, confirmed that I frequently, almost always and occasionally challenged them to learn programming beyond their comfort zones. The remaining 7% disconfirmed the claim by choosing hardly ever. For 'item d', the students established that I almost always (68%), frequently (25%) and occasionally (7%) allowed them to freely express themselves during the learning of programming. Students' responses to 'item e' revealed that 48%, 38% and 14% respectively, affirmed that I frequently, almost always and occasionally encouraged

them to critically think, reflect on learning as well as self-regulate their learning of programming. In addition, 64% and 36% supported that I almost always and frequently encouraged them to learn cooperatively in groups in 'item f'.

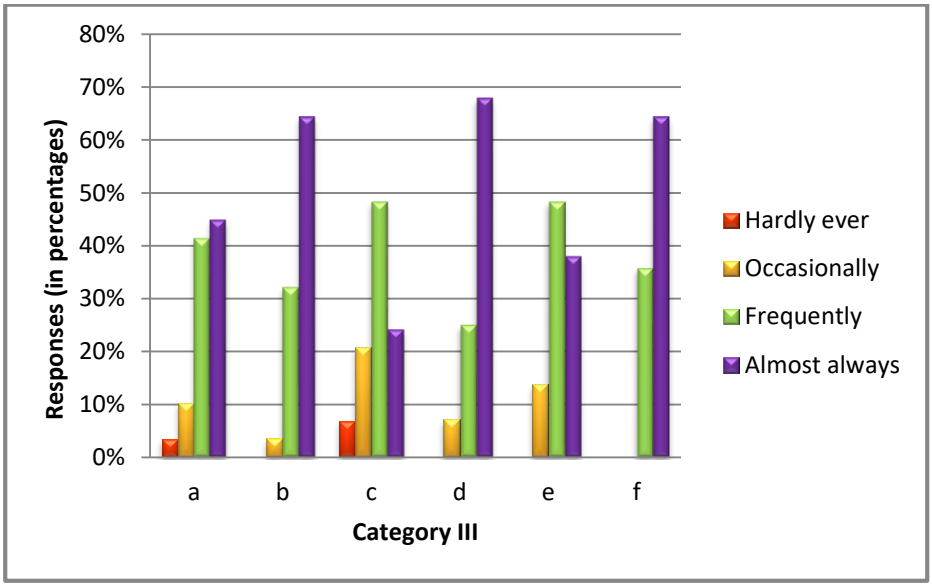


Figure 6.8: Feedback on how the lecturer maintained learning

6.2.2.2 Students' contribution to learning

The participants also assessed *themselves* and gave feedback through a survey. The results are presented in a discussion on students' contribution to their own and others' learning of programming during the semester. Other discussions focus on learning application and strategy. Their responses are illustrated in bar charts (see Figure 6.9). In Figures 6.9 to 6.11, the x-axis features letters a, b, c or d which represent the information the participants responded to on the questionnaire (*q.v.* Appendix A 15). The y-axis represents participants' responses in percentages. This section commences with students' contribution to their own and others' learning.

Students' contribution to their own and others' learning

Figure 6.9 depicts students' responses on how they contributed to their own and other students' learning in the programming classroom. For 'item a', the result shows that 90% almost always and frequently showed a strong interest in the learning of programming concepts in the classroom. The remaining 7% and 3% respectively, said they occasionally or hardly ever showed interest. Another way in which they contributed to their learning was noted in 'item b' where 52%, 28% and 14% respectively, noted that they frequently, almost always and occasionally expressed themselves well in the programming classroom. Only

7% hardly ever expressed themselves. They also noted in 'item c', that they frequently (69%) and almost always (7%) gained insight on the importance of each programming concept and related problem/s. The remaining 17% and 7% occasionally, or hardly ever, did. For 'item d', results show that students frequently (54%) and almost always (34%) contributed to their own and others' learning in the programming, such that each lesson was lively and encouraging. Only 11% noted that they hardly ever did.

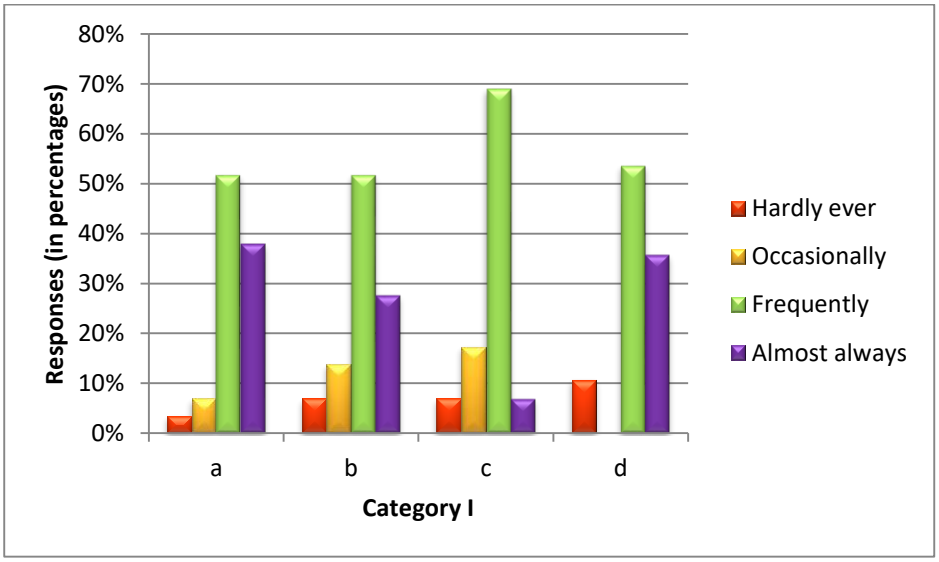


Figure 6.9: Feedback on students' contribution to learning

Students' learning application

Figure 6.10 represents *how* students applied the learning of programming to real-life situations to encourage deep learning. For 'item a', students' responses showed that they frequently (55%), almost always (20%) and occasionally (17%) co-created a climate conducive to the learning of programming. However, 7% noted that they hardly ever did. 'Item b' reveals that students frequently (48%) and almost always (32%) linked their learning of programming to real-life experiences. Only 20% of the students occasionally did. For 'item c', students frequently (46%), almost always (21%) and occasionally (14%) had a mental representation of the multidimensional nature of each programming topic learnt while 18% hardly ever did.

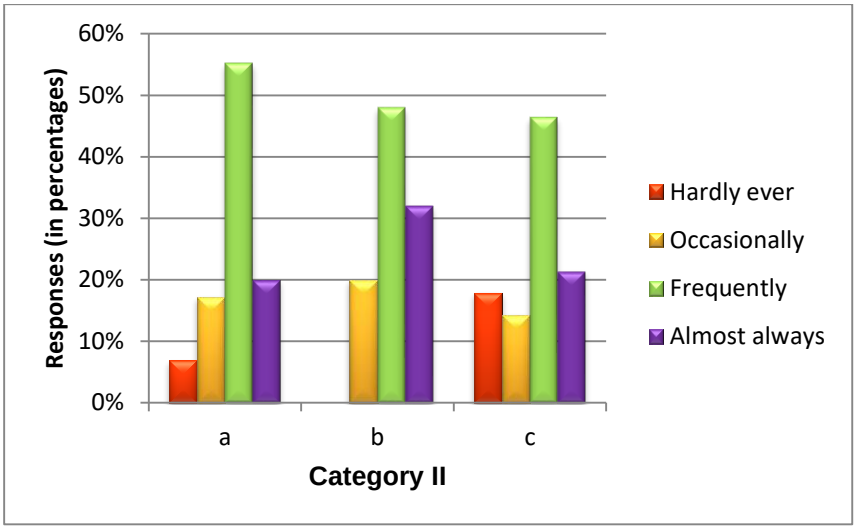


Figure 6.10: Students' learning application feedback

Students' learning strategy

Figure 6.11 illustrates the learning strategies students utilised in the programming classroom. For 'item a', students frequently (48%), almost always (34%) and occasionally (17%) developed an enquiring mind in the programming classroom through lecturer-student discussions. 'Item b' revealed that students frequently (51%), almost always (17%) and occasionally (21%), constructed understanding on programming concepts and activities done in the class. Thirteen % noted they hardly ever did. 'Item c' showed that students sought opportunities towards moving out of their comfort zones. Therefore, 38%, 34% and 14% respectively, claimed they frequently, almost always and occasionally did. However, 14% hardly ever left their comfort zones. Similarly, for 'item d', 37% and 41% respectively, frequently and almost always expressed themselves freely and openly in the classroom. Others occasionally (19%) and hardly ever (4%) did. On their attitude to learning ('item e'), 55% almost always and frequently reconsidered their former attitudes to learning while the remaining 41% and 13% occasionally and hardly ever did. This may, however, explain why in 'item f', 86% almost always and frequently had the opportunity to gain a better understanding. But, the remaining 10%, occasionally and hardly ever (13%) did.

Discussing 'item g' and 'h' together, 93% almost always and frequently developed a better sense of responsibility towards the learning of programming while 7% occasionally did. They, therefore, exhibited this responsibility further by noting that 45% frequently, 41% almost always and 14% occasionally contributed to their peers' learning by helping them find solutions to programming problems. Responses on 'item i' reveals there was a community of practice during learning programming. Since they frequently (52%), almost

always (25%) and occasionally (17%) took part in cooperative learning activities. Only 7% hardly ever did.

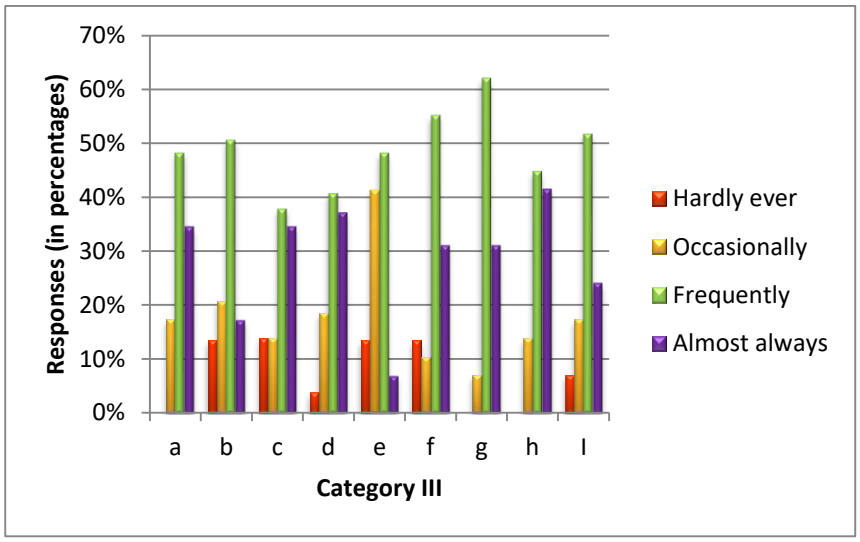


Figure 6.11: Feedback on students' learning strategy

6.2.2.3 Feedback from non-participant observer

The non-participant observer, who was present in the first cycle, also observed the programming classroom in this cycle. His observations as to teaching and learning in the programming classroom for this cycle corresponded with what he had observed in the first cycle. To avoid repetition, *q.v.* section 5.2.2.3 for explanation. He summarised his observation as follows: *"The union of teacher's methods and the instructional materials, such as projector, camera and the software (Scratch) actually had positive impact on the teaching-learning activities that took place. Thus, if such programming packages could be identified and used for teaching the students, learning programming would be made easier for student. Her classes were interesting, as she combined many teaching methods".*

6.2.2.4 Feedback from colleague

During the research process, a colleague (Mrs Rose), who also teaches computer science in the department was invited to observe some of the class sessions. She observed the students once in their groups and my teaching on several occasions. Table 6.2 gives a summary of her observations.

Table 6.2: Feedback from colleague

Date and Activity	Groups	Observations
01/03/2017 Variables in Scratch	B	"They made helpful suggestions in the class, encouraging their colleagues who do not get the problem easily to continue practicing it often and often"
	C	"The students showed disagreement in a constructive way"
	D	This group showed their disagreement in a constructive way and hereby learned more on the areas where they have problems"
	E	"Some of them attempted leadership opinions and were corrected"
	F	"...the student asked questions when they encountered problems and they were helped out by their instructor"
	H	"...students repeated the program often and often before getting it"

She also completed a feedback questionnaire in which she noted that I inspired my students by showing passion for the teaching of programming, as well as stating the significance of each programming concept through creating active and inspiring learning opportunities. She did not observe whether I initiated teaching by first stating the purpose and learning outcomes of each lesson, because she always joined the class halfway. Even at that, she affirmed that the programming classroom created a climate that promoted deep learning and where learning was linked to real-life situations. Furthermore, she observed that a linear relationship between the teacher and students on the discourse of learning, promoted the development of questioning minds in the students. She also supported that I: fostered critical thinking, encouraged self-reflection, promoted cooperative learning, provided scaffolds and opportunities for learning flexibility during the teaching and learning of programming. She concluded by supporting her claim with the following explanation: *"The teaching method was student-centered. The class was so interesting and the students really participated in the teaching-learning process. She encouraged the students on how to become self-dependent by solving problems on their own. Group learning was also incorporated into the learners. The teaching methods used, boost the academic performance of the students, which was discovered through the evaluation of performance of each student. She has good class management and the use of instructional materials through multi-media was adequate. The questioning and answering techniques were adequate. Generally, her teaching was full of enthusiasm."*

6.2.3 Observation of classroom processes

The classroom observation took place between 27 January and 7 April 2017. Twenty-one lessons were observed and two observations, one each from Scratch and QBASIC programming lessons, are discussed in this section. To begin these observations, the simulated profile scores of the participants were used to divide them into eight quadrant groups. There were two group As, two group Bs, three group Cs and one group D quadrant/s. For clarity, individual participants were identified by their group names till the ninth lesson (15 February 2017). However, when some of the participants were exposed to the validated HBDI survey, the class was regrouped in such a way that each whole brain quadrant was represented in each group. This was done so that participants could move out of their comfort zones and learn from their peers (Coffield et al., 2004). The names of the groups were further denoted as group A, group B, up to group H. The names of the participants, as used in this section, are pseudonyms. The procedure for interim analysis of data employed in the first cycle (*q.v*. section 5.2.3) was used in the data analysis in this cycle as well. The following discussions, sections 6.2.3.1 to 6.2.3.4, will analyse and describe the four lessons. A comprehensive analysis of all the twenty-one lessons can be found in Appendix C 3.

6.2.3.1 Classroom observation: Lesson 9

The concept of repetition was taught on Scratch in the ninth lesson. From the teaching activity of Lesson 9, ten codes were identified. These codes are: positive emotional climate (PEC), previous experience (PE), brainstorming (BRST), faulty computers (FC), pair programming (PP), student engagement (SE), meaning construction (MC), discovery learning (DL) and scaffolding by the teacher (SCT) and scaffolding by student (SCS). The corresponding categories are: self-regulation, word processing skill, learning strategy, contextual factor, constructivist teaching strategy and scaffolding. Table below 6.3 presents the breakdown of the analysis. These codes are further highlighted in sky-blue next to each coded sentences or phrases.

Table 6.3: Interim analysis of lesson nine

Sub-categories	Codes	Categories
Positive emotional climate	PEC	Self-regulation
Previous experience	PE	Previous experience
Brainstorming	BRST	Problem-solving
Faulty computers	FC	Contextual factor
Pair programming Student engagement Meaning construction Discovery learning	PP SE MC DL	Constructivist teaching strategy
Scaffolding by teacher Scaffolding by student	SCT SCS	Scaffolding

The lesson commenced with an explanation of Scratch program blocks' terminologies such as stack, stack blocks, hats and script. The students were asked to pick the stacks and explore it on the Scratch environment (SE). Then students solved problems from their Scratch text for 30 minutes. They performed basic tasks and constructed meaning regarding the concept of repetition (MC). The students were paired (PP). The majority of them could not locate the costumes, so I helped them (SCT). Some students could not locate the 'ROTFL CAT' and I asked them to use another costume instead. I then observed that some students, who were not familiar with the computer system, found it difficult to find their way (PE). As students were solving the problem, some systems developed fault (FC). In the process of figuring out the problem, some students held discussions (BRST) and those, who achieved the desired result, applauded themselves (PEC) and moved on to the next activity. Students in groups B1 and B2 were seen discussing the problem (BRST). One student discovered that her group had made a mistake and that was why they were experiencing problems using the 'Pen up' (DL). Others, who were still struggling, received scaffolding from me (SCT) while Farai scaffolded others in group C (SCS).

6.2.3.2 Classroom observation: Lesson 14

The concept of expression in Scratch was taught in Lesson 14. Fourteen codes were identified from this teaching activity. These codes were: positive emotional climate (PEC), knowledge transfer (KT), critical thinking (CT), learning strength (LS), learning challenge (LC), group behaviour (GB), no group cooperation (NGRC), group cooperation (GRC), group behaviour (GB), knowledge of repetition (KOR), student contribution (SC), scaffolding by teacher (SCT), scaffolding by student (SCS), creative (CR), and faulty computer (FC). The corresponding categories are: self-regulation, self-efficacy, group cooperation, student behaviour, programming knowledge, social interaction, scaffolding, and creativity. Table 6.4 presents the analysis.

Table 6.4: Interim analysis of Lesson 14

Sub-categories	Codes	Categories
Positive emotional climate Knowledge transfer Critical thinking	PEC KT CT	Self- regulation
Learning strength Learning challenge	LS LC	Learning strength and challenge
No group cooperation Group cooperation	NGRC GRC	Group cooperation
Group Behaviour	GB	Student behaviour
Knowledge of repetition	KOR	Programming knowledge
Student contribution	SC	Social interaction
Scaffolding by teacher Scaffolding by student	SCT SCS	Scaffolding
Creative	CR	Creativity

The class activity commenced with solving problems involving order of precedence. In problem number 4 they did *not* follow the order of precedence correctly (Wassermann et al., 2011, p. 40). I guided them on how to solve the problem (SCT) and asked them to calculate the answer themselves. Only Benjamin could work out the answer and I asked him to explain it to the class (SCS). They practiced several other examples focusing on expressions and could, at the end, confidently solve calculations on expressions (LS). Project presentation followed with each group presenting its animations. In group A, students were displeased with the behaviour of certain group members who did not participate during the assignment (GB). There was cooperation and unity in the production of the project in group B (GRC). The students applauded themselves for the successful presentation (PEC). The group could use ideas learnt from previous classes to ensure creativity in their work (KT). Groups C and D's members all worked together and contributed to the success of the project (GRC). Their expression showed that they were happy about their achievements. Students commented on each group's work and offered suggestions, where needed (SC). As the presentations continued, some students talked loudly at the back of the class and did not participate during the presentation (SB). Figure 6.12 shows a group's animation.

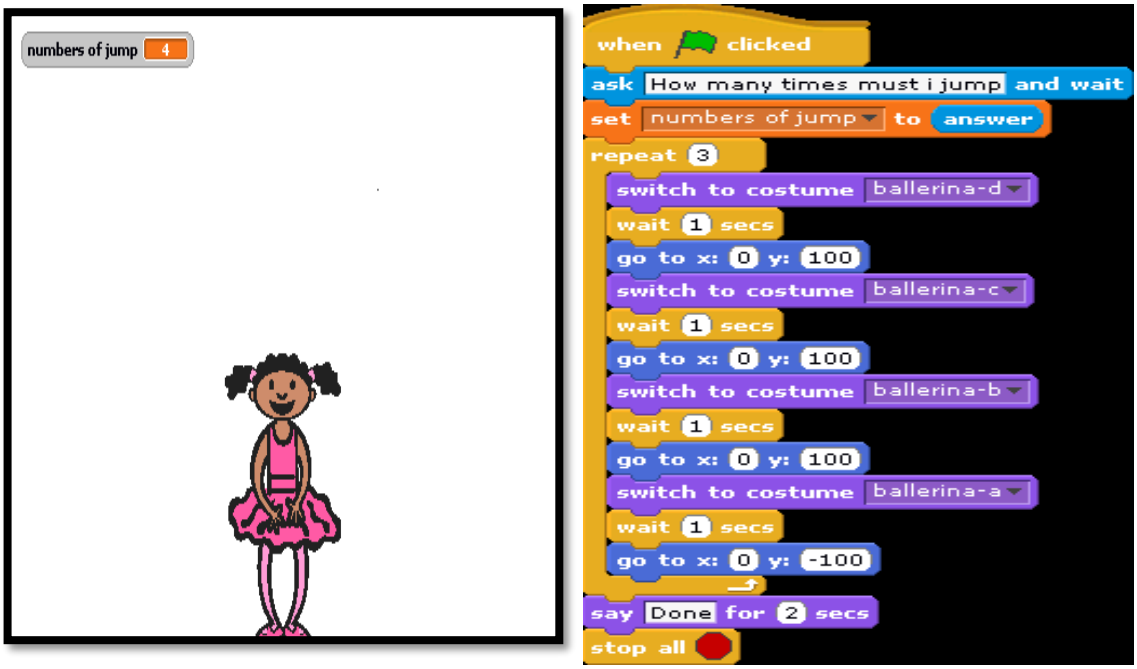


Figure 6.12: Sample of group D's animation and scripts on Ballerina

Two students did the assignment in group E (NGRC). One observed that the program animation was not jumping like the others (SC). According to the group members, the command they had given their ballerina was that after running the program, the ballerinas should hang up and not come down (LS). For group F, students observed that the sprite did not stop at the finishing line. The group corrected the work and the class applauded them (PEC). For group G, they could not determine why the horse had not run over the duck (LC).

Figure 6.13: Sample of group G's project and script

The students were creative (CR) and learnt repetition by creating their own animations (KOR). Through their questioning and contributions, I could deduce that my students' thinking had improved regarded the given problems (CT). Figure 6.13 provides a sample of group G's animation and script on repetition.

6.2.3.3 Classroom observation: Lesson 17

The concept of control structures was taught in QBASIC. From the teaching activity, eight codes were identified. The codes are: regulation of cognition (RCG), problem solving (PBS), student engagement (SE), disequilibrium (DSE), accommodation (AC), meaning construction (MC), debugging skill, student contribution (SC), social interaction (SI), learning challenge (LC) and program writing. The corresponding categories are: problem-solving, constructivist teaching strategy, learning strategy, debugging skills, social interaction, challenges face in programming and programming knowledge. Table 6.5 presents the analysis breakdown.

Table 6.5: Interim analysis of Lesson 17

Sub-categories	Codes	Categories
Regulation of cognition	RCG	Self-regulation
Problem solving	PBS	Problem solving
Student engagement	SE	Constructivist teaching strategy
Disequilibrium Accommodation Meaning construction	DSE AC MC	Learning strategy
Debugging skill	DS	Debugging skills
Student contribution Social interaction	SC SI	Social interaction
Learning challenge	LC	Learning strength and challenge
Program writing	PW	Programming knowledge

The first activity focused on correcting a program which had been written in the previous class. Each group were asked to solve the program on the whiteboard (SE). Mercy was chosen to represent group A, while students in groups B, E and G made corrections, where necessary (SE). Some students were confused (DSE) about the solution. After further explanations, these students grasped the problem-solving procedure which the group had used (AC). Joy represented groups C and D and other groups made corrections (SE). Students then had the opportunity to run their programs on the QBASIC environment using my laptop. Students reflected on the exercise and discussed what had been learnt (SI). They constructed meaning on the usage of IF and ENDIF statements and the connection between them (MC). They extensively discussed the QBASIC compiler's sensitivity to error which occurred because of human error (SC, MC). However, not all the students could

properly view the solution on the laptop, due to the student-laptop ratio (LC). In the second activity, students solved the following problem:

"A worker is paid a fixed rate per hour. His gross wage depends on the number of hours he puts in. Assuming the fixed rate 350 naira per hour, write a program to calculate the gross wage of 10 workers"

Through classroom interaction (SI), students solved the problem by first identifying the data which were then used to design an algorithm for the solution (PBS). As explained, students discovered that there was an error in the data they had identified (DS). They corrected this and jointly solved the problem (SE). They were exposed to a third activity in which only groups B and D could write the program with few errors identified (PW). Figure 6.14 presents a sample of the students' program on QBASIC.

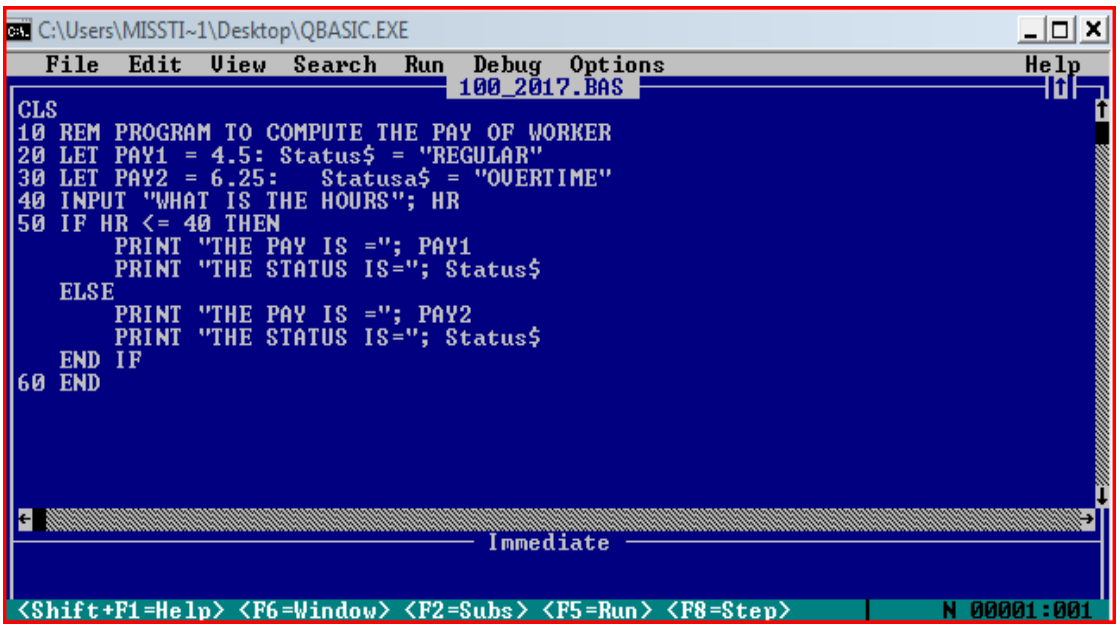


Figure 6.14: Samples of students' program

6.2.3.4 Classroom observation: Lesson 19

The concept of *looping* was taught in QBASIC. The topic had been previously taught. Lesson 19 focused on, amongst other things, solving the factorial of some numbers and N value. From the teaching activity, thirteen codes were identified. The codes are: student seeking help (SH), focused goal attainment (FGA), positive emotional climate (PEC), regulation of cognition (RGC), low self-efficacy (LC), brainstorming (BRST), student engagement (SE), meaning construction (MC), student behaviour (SB), debugging skill (DS), word processing skill (WPS), social interaction (SI), student contribution (SC),

scaffolding by teacher (SCT) and scaffolding by student (SCS). Table 6.6 presents the breakdown of the analysis.

Table 6.6: Interim analysis of Lesson 19

Sub-categories	Codes	Categories
Seeking help Focused goal attainment Positive emotional climate Regulation of cognition	SH FGA PEC RGC	Self- regulation
Learning challenge	LC	Learning strength and challenge
Brainstorming	BRST	Problem solving
Student engagement	SE	Constructivist teaching strategy
Meaning construction	MC	Learning strategy
Student behaviour	SB	Student behaviour
Debugging skill	DS	Programming knowledge
Word processing skill	WPS	Word processing skill
Scaffolding by teacher Scaffolding by student	SCT SCS	Scaffolding

The lesson commenced with solving more looping problems, jointly attempted by the teacher and students (SE). The programs were explained, after which students were afforded the opportunity to ask questions (RCG). Students were then exposed to a group activity. They were asked to: *Write a program to calculate 8! using INPUT or READ statements.* Students had to first solve the problem with pencil and paper whilst discussing possible solutions (BRST) in groups. A practical session then followed during which students could run their programs on my laptop (SE). Alfred, from group C, showed his intention to represent the group, but because he was not yet confident with QBASIC environment, he immediately withdrew stating, *"I cannot run the program because I am afraid*" (LC). Then, Faith from group A represented the groups. While keying in the data, I noticed that students could not identify a quotation mark meant to be used for a string constant (WPS). They called on Farai (SH) who explained it to them (SCS). Students identified some errors in their program after running the program the first time, but they were unable to debug (DS). Alfred suggested that they close the program and move to the second question (SB), but the other group members persisted (FGA). Farai assisted them (SCS) but they still identified another syntax error which they corrected (DS). They ran the program and were pleased with their effort (PEC). At this point, Faith noticed another error and they re-tested the program with different values to ensure its correctness (FGA).

Lucky represented the second group of students. They ran the program and debugged it (DS). As I checked their program, I noticed that there was a syntax error. I guided them on what to do and asked them to continue (SCT). Goodluck represented the third group. As they were running their program, they asked me to help (SH) them locate a quotation mark.

Farai guided them to locate it (SCS). They ran the program and experienced different syntax errors, such as using a 'NEXT without FOR'. Ogechi identified the error and the groups corrected the program (DS). Figure 6.15 shows the program written by students in groups D, E and F during the practical. At the end of the practical, students were asked to define their experiences. They understood the necessity to always input the correct figure, to use the correct command and to follow the rules of programming in QBASIC (MC).

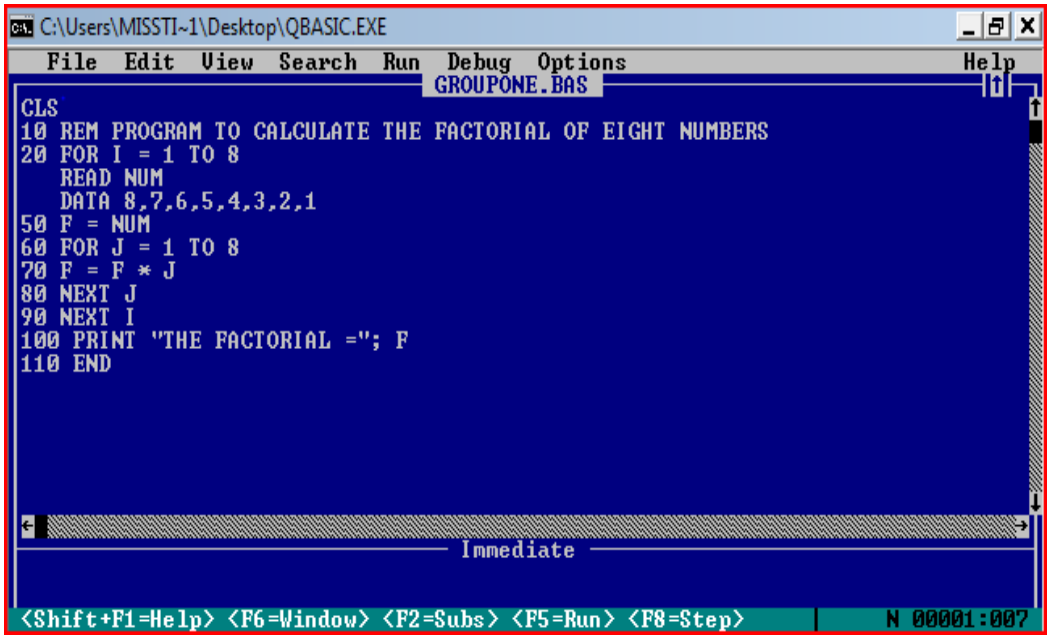


Figure 6.15: Screenshot of groups D, E and F program

6.2.4 Artefacts

Artefacts refer to different types of tests which were used in teaching during the observational period. Five different formative tests were administered during the semester: two tests on individual concepts, two interim tests and one final test. The results of each of these tests are discussed below.

6.2.4.1 Test of individual concept I

The test was administered on 1 February 2017 and it lasted 25 minutes. It comprised of recall and application type questions which centered on algorithm, pseudocode and flowcharts (Appendix A 10). Twenty-two students attempted the test. Some weeks after this test had been administered, a make-up test was given to those students who did not participate in the first test. The results of the make-up test are not included in this section.

Table 6.7 shows the revised Bloom's and SOLO taxonomies classification (*q.v*. Chapter 3) as well as the percentage correct answers for each question. Describing the classifications, questions 1 to 5, 8 and 9 required the participants to retrieve information from memory and were hence classified as *remember*. The questions are not program elements and were therefore not classified as atoms on the Block model. Since they had one relevant answer, they were classified as *unistructural.* Questions 6 and 7 required the participants to write pseudocode and draw a flowchart which involved putting parts together to form a whole (Thompson et al., 2008). Therefore, they were classified as *create* on the revised Bloom's taxonomy and *relational* because they required seeing the forest, and not trees. The students' understanding and application of knowledge gained, to write the program were tested. They were not classified on the Block model because this involved *writing* which fell beyond the scope of the model. However, question 10 was classified as *understand* because students had to construct meaning from the pseudocode to derive a flowchart. Also, classification on the Block model was *macro* and *program execution* because they needed to understand the data and control flow of the algorithm before they could change the representation to a flowchart.

The result shows that the students found questions 2, 3 and 4 easy. The reason for this could be that the questions were of the *recall* variety. Questions 6, 7 and 10, where they were expected to *apply* what they had learnt in writing a pseudocode and drawing flowcharts, was quite challenging for them. Students' inability to fully grasp the concept of flowchart and pseudocode was noted in the classroom before they attempted the test. However, questions 1, 5, 8 and 9 are QBASIC definitions and algorithm type questions which yielded a low percentage of correct answers. There was no correct identification of an abstract and concrete algorithm. Other points, noted from the scores, where students still needed improvement included: not mixing algorithms, flowcharts and pseudocode together, the need to use the PRINT statement in the flowchart representation and non-splitting of flowchart. Table 6.7 presents an analysis of the test scores.

Table 6.7: Analysis of test of individual concepts I

Question	Revised Blooms	Block model		SOLO	% correct Answers
1	Remember	-	-	Unistructural	25%
2	Remember	-	-	Unistructural	77%
3	Remember	-	-	Unistructural	87%
4	Remember	-	-	Unistructural	100%
5	Remember	-	-	Unistructural	30%
6	Create	-	-	Relational	31%
7	Create	-	-	Relational	19%
8	Remember	-	-	Unistructural	28%
9	Remember	-	-	Unistructural	20%
10	Understand	Macro. S	Program. E	Relational	21%

All students who scored below 10 marks struggled with solving programming problems on flowcharts and pseudocode. Students who scored above 10 marks could solve problems on flowchart and pseudocode, but their solution showed that they needed more clarification on their structuring. Overall, only 26.3% performed better while 73.7% performed low. It is interesting to note that all new students who attempted the questions scored very low. This might have influenced the percentage of students who performed low. The students' performance and understanding of constructs in the test may, or may not, have affected *the way in which* they use the constructs in solving programming problems. However, these are basic facts which the participants were supposed to have mastered at the time of testing. Figure 6.16 gives a summary of students' test scores. Out of 20 obtainable marks, result shows that six participants scored *above* average while one participant scored average. The remaining fifteen participants scored *below* average.

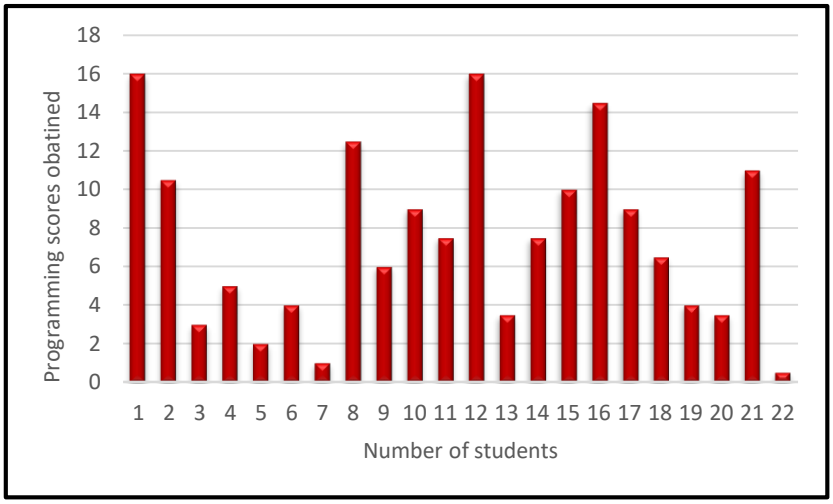


Figure 6.16: Student scores on test of individual concept I

6.2.4.2 Test of individual concept II

The test of individual concept II (Appendix A 11) comprised eight multiple choice and four subjective questions and lasted 15 minutes. It was administered to the students on 3 March 2017 and 26 students attempted the test.

Questions 1 to 7 and 9 were classified as *remember* on the revised Bloom's because they required that the students recall facts from memory. Question 8 required that the participants first grasp the essence of the line of code, before they could display the value. Hence it was classified as *understand.* Regarding questions 1 to 9, only question 4, 6 and 9 were *not* code related. Other questions were classified as *atoms* at the *text surface* level, because they were either QBASIC program elements or question about the program elements. Questions 10 to 12 were classified as *apply* because the participants were expected to use an already known strategy to arrive at the correct answer. On the Block model, they were classified as *atoms* because they are program elements and *program execution* because the participants need to understand the flow of data in each question. All the questions were tagged *unistructural* on the SOLO classification because they supplied one idea for each question.

Table 6.8: Analysis of test of individual concept II

Question	Revised Blooms	Block model		SOLO	% correct Answers
1	Remember	Atoms	Text S	Unistructural	60%
2	Remember	Atoms	Text S	Unistructural	56%
3	Remember	Atoms	Text S	Unistructural	28%
4	Remember	-	-	Unistructural	80%
5	Remember	Atoms	Text S	Unistructural	48%
6	Remember	-	-	Unistructural	16%
7	Remember	Atoms	Text S	Unistructural	36%
8	Understand	Atoms	Text S	Unistructural	20%
9	Remember	-	-	Unistructural	36%
10	Apply	Atoms	Program. E	Unistructural	52%
11	Apply	Atoms	Program. E	Unistructural	44%
12	Apply	Atoms	Program. E	Unistructural	12%

Understanding gained from the interpretation of Table 6.8 shows that the questions students found easy were 1, 2, 4 and 10 with 60%, 56%, 80% and 52% percentage scores respectively. This shows that they could identify *both* string and numeric constants. They understood *debugging* and were able to assign a value to an expression using the LET statement. However, students found questions 3, 5-9, 11 and 12 difficult to answer. They had not yet internalised the functions of QBASIC commands, such as INPUT, READ, PRINT and F5. Even though they could identify string constants, they were unable to appropriately identify the string variable. Also, they were unable to solve application questions (11 and

12). A reason for this may be that students had not previously been exposed to the application problems included in the questions.

Table 6.9 shows the mapping of both the SOLO and the Block model classifications for the second test of individual concepts. The table shows that the participants' level of comprehension was considered at *atom*. The knowledge dimension moved from *text surface* to *program execution*. The SOLO classification of the questions answered was unistructural at the text surface with 41.3% correct answers, and program execution with 36% correct answers to all questions. This shows that students performed better on the atoms than the program execution level. The SOLO classification supports the work of Whalley and Kasto (2013) and Schulte (2008). Although their research focused on Java programming which does not share the same paradigm as QBASIC. Research work on SOLO and Block model mapping on the QBASIC has not been found.

Table 6.9: Mapping of SOLO and Block model on test of individual concept II

Macrostructure			
Relations			
Blocks			
Atoms	Unistructural (41.3%)	Unistructural (36%)	
	Text surface	**Program execution**	**Function**

Figure 6.17 depicts a summary of students' test performances. The test counted out of 12 marks. Five students scored *above* average and six students scored *average*. The remaining fifteen students scored *below* average.

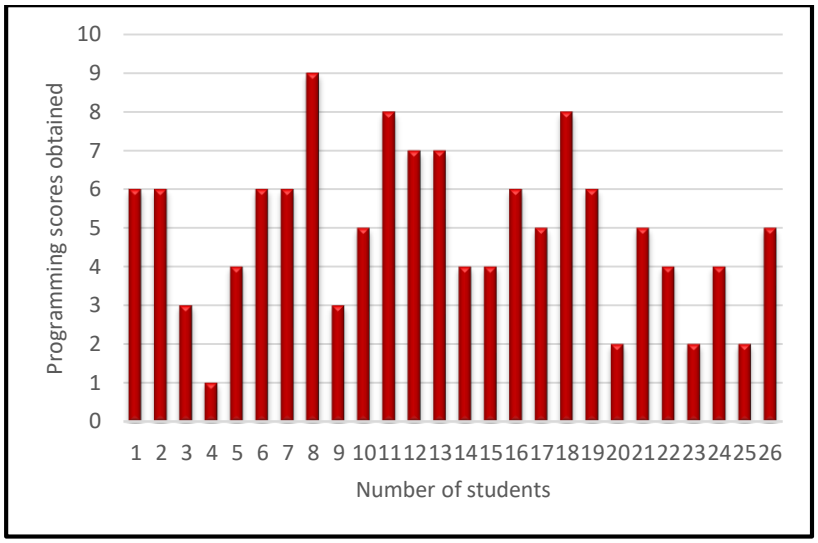


Figure 6.17: Students' scores on test of individual concept II

6.2.4.3 Interim test I

The interim test I (Appendix A 12) comprised of six different questions and lasted 45 minutes. Table 6.10 presents an analysis of the cognitive and comprehension dimensions of the test questions on the revised Bloom's and the Block model, respectively. Students responses on each question were categorised on the SOLO taxonomy.

Questions 1 and 6b had the same classification. In question 1, a few bugs were introduced into the code and students were expected to trace, identify syntactic errors as single program elements (variables) and rewrite the code. Question 6, on the other hand, required students to recall facts from memory. Therefore, both questions were classified on the Block model as *atom* at the *text surface* level and *remember* on the revised Bloom's taxonomy. Students' responses on this question were single answers which required their partial understanding and therefore classified as *unistructural.* The percentage correct answers on the question (41%) was quite low.

Table 6.10: Analysis of interim test I questions

Question	Type	Revised Bloom's	Block model		SOLO	% correct Answers
1	Tracing	Remember	Atom	Text surface	Unistructural	41%
2	Code purpose	Understand	Macro structure	Function	Relational	68%
3	Writing	Apply	-	-	Multi-structural	30%
4a	Change in representation	Understand	Atom	Text surface	Unistructural	16%
4b	Change in representation	Apply	Atom	Program. E	Unistructural	52%
4c	Tracing	Apply	Atom	Program. E	Unistructural	38%
5	Parson's puzzle	Apply	Block	Text surface	Multi-structural	59%
6a	Program writing	Create	-	-	Relational	32%
6b	Recall type	Remember	Atom	Text surface	Unistructural	31%

Question 2 required the students to explain, in plain English, what the program does. Therefore, it was classified as *understand* because it required students to form a mental representation of the code and describe it in their own words. As per the Block model, the level of comprehension was classified as *macrostructure* while the knowledge dimension as *function,* because students needed to understand the program goal. In terms of the SOLO, they were expected to understand the relationship between each line of code and it was therefore classified as *relational*. The percentage correct answers on this question was 68%. Question 3 involved writing a QBASIC code for a set a pseudocode. This question involved writing and so it was not classified on the Block model. The classification on the revised Bloom's was *apply* because students were expected to use the knowledge gained

to produce a QBASIC code. Student responses were classified as *multi-structural* because the language constructs were more than one and the students only needed to understand the structure of the pseudocode. Percentage correct answers on this question were 30% which was very low. Compared to the questions posed on writing in the individual concept test 1 (see section 6.2.4.1), the students writing skills were still developing.

Question 4 was divided into three sections (a, b and c), each with its own classification. Question 4a required students to represent the code using a single QBASIC statement and therefore it was classified at the lower level of the Block model as *atom* at the *text surface*. It was categorised as *understand* on the revised Bloom's, because the students needed to deduce the meaning of each statement before they could represent it. A single response was given which falls at the *unistructural* level on the SOLO. The percentage correct answers for this question was very low (16%). I think the students did not interpret the question correctly or they were, in fact, unfamiliar with this type of question. Question 4b, on the other hand, involved using a known process of writing QBASIC expressions to solve the given problem and was thus classified as *apply*. Because language elements required students to determine the operation of each expression in QBASIC, the comprehension and knowledge dimensions were classified as *atoms* and *program execution*. Students gave a single response to each statement, therefore classifying it as unistructural. The percentage correct answers were 52%, which denotes an average performance. Question 4c focused on the application of the order of precedence to solve a familiar problem, using unfamiliar data (Thompson et al., 2008). This problem, in term of the cognitive process, was *apply* and it was classified as *unistructural* according to the SOLO taxonomy because it required the students to produce a single answer for each problem. As per the Block model, the students needed to understand the control and data flow of the code before they could operate at the *program execution* level. To trace the question, they had to operate at the lower *atom* level.

Question 5 contained a Parson's puzzle which required the students to correctly arrange the code to fit its purpose. They were provided with the purpose of the code which was to calculate the roots of an equation. This question was classified as *apply* because they had to use a known process to place the code in the right order. Students operated at the Block comprehension dimension on the *atom level* because they dealt with the sequencing of codes which must be semantically arranged to build the unit. Since students made a connection between the parts of the code to form a coherent whole, it was classified as *relational*. Although they were given the purpose of the code, they still needed to arrange the codes to achieve the goal. Correct answers recorded for this problem was 59%. Question 6a involved writing a program using the IF…THEN statement. The students were

expected to utilise their knowledge of the notional machine, problem solving, syntax and semantics to *create* a set of codes. It required that they build a connection between the codes to arrive at the correct answer, therefore it is *relational*.

The bar chart in Figure 6.18 represents each student's overall score on the interim test I. Out of thirty obtainable marks, ten students scored *above average* while the remaining twenty students scored *below average*. One student scored an average mark of 15. The questions students found difficult were *change in representation* and *writing type* questions which comprised question 3, 4a, 4c and 6a. Other questions with low percentage scores were tracing (41% and 38%). This result indicates that students might find it problematic to think at the relational level (Philpott et al., 2007). Recall type questions also scored at a lower level (31%).

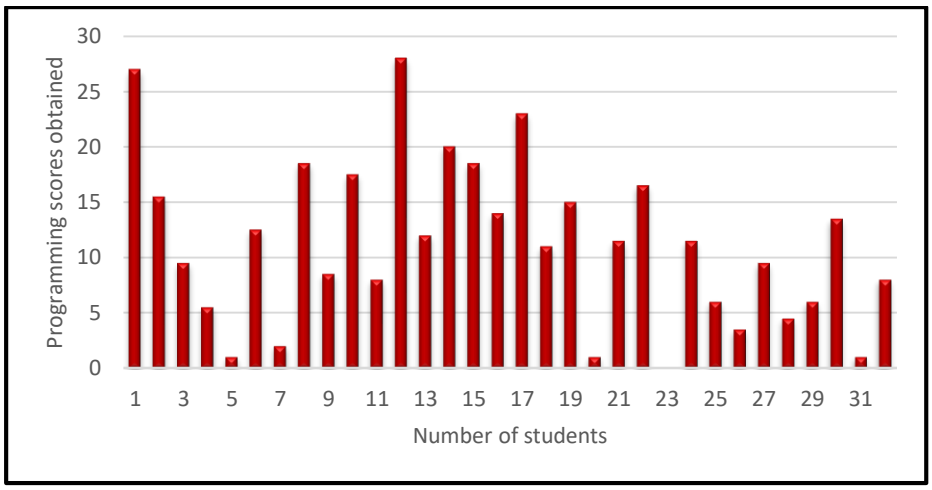


Figure 6.18: Students' scores on interim test I

While reflecting on these scores, I discovered that:

- Students do not know when to use a decision statement in a program.
- Some students lack simple calculation skills.
- Salina cannot differentiate between a flowchart and pseudocode.
- They did not decompose a problem i.e. they did not apply algorithmic techniques in writing a program.
- Some students had not yet mastered the use of QBASIC syntax in program writing. It could be that they did not practice in the QBASIC environment. This was mainly because the QBASIC classes were taught from 16:00 to 18:00 when the power was cut. An effort was made to engage students in QBASIC practical from 12:00 to 14:00, when there was still natural light.

Table 6.11: Mapping of the SOLO and Block model for interim test I

Macrostructure			Relational (68%)
Relations			
Blocks	Multi-structural (59%)	Cross-referencing	
Atoms	Unistructural (29.3%)	Unistructural (45%)	
	Text surface	**Program execution**	**Function**

Table 6.11 shows the mapping of both the SOLO and the Block model classifications for interim test I. It was an improvement over the test of individual concept II, because the table now reflects questions at the blocks and macrostructure. From the table, overall percentage scores for the three questions at the atom/text surface was 29.3%. For the atom/program execution, the percentage score for the two questions was 45%. At the blocks/text surface, the percentage score of the one question was 59% while macrostructure function percentage score for the one question was 68%. Understanding gained from this analysis revealed that the participants had trouble regarding the text surface, but an improvement was noticed as they moved towards the program execution of the knowledge dimensions. As they progress towards the Blocks, the percentage score at the text surface increased. However, at the macrostructure/function, the achievement was higher than atoms and blocks of the knowledge dimension. This result is surprising because it was earlier stated that there is always a cognitive difficulty as they move up the block (Whalley and Kasto, 2013).

6.2.4.3 Interim test II

Interim test II (Appendix A 13) involved a program writing question which was administered on 5 April 2017. Students were asked to write few program codes and run it on the QBASIC environment for 40 minutes. Twenty-five students attempted the test. This question required that students apply their knowledge of the notional machine, problem solving, syntax and semantics to construct an algorithm and produce a code. Therefore, it was classified as *create.* On the SOLO taxonomy, student responses are classified as *relational* because it required building a connection between the codes to achieve the required result. Percentage correct answers on this task was 50.4% (see Table 6.12). Students' writing skills had improved compared to performances in the previous tests.

Table 6.12: Analysis of interim test II

Question	Type	Revised Blooms	SOLO	% correct Answers
1-4	Writing	Create	Relational	50.4%

Figure 6.19 shows each student's overall score on interim test II. Seven students scored 10 marks, nine students scored *above average* while nine students score *below average* out of ten obtainable marks. Students' writing skills had improved.

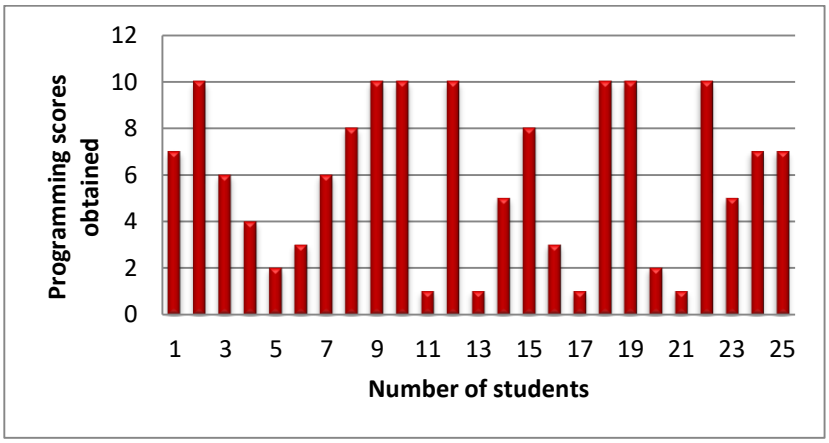


Figure 6.19: Students' scores on interim test II

6.2.4.3 Final test

The final test (Appendix A 14) was administered using pen and paper on 7 April 2017 with 22 students present. The test, which lasted one hour, covered all topics learnt over the semester. Table 6.13 presents a breakdown of the test analysis. The following account presents the analysis and discussion of the seven program comprehension questions. Questions 1, 2 and 4a (i) had the same classification on the table. Question 1 comprised of programming basics which required students to recognise their function in QBASIC (Thompson et al., 2008). Question 2 involved tracing the code to identify the syntax errors while the students were expected to trace the code to identify the variables in the code. Because all these involve language elements which were retrieved from long-term memory, they were classified as *atoms* at the *text surface* level and *remember* on the revised Bloom's. They are *unistructural* on the SOLO because the students only gave one relevant idea regarding the code sections.

The next two questions (3a and 7b) are analysed and discussed together because of the commonalities in their classification (*q.v*. Table 6.13). Question 3a is a *skeleton type* question with three lines of array missing. Students were expected to complete part of missing lines of code in the array. They needed to employ a certain process to declare an array learnt in the programming classroom to provide the missing lines, therefore it was

classified as *apply* on the revised Bloom's. In question 7b, bugs were introduced into the loops and students were expected to hand execute the loop and correct the bugs. To locate these errors and missing lines (3a), they had to *understand* the algorithm of the codes, even though the goal had been given. Comprehension is therefore at the *macrostructure*. Because the function of the code had been given, they had to manually trace the control and data flow of the programs to obtain the required result, therefore it was classified as *program execution*. Both questions are *relational* on the SOLO classification since the students needed to integrate the constituent parts of the program to determine its structure so that the goal could be achieved.

The next analysis focuses on questions 3b and 5a (ii), which have similar classifications. Both are 'code tracing' questions which required students to hand execute an expression (3b) and produce a QBASIC form of an arithmetic expression using a known strategy. Therefore, they were regarded as *apply* on the revised Bloom's taxonomy. Since both questions involved performing certain operations that required understanding the data flow, but not necessarily the function of the expressions, their comprehension level is *atoms* while the knowledge dimension is *program execution*. The students' responses, as per SOLO, were unistructural because they provided answers to a single program element.
Question 4a (ii) and 4b are *code purpose* and *change in representation* types, respectively. Both required that students conceived what the code was doing before they obtained the result. They were therefore classified as *understand* on the revised Bloom's taxonomy. Their classification on the comprehension level of the Block model was *macrostructure*. For question 4a (ii), the students needed to form a representation of the program before they could explain its purpose, therefore the knowledge dimension was classified as *functions while* question 4b was classified as *program* execution since they first needed to mentally form, or design, the algorithm before they could prepare the flowchart. In terms of the SOLO, they were expected to understand the relationship between each line of code, and therefore both questions were classified as *relational*. The percentage correct answers for the questions were 59% and 36% respectively. Question 5b shares the same classification as 4b, but to classify it as *macrostructure* the students needed to first understand the purpose of the code, identify each variable type with the associated string and understand how the data and program flow would be controlled to achieve the required result. Therefore, the students needed to first *understand* the program itself before they *applied* the known process.

In questions 6a and 6b, the students were presented the same question, but with different tasks. *Code intent*, where students were required to give a description of what the code was

doing, was the task given in 6a, and therefore classified as *functions* on the knowledge dimension of the Block model. However, the code involves two different sub-goals which students had to relate together to obtain a sense of the code purpose, hence *relations* on the comprehension level. Question 6b, on the other hand, comprised *syntax errors* which required the students to trace and debug the line of code with errors. They needed to know the purpose as this would assist them in identifying the errors. Therefore, the question was classified as *understand*. As they needed to identify the data and control flow of the program, it was classified as *atoms* at the *program execution* knowledge dimension of the Block model. They are regarded as *unistructural,* as per SOLO, because the students only gave one relevant idea about the code sections. Percentage correct answers were 77.3% for 6a and 37.9% for 6b.

Question 6c is a *change in representation* type which required the students to rewrite the code using their own method while judging *why* their own method was better. To analyse this question, the students moved from the lower to the higher cognitive level on the revised Bloom's taxonomy. They first needed to *understand* the purpose of the program since the goal had not been given. Then they *applied* a known process to the unfamiliar problem, by creating another representation of the same program. A further activity involved judging between the old and new program codes to explain the reason for choosing their method and why it was better. Therefore, 6c was classified as *evaluate*. The program execution is associated with *data and control flow* which students needed to be aware of before rewriting their own program, hence it was classified as *program execution*. The program comprised two branching IF…THEN and IF…GO TO statements with supporting data. Although they are single line statements, they could also be a block. Hence it was classified as a *block*. On the SOLO taxonomy, it was classified as *relational* because the students needed to have a full understanding of the code, i.e. they needed to see the forest and not just the trees. Percentage correct answers on this question was 14.4% which was very low.

Question 7a was a *change in representation* type which required that students apply their knowledge of the notional machine, problem solving, syntax and semantics to produce a QBASIC program for the pseudocode. Therefore, it was classified as *create.* On the SOLO taxonomy, student responses are classified as *relational* because it required building a connection between the codes to achieve the required result. Percentage correct answers for this task was 34.6%.

Table 6.13: Analysis of final test questions

Question	Type	Revised Bloom's	Block model		SOLO	% Correct answers
1	Basics	Remember	Atoms	Text. S	Unistructural	56.4%
2	Syntactic Errors	Remember	Atoms	Text. S	Unistructural	73%
3a	Skeleton Code	Apply	Macro. S	Program. E	Relational	62%
3b	Code tracing	Apply	Atoms	Program. E	Unistructural	26%
4a(i)	Code tracing	Remember	Atoms	Text. S	Unistructural	36%
4a(ii)	Code purpose	Understand	Macro. S	Functions	Relational	59%
4b	Change in representation	Understand	Macro. S	Program. E	Relational	36%
5a(i)	Code tracing	Analyze	Atoms	Text. S	Unistructural	38%
5a(ii)	Code tracing	Apply	Atoms	Program. E	Unistructural	30%
5b	Change in representation	Apply	Macro	Functions	Relational	78%
6a	Code intent	Understand	Relations	Functions	Relational	77%
6b	Syntactic error	Understand	Atoms	Program execution	Unistructural	38%
6c	Change in representation	Evaluate	Blocks	Program. E	Relational	14%
7a	Change in representation	Create	-	-	Relational	26%
7b	Syntactic errors	Understand	Macro. S	Program. E	Relational	35%

Table 6.14 shows the mapping of both the SOLO and the Block model classifications of the final test. From this table one can see that the participants experienced difficulties with the text surface, especially questions 4a(i) and 5a(i), though the overall percentage correct answers were average (51%). Moving towards program execution, the percentage correct answers were low (31%) with difficulty experienced in questions 3b, 5a(ii) and 6b. They still had difficulty as they moved up the blocks at the program execution (question 6c) which had the lowest percentage (14%). At the relations and function level, an increased percentage correct answers (77%) was found. On one hand, at the macrostructure, there seems to be a low percentage correct answers (34%) at the program execution level, on the other hand, a high percentage was found at the functions (69%). Generally, it was noted that students found questions at the *function* level easier than both *atoms* and *program execution* of the comprehension dimension. In addition, further careful design needed to cover other blocks to foster cross-referencing.

Table 6.14: Mapping of SOLO and Block model classifications

Macrostructure		Relational (34%)	Relational (69%)
Relations			Relational (77%)
Blocks		Relational (14%)	Cross-referencing
Atoms	Unistructural (51%)	Unistructural (31%)	
	Text surface	**Program execution**	**Function**

Figure 6.20 graphically represents students' scores on the final test. Out of 60 marks, ten students scored *above average*, while the remaining twelve students scored *below average*.

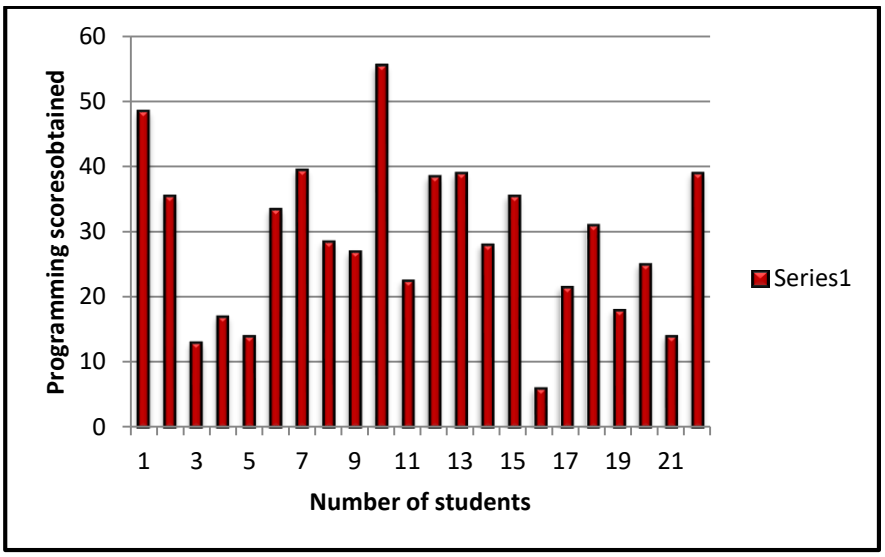


Figure 6.20: Students' scores on final test

6.2.5 Interview with participants

This section presents results of the interviews conducted with eight participants. As in Cycle 1, participants were selected based on their overall performance in programming. As a result, high, medium and low achievers were selected. The interview was divided into two sections. The first part, the retrospective interview, was first conducted and then followed by a semi-structured interview. This discussion follows the same order.

6.2.5.1 Retrospective think aloud interview

The purpose of this section aligns with the processes used in the first cycle, which was to understand participants' mental representations of small sets of programming codes. However, this process was done differently in the second cycle. A video recording of students' programming activities was made and the recorded video was used as a pointer to the retrospective interview (Whalley and Katso, 2014). The interview was based on the

sets of questions (Appendix A 5) used in the first cycle. Here, the questions are reduced to two (codes 1 and 3) and mapped to the Block model. The problems given to the participants were varied, but the interviews focused on central ideas which included: variables in a code, explanation of the program codes, function of a statement and the goal of a program as well as re-writing the program in another form. The questions were further mapped to the Block model, Bloom's and SOLO taxonomies. These taxonomies were discussed earlier (*q.v.* Chapter 2). An overview of the classification of questions is presented in Table 6.15.

Table 6.15: Classification of retrospective questions

Question	Type	Revised Bloom's	Block model		SOLO
			Level	Comprehension dimension	
1	Tracing	Remember	Atoms	Text surface	Unistructural
2	Code intent	Understand	Macrostructure	Text surface	Relational
3	Explaining	Understanding	Relations	Program execution	Relational
4.	Writing	Creating	-	-	Multi-structural
5.	Code intent	Understand	Atoms	Function	Unistructural

The guidelines used to classify these questions were described in Cycle 1. The essence of this classification was to determine the level at which the participants operated during the think-aloud interview. What follows is a discussion of students' mental representation on each program comprehension questions along the model.

Question 1: Code Tracing

In question 1, the participants were asked to identify the variables in the set of code. Tracing questions, as noted by Whalley and Kasto (2013), involves a line by line tracking of data. The question required students to retrieve knowledge from memory and was therefore classified as *remember.* The students' answers comprised one or two constructs, and therefore its SOLO classification is *unistructural*. As per the Block model, the question was classified as *atoms* at the *text surface* level because this construct helped them to understand only the rules and not the flow or the purpose of the program. They could list the "NAME$", "SALES", "GAIN", "MATHSCORE", "ENGSCORE", "SCORE", "AVESCORE", "DOUBLESCORE" and "TOTSCORE" as variables in the code. They were all able to differentiate between numeric and string variables in the program given.

Question 2: Code intent

The question required students to give a mental representation of the code. Since they were to describe the overall goal of the program, the classification on the Block model fell to the

text surface level while the comprehension fell at *macrostructure*. Code intent questions were formally classified by Thompson et al. (2008) as *understand* at the revised Bloom's and *relational* at the SOLO classification (Clear et al., 2008b). The participants could give the intent of the code based on their representation. One participant said *"to calculate gains of the sales and gain of worker"* for code 1.

Question 3: Explanation of a line of code

This question required the student to express their explaining skills, whereby they first needed to understand the purpose of each program code in context, therefore it was classified as *understand*. This question looks like a *code intent* question, but it differs because students are to study the program by relating each segment, or sub-goals, to the main goal. It was classified to be at the *relations* level with the comprehension dimension at the *functions.* It is *relational* because students must see the forest to be able to offer a description of all the program segments. All the participants, except one, saw this question as a code intent question. For example, she said, "*the program allows us to know the gain of a salesman by introducing counter and the number of counts to work on*...*the program try to identify worker who has lesser sales and income, who (which) has middle sales and income, and who has the highest sales and income*". An example of the participant's representation for the program codes is "*the INPUT asks for the company's name…the computer expects the user to INPUT the expected value".*

Question 4: Writing

This question required the participants to re-write the program codes using the FOR…NEXT statement. It was classified as *creating* at the revised Bloom's because it required the students to produce the program in another way by using knowledge gained in the programming classroom to construct their own program codes. It was difficult to classify the code in the Block model because the model was designed to assess only program reading and not writing (Schulte, 2008). It was classified at the SOLO as multi-structural because it entailed that the students construct understanding of each line of code before writing it. Program codes written by participants differed based on the understanding of the FOR…NEXT statement. A sample of the written program will be explained in section 5.3.2.

Question 5: Code intent

Participants gave a mental representation of the purpose of the code. Since they were to explain the function of a single statement in context, the classification as per the Block model fell at the *text surface* level while comprehension fell at the *function*. Code intent questions have been formally classified by Thompson et al. (2008) as *understand* at the revised

Bloom's. At the SOLO classification, it is *unistructural* because a single statement explanation was required. The participants could explain the function of the statements in the program. For example, "*if counter is counting student and counter count 1 student, the function of GOTO is to direct the program back to counter".*

What they were thinking when solving the problem

When asked what they were thinking when solving the problem, two participants revealed that they were confused with the use of GOSUB. Some of them said they were thinking of how to introduce the FOR...NEXT in their own program, even though they had done it in the class before. Others said that they were thinking of *where* to place the FOR and the NEXT to obtain the desired result. They were unable to do that, except for one participant who said he could obtain the correct answer. Some of the participants said that while they were writing the program, they were worried that there were still syntax errors, even after debugging the program. Inability to correctly run the program made them feel worried and confused.

6.2.5.2 Semi-structured interview

This section describes the responses elicited from the students through the interview protocol (Appendix A 4). The interviews were transcribed and coded. In all, 63 codes which are both pre-existing and data-driven emerged from the transcripts (*q.v*. section 4.4.2). The codes were further patterned to form 21 categories. Table 6.16 presents the breakdown of the analysis. It features each code, its acronym and the categories. The codes that represent each sentence, or phrase, in the following discussion are highlighted in sky-blue.

Table 6.16: Analysis of interview transcripts

Sub-categories	Codes	Categories
Knowledge of self Knowledge of group characteristics Quadrants grouping aided learning B-aided decision making Whole brain boosts thinking	KS KGC QG DM WBT	Whole brain influences learning
Project Assignment Group learning	PJ AS GRPL	Constructivist teaching
Expected teacher qualities	ETQ	Expected teacher qualities
Prefer pair programming Pair cooperation Ideas not respected in pair Perspectives about pair programming	PPP PC INRP PAPP	Pair programming
Quadrant grouping aided learning Expression skills	QG ESP	Group encouraged learning
No group cooperation Group members attitude Student attitude Group Behaviour Group struggle “We adapted”	NGRC GMA SA GRB GS WA	Group members’ negative behaviour
Group negative impact	GNI	Negative impact of grouping
Solution regarding practical Provision of computers Provision of electricity Whiteboard New programming laboratory Possession of a PC by student Improved Lab environment	SRP PRC PEL WB NPL PPS ILE	Solution to contextual problem
No enough practical	NEP	Contextual problem
Focused goal attainment Seeking help	FGA SH	Self-regulation
Group interaction Social interaction Expression skill Sharing of ideas	GRPI SI EXPS SH	Social interaction
Student mood	SM	Student mood
Scratch is Easy Fun Enjoyable QBASIC easy with Scratch Scratch application to QBASIC	SIE F ENJ QBES SAQB	Perspectives about Scratch
Knowledge of variables Previous experience Running of programs	KOV PE ROP	Programming knowledge
Program writing Scratch helped to learn order precedence Program arrangement	PW SHOP PAR	Scratch builds program writing skills
Scratch encourage debugging in QB	SED	Scratch encourage debugging
Scratch boosts interests QBASIC interesting with Scratch	SBI QBIS	Scratch boosts interest
Scratch inclusion CS curriculum adjustment “Programming as a major course’	SINCL CADJ PMJ	Scratch inclusion to the curriculum
Scratch challenges	SC	Scratch challenges

"No serious thinking" Scratch aided thinking in QB Programming involves thinking Thinking ahead Thinking Relax and Think	NST STQB PIT TA T RT	Programming aids critical thinking
Programming impact Love to be a programmer	PIM LPR	Programming impact

How teaching intervention helped learning

When the participants were asked *how the teaching intervention helped learning*, they noted that the whole brain grouping aided the development of their less dominant profiles, especially in the A and B quadrants (QG). In the group it also fostered free expression, the making of new friends and learning through social interaction (SI). They also noted that they felt privileged to know their preferred way of learning (KS) and that of the other group members (KGC). The intervention also enabled them to learn by engaging in thinking (WBT). However, there was lack of cooperation and frequent arguments which always resulted in confusion within groups (NGRC). They said the single quadrant grouping aided understanding in programming since they belonged to the same quadrants (WBG). They stressed that the whole brain four quadrant grouping caused different thinking, but they later adapted to the method (WBG). One participant said the whole brain grouping made him to relent in programming because of the group's behaviour (GNI). Another participant noted that the effectiveness of the whole brain grouping depends on the seriousness of the group members (GNI). They emphasised that pair programming aided learning (PPAP), and as such they preferred it to programming in groups (PPP) as it fosters: concentration and cooperation, sharing of ideas, easy and interesting (PPAP). On the contrary, one participant said ideas were not respected in pair programming (PPAP).

Perspectives of Scratch programming

The participants noted that Scratch was easy (SIE), fun (F) and enjoyable because it involves graphics (ENJ). Scratch also helped them in program writing (PW), especially in the orderly arrangement of programs (PAR) and the order of precedence in QBASIC (SHOP). Explaining further, the knowledge of block arrangement in Scratch aided the orderly arrangement of codes in QBASIC (PAR). They all noted that Scratch helped them to understand programming better and that without the Scratch, QBASIC programming would have been very difficult (SAQB). They also remarked that Scratch programming helped them to check programs for error as well as debug their written programs on QBASIC (SED). They supported this by stating that Scratch boosted their interest in programming because it was interesting (SBI). Participants said the grouping enabled them to learn Scratch programming (SBI). On the contrary, others said that Scratch was not good for

group works (GNI). Participants explained their frustrations relating to location and importing of costumes from the computer (SC). Also, they were frustrated with the series of projects and assignments given during the course (SC). Though they enjoyed Scratch (SBI), their friends were not much interested in Scratch because it was not the major course registered for and needed for graduation (SC).

Overall experience in procedural programming

When asked about their *overall experience in programming*, the participants noted that programming was interesting because of Scratch (QBIS), and the grouping of students aided learning (QG). They also noted that learning some concepts first in Scratch and then in QBASIC solidified their understanding of: order of precedence, looping, IF statement and program arrangement (QBES). One participant added that he loved to help his peers but that he felt bad whenever he was unable to solve their problems (PIM). They enjoyed working in groups during learning because it enabled social interaction and knowledge of their abilities (SI, KS). However, one participant said that not all group members were prepared to work and that some were easily distracted (GRB). Therefore, the participant noted that Scratch had a negative effect on QBASIC programming because of the grouping of students in Scratch class (GNI). This, in the long run, affected his understanding of pseudocode. They also frowned on the practical aspect of programming. They noted that there were not enough practical sessions during the semester (NEP).

Participants' feeling about programming

When asked how they *felt about programming*, three participants noted that programming involves thinking (PIT), thinking ahead (TA) and that a programmer needs to relax and think carefully before solving any problem. Two of the participants said they felt positive about programming, so much so that they now have passion for programming and that they might even consider becoming programmers (LPR).

Relationship between Scratch and QBASIC

The participants noted the relationships between Scratch and QBASIC as: data type representation, programming arrangement, running of program, the concept of repetition and order of precedence (SAQB). They also noted that both involve the following of procedures.

Recommendation towards the design of a new learning framework

Participants stressed that the teaching of Scratch programming should be emphasised and included in the curriculum, from primary to tertiary levels (SINCL), to boost students' interest

in programming. One participant said the computer science curriculum should be adjusted. Basic levels of programming should be taught in primary and secondary schools, with extensive practical on Scratch, while more advanced levels are to be taught at tertiary institutions (CAD). He also noted that if programming had been taught in this way, it would be easier and more interesting. The Government should also consider making 'programming a major course' to be studied at tertiary institutions and not to be paired with another course (PMJ). This will reduce overcrowding of the curriculum while students can then focus on a few courses. They said the college should ensure constant electricity so that students could always practice in the computer laboratory (PEL). They also said that enough computers should be provided in the laboratory so that each student is assigned a computer system (PRC). One participant noted that the college should provide a 'programming laboratory' which is peaceful and free of noise, with pictorial signs about programming on the laboratory walls (NPL, ILE). This, they claimed, would aid the quicker understanding and retrieval of knowledge. They said the laboratory whiteboard needed to be replaced, because it was not visible to students at the far end of the laboratory (WB).

Recommendations about their expectation of teaching include: lecturer's knowledge of students' learning approach before teaching, group programming (GRPL), pair programming (PPP), projects and assignments (P, AS) with complex examples done in the class. They further noted that they expected programming teachers to: be friendly, use simple English, make video recording of each class, use real-life explanations and maintain eye contact with students during teaching (ETQ). In conclusion, all the students gave suggestion on how practicals should be handled in the programming classroom (SRP).

6.2.6 Reflective learning journal

The participants' summative reflections on both Scratch and QBASIC programming courses learnt during the semester were detailed separately in their reflective learning journals. From the analysis, 21 codes were identified. The codes, and corresponding categories, are presented in Table 6.17. Each code is further represented next to the sentence, or phrase, in the following discussions.

Table 6.17: Analysis of reflective learning journal

Sub-categories	Codes	Categories
QBASIC understanding Knowledge of QBASIC concepts	QBU KOC	QB understanding
Student personal issues Student health Student mood	SPI SHLT SMD	Student well-being
"Practical and explanatory terms" Presentation Practical Real life examples	PET P PRTL RLAP	Teaching strategies
Group negative impact	GNI	Group negative impact
Previous experience	PE	Previous knowledge
QBASIC interesting with Scratch Scratch boosts interest Scratch is interesting	QIS SBI SIN	Scratch developed interests in QB
Fun Scratch is easy	F SIE	Perspectives about Scratch
Sharing ideas Social interaction	SHI SI	Social interaction
Teacher's qualities	TQ	Teacher qualities
Electricity problem	EP	Contextual problem
Learning challenge	LC	Learning challenge

According to participants, some students understood QBASIC programming, while others said that they did not understand some aspects of programming (QBU). They ascribed their lack of understanding to: not being in class (SPI), not being in a good mood (SM), lack of concentration because of health status (SHLT). However, they said the teacher explained herself clearly (TQ) and the teaching of programming was interesting, especially the group presentations (P) and practicals (PRTL). Some ascribed their interest in programming to their mathematics background and previous knowledge of Scratch programming (PE).

For Scratch programming, the participants noted that they understood the topics taught (SU). However, one participant said he did not understand the order of precedence in Scratch (SU). They further noted that they loved Scratch programming because it was fun (F) and easy (SIE) and it facilitated the sharing of ideas in groups (SI). The practical and visual aspects of the classroom teaching also contributed to their interest in Scratch (SEI). One participant added that he found Scratch interesting because he had formerly been a graphic designer (PE). They noted that the teacher explained herself in clear terms (PET) and used real-life experiences (RLAP) to elucidate programming concepts. This benefitted them, especially when a concept was taught for longer than a week. They complained about the unstable supply of electricity because it hindered their practice (EP). To conclude, some of the participants noted that they had difficulty concentrating in class due to being in bad mood (SMD), having a terrible headache or being sick (SHLT).

6.3 RESEARCH FINDINGS – AR CYCLE 2

This section presents *phenomenological reflection* on data with the exact structures of students' lived experiences in programming. It involves both a *phenomenological description* and *interpretation* of the participants' lived experiences in programming (Van Manen, 1990). Therefore, the section commences with a description of seven participants' demographic data followed by a construction of themes using the thematic analysis of data (*q.v.* section 6.3.2).

6.3.1 Background information of participants

This section presents background information as to each of the seven participants including: educational background, reasons for joining the college and possible motivation for choosing computer science as a course of study. The participants were assigned pseudonyms. Although eight participants were interviewed, the voice of one of the interviewees (Uchena) was not audible. As a result, I was unable to transcribe the recorded interview. She was re-invited twice but refused claiming that she was very busy. This necessitated the decision to use the transcriptions of the remaining seven participants.

Joy is nineteen years old and she started her educational journey early because her parents were bankers. After the death of her father, her education was curtailed for three years before she gained admission to the college of education. She completed both her primary and secondary education at Goldbeam School Sango Ota. Her dream is to be a doctor, but because she failed biology during her final exams at secondary school, she decided to study another discipline, computer science. She did computer studies at the secondary level but had no prior programming knowledge or exposure to laboratory practical.

Benjamin is seventeen years old. He did his primary and junior secondary school education at St. Margaret and Estate Ilogbo, both at Ikorodu, Lagos state, Nigeria. He furthered his education at the government college in Ikorodu where he completed his secondary school education. After his final secondary school exam, he sat for Joint Admission and Matriculation Board (JAMB), a qualifying exam for university admission. The outcome was not satisfactory. His father does not want him to be idle and therefore secured him an admission to the college of education. He did computer studies at secondary school level but had not been exposed to either programming or practical knowledge.

Mercy is twenty-one years old and studied at Iberekodo Community High School and Magbon Alade Junior Grammar School, both in Lagos state, Nigeria where she completed

her primary and secondary education, respectively. She chose to study at the college because her parents are poor. Notwithstanding, she wants to further her education. She wants to be a pharmacist but as the courses offered at the college do not include pharmacy, she chose computer/integrated science. However, she noted that she was motivated to study the computer science aspect of the course because she wanted to gain knowledge about the internet and social media. She had no prior programming knowledge because there were no computer studies teachers in her primary and/or secondary school.

Joyce is seventeen years old and completed her primary school educational at I.D.C School One, Ibadan. She continued her education at Methodist N5 Grammar School where she finished her secondary school education. She was motivated to study computer science because she is interested to know more about the subject. She was not exposed to computer studies at the secondary school and therefore had no prior knowledge of programming.

Farai is twenty years old and completed his secondary education at Homat Comprehensive College Ikorodu, Lagos, Nigeria. He gained admission into Tai Solarin University of Education where he was admitted to study computer science but, due to financial restraints, he could not continue his studies. However, he was motivated to study computer science because he loves anything related to technology. He had no prior programming knowledge but was exposed to computer education at junior secondary level.

Faith is twenty years old and completed both her primary and secondary school education at Odomola junior and senior secondary schools, respectively. She was unable to continue her education because her father does not believe in educating female children. After staying home for two years, she started working at the college. Her employer was impressed with her final year performance and decided to sponsor her undergraduate studies. She had no prior programming background but was taught different programming languages during her second year at senior secondary. However, QBASIC programming was not taught in detail.

Peter is twenty-three years old and was born in the Akotogobo area of Ondo state, Nigeria. He claimed that his educational career, both primary and secondary high school at Akotogbo, was fun because he could confidently debate any topic in and outside the school. However, learning to read and write was quite challenging when he attended the Roman Catholic Missionary primary school. However, he loves music and his flair for IT made him decide to study computer science at the college.

6.3.2 Theme construction

Presenting an extensive phenomenological description involves determining the essential themes on which the phenomenological description will be built (Van Manen, 2002). The processes used for data analysis in the first cycle action research were employed in this second cycle action research (see section 5.3). Based on this premise, 177 codes were generated during the first cycle coding. The word cloud in Figure 6.21 below shows the prominence of the worded codes.

Figure 6.21: Word cloud representing the codes

During a second cycle coding, the patterned codes were clustered together to form 34 *categories* (Miles et al., 2014). These categories were further reduced to form 12 subthemes (Braun and Clarke, 2006). While determining the essential themes, some subthemes were discarded and compressed within other themes based on *latent content* (Braun and Clarke, 2006) to produce eight subthemes and three central themes. These themes are the *phenomenological themes* which truly describe the structures of the participants' lived experiences in programming (Van Manen, 1984). Table 6.18 presents a summary of the analysis. The comprehensive theme analysis and construction can be found in Appendix C 1.

Table 6.18: Summary of themes, subthemes and categories

Themes	Subthemes	Categories
Theme 1 Strategies for teaching and learning programming	**Subtheme 1.1** Teaching strategies.	Whole brain teaching strategies. Constructivist teaching strategies. Grouping benefits and negative impact on learning. Students perspectives and preferences for pair programming. Teacher's personality.
	Subtheme 1.2 Strategies students used for the learning of programming.	Cognitive learning strategy. Self-regulation of learning. Social interaction during learning. Problem-solving.
Theme 2 Programming knowledge gained by students	**Subtheme 2.1** Programming knowledge gained by students.	Perspectives about Scratch and QBASIC programming. Knowledge of concepts. Knowledge of solution development. Previous experience aided programming knowledge gained. Debugging skills. Word processing skills.
	Subtheme 2.2 Impact of programming knowledge on students.	Programming aids critical thinking. Programming impact on students. Scratch builds program writing skills. Scratch boosts interests. Creativity. Scratch inclusion to curriculum.
	Subtheme 2.3 Impact of student factors on the programming knowledge gained.	Students' strengths in programming and challenges faced. Student behaviour. Student well-being.
	Subtheme 2.4 Contextual problem as impediment to the teaching and learning of programming.	Contextual problem. Suggested solution to contextual problem.
Theme 3 Mental representation during programming	**Subtheme 3.1** Mental representation of the block model.	Atom. Macro-structure. Relations.
	Subtheme 3.2 Affective and behavioural states during programming.	Affective states.
		Behavioural states.

6.3.3 Description of themes

This section presents a phenomenological description of both the subtheme and the main theme. These subthemes are used to describe participants' lived experiences in the programming classroom. The empirical evidence is based on: interview transcripts, observation transcripts, students' reflective journals, researcher journal and whole brain feedbacks of eight participants, one non-participant observer and a colleague. The subthemes are supported with verbatim quotations from the participants. I acknowledge that the quotations are the participants' own words and therefore may contain grammatical

errors which cannot be changed. Pseudonyms used in the description are: Benjamin, Faith, Farai, Joy, Joyce, Mercy, Peter and Uchenna.

6.3.3.1 Theme 1: Strategies for teaching and learning of programming

The theme, constructivist strategies for teaching and learning programming, presents teaching strategies and strategies used by students in learning programming as subthemes. The categories describing the first subtheme are: whole brain teaching strategies, constructivist teaching strategies, scaffolding, grouping benefits and negative impact, students' perspectives on pair programming and teacher's personality. Constructivist learning strategy, self-regulation of learning, social interaction during learning and problem-solving, describe the second subtheme. Figure 6.22 presents the thematic network of Theme 1.

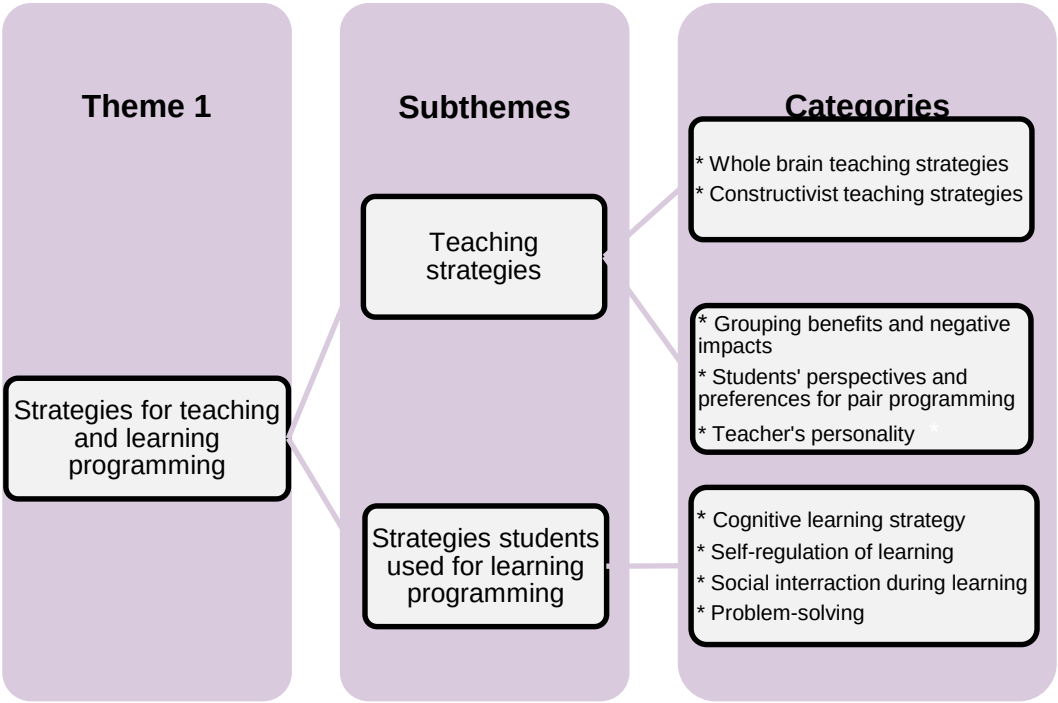


Figure 6.22: Thematic network of Theme 1

Subtheme 1.1: Teaching strategies

Teaching strategies link with different methods used to facilitate learning for the development of programming skills. The teaching strategies discussed in this section are: whole brain teaching and constructivist teaching strategies.

Category 1.1.1 – Whole brain teaching strategies

The participants discussed the significance of the whole brain teaching strategy in learning programming. They explained the use of whole brain quadrant grouping to aid learning, and group members were praised for any successful task. Joy said, *"I can say that also what helped me is the A aspect, and the B aspect"* and *"I really enjoyed because I was able to learn more".* In addition, whole brain teaching also boosted participants' thinking, "*since I do analysis, I also make a critical thinking…it affected me upgrading the way I handle things"* (Peter) and aided decision-making, "*when it comes to the decision making, and all these symbols, the rectangles and everything, that B, it helped me*" while learning programming. They also stressed that whole brain grouping gifted them with self-knowledge, "*I was able to use the opportunity to know more about self*" (Faith) and, *"I came to know about some of my weakness and what am very good at…and I know the best way for me to learn is when the practical aspect is being taught"* (Farai) and that of their peers, "*you actually know the strong people you have, people you can call for contribution. But...the unserious one, you will want to look down on them*" (Farai).

Category 1.1.2 – Constructivist teaching strategies

Constructivist teaching strategies were also used in conjunction with whole brain teaching. Group learning, whereby students discussed and brainstormed in groups, was used to facilitate the teaching of programming. A classroom observation reveals that, *"they were given a group activity where they produced a poem on steps in solving algorithm".* To support the findings, students' responses on the whole brain feedback reveals that the teacher almost always (64%) and frequently (36%) encouraged them to learn cooperatively in groups. Therefore, through group work, the students found the class interesting because of the group discussion and arguments which normally crop up during learning. As a result, the group work "*has made me to improve both socially and mentally*" (Joy, whole brain feedback). The participants also noted that group learning was beneficial to the learning of programming, because it enabled them to easily ask questions from group members, instead of going to the lecturer every time. In an interview excerpt Peter said, *"we carry ourselves along".*

Pair programming was also used during teaching facilitation. Students were paired so that they could solve a problem together. Observations show that it was used six times during the semester. For example, *"students were paired for pair programming. Others that their partner did not come to class today were paired with other people"* (classroom observation, 1 March 2017). Students were encouraged to present their projects. As a result, participants said, "*Data types in QBASIC…and we are asked to do presentation on it so with the*

presentation I understand the topic" (Mercy, reflective learning journal) and "*due to the presentation, the topic was not much as problem to me"* (Farai, reflective learning journal). Another participant added "*the presentation, I…it helped us. And everything I had in mind, I could pour to another people's hearing, so that they could correct me"* (Faith, interview excerpt).

Scaffolding, done by both the teacher and students, was prevalent during the teaching and learning of programming. Findings revealed that, "*as we are in group we used to help ourselves*" (Uchenna, whole brain feedback), and "*I have been of great help to my group and classmate at large"* (Farai, whole brain feedback). While "*I moved around to check their programs and guide those that needed help…after this, some student encountered problems in their program due to syntax error. Some of them could fix the error while some could not. Farai assisted Alfred and his partner. Queen also assisted students in group E and they were happy when the program was able to run"* (classroom observation, 7 April 2017).

Feedback on teaching and learning was also utilised. In one classroom teaching session, "*I commented on their solution and corrected them where necessary*" (classroom observation, 1 February 2017). Observation drawn from another lesson shows that the lesson "*commenced with a feedback and review of the error each group made in algorithm in the last class"* (classroom observation, 17 February 2017). Exposing students to project was also part of the strategies used for teaching and which, in return, enhanced the development of programming knowledge. They explained that "*giving students project actually help them and they will know easily what they are good at and what they are not good at*" (Farai, interview excerpt). Faith supported Farai's statement and recommended that *"giving us projects to do, groups and individual projects"* should be used. However, she added that even though some liked projects, *"some people are frustrated because…the project we are (were) doing is (was) much, and if we don't do that we won't have much knowledge of what programming is*".

Other constructivist teaching strategies used for facilitating the programming content were the use of explanatory terms and real-life applications. In support of this, participants noted that clear and deep explanations with real-life applications enabled them to perform well "*in algorithm and decision with repetition because the lecturer used clear and explanatory terms to gear the student interest about the topic thereby relating the topic to real life activities"* (Faith, reflective learning journal). This is also supported by 75% of the participants responding that the teacher frequently, and almost always, linked programming

to real-life situations (see section 6.2.2.1). However, one participant's experience in programming contradicts the above. He wrote, "*algorithm, flowchart and pseudocode, I gained a bit because the topic was fairly handled by the lecturer"* (Peter, reflective learning journal). Practical exposure to programming was used with one participant noting that "*on (in) the fifth class I understood the lesson because we did a lot of practical, class-work and group works that made me have more understanding*" (Joy, reflective learning journal). The use of cooperative learning which "*created opportunities for cooperative learning between learning group to achieve an aim, also encourages student to express themselves freely and openly*" (Mercy, whole brain feedback) was also noted. To corroborate findings, students' feedback of teaching and learning shows that 64% of the students supported that the teacher almost always encouraged cooperative learning in groups, while 36% also observed frequent use (see section 6.2.2.1). The practice of revising the previous topic was also utilised. It was observed that *"I revised the last topic on expression with them. I dictated the notes for the class verbally and the students wrote them in their notebooks"* (classroom observation, 10 March 2017). Students were exposed to series of assignments which they did not approve of. They explained that these series of assignments were sometimes too complex for them. This was revealed in an interview transcript where Farai said, "*not telling the students to work out an example and giving the students complex questions in her presence*". Other strategies included: individual learning and explaining programming using clear explanatory terms.

The constructivist strategies used to facilitate programming produced student engagement towards the learning of programming. It was observed that the teacher engaged the students 23 times (see Appendix C1) during the teaching of programming for the semester. The teacher involved the students in all programming problems, whether they worked in groups or individually. For example, "*they decided that one group should do the correction while another group correct the solution. They came to the board to solve and explain turn after turn*" (classroom observation, 10 March 2017). This was supported in the qualitative whole brain feedback of one of the participants who said, "*she knows how to carry the students along and make every student challenge themselves and participate in each activity"* (Farai). This result may explain why 66% of the students noted that the teacher facilitated learning by providing lively and encouraging programming sessions and 86% said the teacher maintained academic relationships that encouraged students to develop an enquiring mind (see section 6.2.2.1).

Category 1.1.3 – Grouping benefits and negative impacts

A more detailed discussion focuses on the impact of some of the teaching strategies, such as grouping and pair programming, on the learning of programming. With grouping, it was beneficial and at the same time had a negative impact. It was observed that there was cooperation amongst students during group learning. For example, "*in group B1, all group members participated in the group assignment. Joycelyn represented the group and solved the problem on the whiteboard and corrections were made where necessary. In group C1, all the students worked together on the assignment"* (classroom observation, 1 February 2017). In another class, *"I observed that there was no cooperation in the group because we divided ourselves into two and worked differently in the absence of the other two people"* (Group A, students' reflective journal). The following observation excerpt also supports students' lack of cooperation in group assignments.

> *"Mary: Ma, I have a witness. I have Peninnah and Joy. I am the only one that do (did) the assignment. When I called him on Monday, he did not answer me"*
> (classroom observation, 1 February 2017).

Joy also testified to a lack of cooperation as she detailed her experience in the following interview excerpt, "*I have more experience in computer but when but when we are in class and we are paired together and I was like explain to me…she was like hooooo, I don't"*. The participants complained that grouping had a negative impact on learning. They explained that the regrouping of students from different quadrants, after they were used to working with peers of the same quadrants, affected them. Because the group contained diverse students with varied thinking preferences, they seldom understood each other. In support of this finding, Peter said, "*I discovered that in my group, there are these set of people, I don't know maybe their quadrant is A, I don't really understand but the way they came up with solution is quite different. So, I find it difficult coping with them…I find it so confusing…sometimes that confusion may take as long as throughout the whole class will be over*". He therefore concluded that *"Scratch programming is ineffective among group members"* (Peter, whole brain feedback).

The attitude and behaviour of group members also impacted negatively on the learning of programming. Some group members left group work to other members while others did assignments without involving other group members. Also, students tended to not 'act free' in groups because of other group member's bossy attitudes. This is evident in the following interview excerpts, "*we won't know where the fault lies. It will be 'you said this, I said that' "* (Joyce) and *"if they are doing assignment they will not call me"* (Mercy). Faith explained one

of her group member's behaviour as "*her own experience she won't want to share with anybody. She believes she knows all and she will only want to work on the system alone and any idea anybody is giving her she will not want to use it"*. In support of students' claims, I also observed in one of the programming classes when "*Group A was called upon to present their work none of them came out whilst I have told them earlier to pick someone in their group. I have observed over time that that is their style*" (classroom observation, 1 February 2017)

Grouping caused struggles which resulted in some members becoming unhappy. Joyce provides a detailed explanation of the struggle she experienced in her group as, "*they won't take mine because is like I don't have a point. Assuming it is group A that was talking, so the group A will want his own to be the one that we will work upon because it is normal we won't have the same thinking capacity. So, group A will think that his own was the best solution alone. At least my own, they should pick a little from my own to make it a solution and we should pick it from the four quadrants to make a solution"* (Joyce, interview excerpt). In a classroom observation where students were regrouped into the four quadrant whole brain learning, I observed that *"some students were happy while some showed displeasure" and "a student in group C3 said he does not like his group members because they are not in good terms"* (classroom observation, 17 February 2017). Despite all the negative impacts of grouping, as observed and reported, students developed a coping strategy by slowly adapting to different group members' attitudes and behaviours. Joyce noted that *"after some practical we adapted to each other, we worked together".*

Category 1.1.4 – Students' perspectives and preferences for pair programming

In contrast to group learning, students discussed their perspectives and preferences for pair programming. From their experiences, they noted that pair programming fostered cooperation since it involved only two and not four people. It further allowed for freedom of expression and taking responsibility for actions. An interview excerpt supports the finding as Faith noted that "*when you are two in a group you know your position, what you are going to do... if you are given a question and I was the one that brought the idea and we missed it, I will know that the fault came from me*". In a classroom observation, "*as students work in their pairs, it was observed that one of the students was working on the system while the other partner was dictating the steps to the problem, checking the solution together"* (1 March 2017). With pair programming, it is "*so interesting in working with a colleague"* (Peter). Therefore, the participants concluded that they prefer pair programming to group programming. This is supported by Benjamin, Farai and Joyce, respectively, "*I prefer pair than group…I know how I will express myself more than when we are like four…"*, "*less*

noise, less distractions, more focused and less thinking" and *"in the two, if it is not this one we have to pick the other one and it is easy to assimilate, it is easy to understand".* However, other participants noted that programming in pairs *"depends on the person you are paired with. If you are paired with the person that can also give out his own idea, some people will be like you know it; I will be like…I can't do this alone, it's a pair work*" (Joy, interview excerpt) and "*the negative aspect is that eeeh, we don't easily respect idea*" (Peter, interview except).

Category 1.1.5 – Teacher's personality

The participants explained the teacher's qualities during the teaching of programming within a constructivist teaching environment. Farai and Joy explained that the personal attributes of the lecturer contributed to the learning of programming, because she "*usually attend class and working hard with us*", has "deep interest in the course", "expresses herself well" and "*she is good in what she does and make sure students have a good understanding of the subject taught"* (Reflective learning journal, 2017). Students' responses on the whole brain feedback reveals that 63% observed that the teacher almost always showed strong interest regarding programming tasks while 33% observed a frequent use. The participants, however, suggested other qualities expected from teachers towards the facilitation of the programming content in colleges of education. They mentioned: the use of simple English, "*using plain English with the student, they will understand than using the vocabulary, all those dictionary words…use the language they use"* (Joyce, interview excerpt), the use of video and grouping method for learning *"the method of grouping students, the method of bringing material to the class for video*, *I think they are okay for programming"* (Mercy, interview except), being friendly with students, teachers maintaining eye contact and video recording of class lectures.

Subtheme 1.2: Strategies students used for the learning of programming

In this subtheme, different methods used by the participants to accomplish the learning of programming is discussed. Categories 1.2.1 to 1.2.4 present the findings.

Category 1.2.1 – Cognitive learning strategies

In this subtheme, the participants learned programming through different means. Students learned through different constructivist teaching strategies used by the teacher to facilitate learning which, in turn, helped the participants to gain programming knowledge. The students constructed meaning on variables and repetition as well as debugging during programming practical. The following interview excerpts supply evidence as to their experiences during a classroom activity.

"I learned about when there is error, the computer will complain" (Benjamin);

"I learned that if the computer is not producing the required result, it is logical error" (Rachael);

"I learnt that if we are running a program with IF and you did not use ENDIF, because we tried it the other time. We solved the problem but when we removed ENDIF, it was telling us program can't be run without ENDIF" (Benjamin).

In addition, the following two classroom observations corroborate the findings, "*the students learned the concept of repetition by giving examples of real-life problems that can be solved with repetition"* (classroom Observation, 1 February 2017).

"*Faith: the number cough, there was space in between it and the thing didn't start with capital letter. So, when we now changed it, the thing was now coming out as it should come.*

Teacher: So, what did you learn from that?

Mercy: we learn that in Scratch, we must put a capital letter and there must not be a space…" (classroom Observation, 1 March 2017).

The participants also learned programming through equilibration and accommodation, as "*they came to the board to solve and explain turn after turn. When the first problem was solved by group B, there was an argument among the students to the extent that they shouted at each other at the top of their voices. I calmed down the situation and a consensus was reached on the solution. Through their argument, students who were at disequilibria accommodated the new knowledge gained on how to use expressions in QBASIC" (classroom observation, 10 March 2017)*. During a classroom activity, the students explained their learning experience through discovery as, "w*e observed that when we made a mistake, the Sprite did not draw triangle. We did not arrange that script very well. We arranged it correctly when we discover it. We now got the correct solution. We checked another group work that we discover a mistake"* (classroom observation, 22 March 2017).

Category 1.2.2 – Self-regulation of learning

Another means through which students learned programming was through self-regulation of learning which they achieved by maintaining a positive emotional climate, *"students were seen dancing and clapping and as well happy during the formation of their poems"* and *"students that were able to get the desired result smiled at their work and moved on to the next activity*" (classroom observation, 17 February 2017). With regulation of cognition, "*the first time they came to the lab to run the program, they made a mistake and the program did not run. They had to come back to rerun the program again*" (classroom observation, 22 April 2017); focused goal attainment, "*but student A1 noticed that the program was not correct. They run it again and tested it several times to ensure its correctness*" (classroom

observation, 29 March 2017) and "*when I got home I do use to read my note to try to understand*" (Mercy, interview excerpt). In addition, they learnt by seeking help from the teacher and their peers. For example, Joyce noted in an interview excerpt, *"I don't like it when am doing something and it doesn't yield a positive result…and…I don't like trying things …on reaching the fourth person the program will run…use other ideas to work on your own project…"* and "*they could not do it and so they called on student F2 to assist them. Then they asked from him what he pressed and he showed them again"* (classroom observation, 29 March 2017). They also challenged themselves to do more complex programming activities, *"it has been of great help to me in the aspects of being creative, hardworking, thinking fast, challenging myself"* (Farai, whole brain feedback), transferred knowledge "*students used ideas learned from last class to ensure creativity in their work"* and valued their learning "*I called Faith and Farai to organize tutorials for all new students. They told me they have already fixed Saturday 25th for tutorials"* (classroom observation, 24 March 2017).

Category 1.2.3 – Social interaction during learning

Social interaction, which the students engaged in during the learning of programming, enabled them to interact among themselves and in groups. Joy said, "*I will say I gain a lot from the group work we do (did), and learn to interact, and help others"* (whole brain feedback) and *"they all echoed 'it was interactive". One of them said 'it was as if we should not leave the lab"* (researcher's journal, 8 February 2017) and "*the class was very educative, were (we) all shared our ideas and understanding"* (Joyce, student's reflective journal). They also made friends and expressed themselves freely, *"I express(ed) myself well, participating also to develop an enquiring mind"* (Mercy, whole brain feedback), such that the introverts learned to interact and develop explaining skills, *"everybody shared their opinion...collate everything together, so if anybody knows anything in the group, we gather it together…that will help us to have more knowledge"* (Faith, interview excerpt). As well as contributing during programming activities, "*there are some case (s) that it will be the person that will bring the right answer and me I can bring in the wrong answer. The person will try to put me through and try to explain…from that am gaining something"* (Benjamin, interview excerpt). In addition to the evidence already presented, students' responses on the feedback questionnaire revealed that 52%, 28% and 14% respectively, noted they frequently, almost always and occasionally expressed themselves well in the programming classroom.

Category 1.2.4 – Problem solving

They also used brainstorming and devised group dynamics which helped them to solve any problem. For example, during classroom observation on 17 March 2017, students brainstormed while communicating in their mother tongue, sometimes mixed with English:

Queen*: "IF Pay = 4.50 ELSE Hours is less than"*

Nina: *"greater than (correcting A3)"*

Queen: *"Please check (others checking the solution)"*

Nina: *"Regular is 4.50, if it exceeds, status is 6.5, cancel ELSE…"*

Nina and Queen: *"That ELSE suppose not to get here since it is still 40…"*

Faith*: "it should be there. We are testing for the second condition now if the first one does not work, we go back to test for the second..."*

Nina: *"As in you did overtime work, your money will increase"*

Queen: *"Wait, wait. This thing is getting clearer" (they all check their solution again).*

In addition to brainstorming and the use of group dynamics, students related the frequent exposure to problem-solving activities in the programming classroom as helping them gain confidence to solve assignment and problems given to them. One participant said, "*because I was opportune to solve problems, if I get home I will do my assignments*" (Mercy, interview excerpt). This also enabled them to help their peers. Farai said, "*when my group come to me, I have to explain how*" (Farai, interview excerpt). This may explain why students' responses on the whole brain feedback questionnaire reveals that 93% almost always and frequently developed a better sense of responsibility towards the learning of programming. This further propelled 86% of the students to frequently and almost always contribute to their peers by helping them find solutions to problems.

6.3.3.2 Theme 2: Programming knowledge gained by students

This theme has four subthemes: programming knowledge gained by students, impact of programming on students, student factors on the learning of programming and contextual problem and suggested solutions. The thematic network is represented in Figure 6.23. According to the network, programming knowledge gained by students was influenced by both contextual and student factors which, in turn, influenced the impact programming had on the students. The subthemes are explained in the following sections.

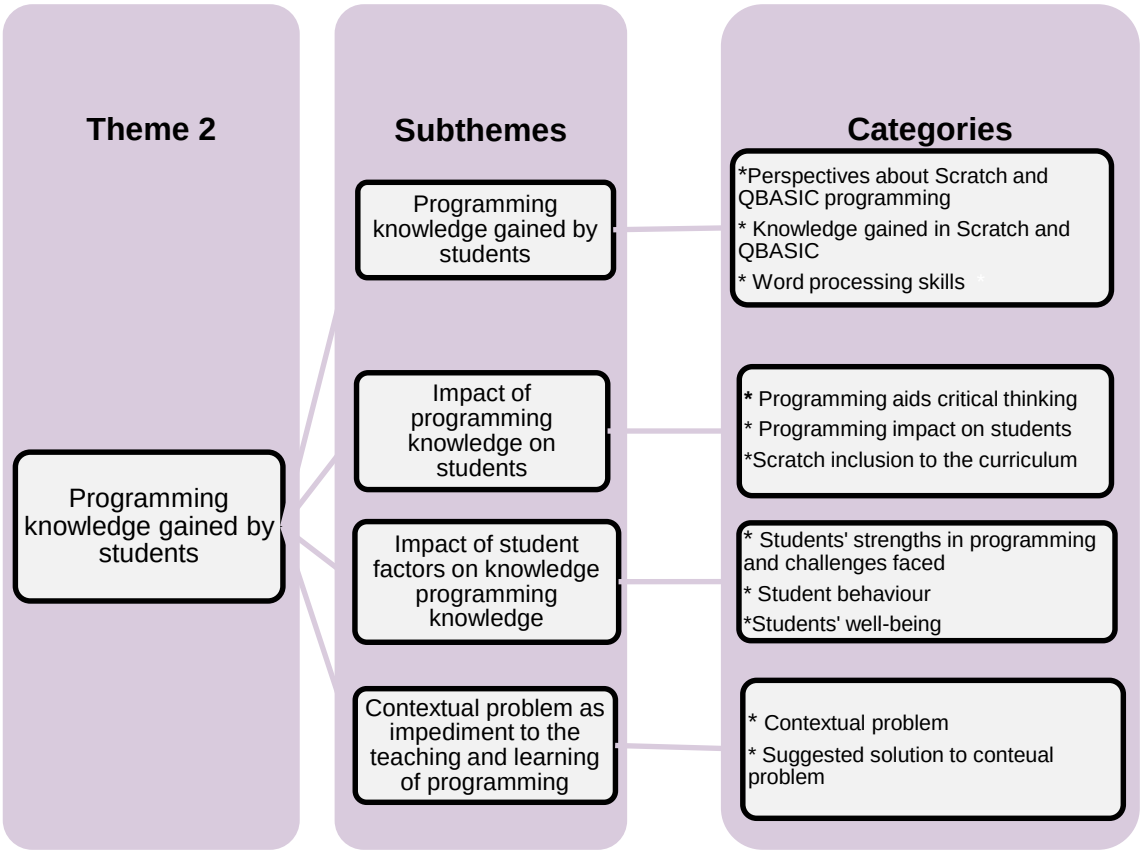


Figure 6. 23: Thematic network of Theme 2

Subtheme 2.1: Programming knowledge gained by students

This subtheme describes the series of programming knowledge students gained in the programming classroom. These findings are discussed in categories 2.1.1 to 2.1.5.

Category 2.1.1 – Perspectives about Scratch and QBASIC programming

Initially they described their perspectives on Scratch and QBASIC programming as fun, easy and enjoyable. It was easy to learn and apply knowledge gained in Scratch to that of QBASIC. Evidence from the interview excerpts, and students' reflective journal, support these findings: *"Scratch is a fun and easy way of writing a program"* (Farai), "*Scratch is more enjoyable*" (Joy), and *"I enjoy most especially when we make a variable, we inserted all our blocks…the program started doing what we instructed it to do. I was like ho! so all this thing I have (been) doing since morning, so this is what is coming out*" (Peter). He added that, "*it's a bit easy to understand than QBASIC…*"and "*most of the things we did in Scratch, we moved it to QBASIC. Learning it first from Scratch made it easy for me to understand it in QBASIC"* (Farai). While discussing Scratch application in QBASIC, the participants said, "*Scratch programming and the QBASIC, they have some similar topics"* (Joyce). Hence, "*it helped me to learn the QBASIC because I have a little knowledge about Scratch, so combined with QBASIC*" (Mercy). Also, *"using the order of precedence in Scratch to that of*

QBASIC…I saw it on Scratch area, I will know how to solve it in the QBASIC, that also deals with the order of precedence" (Joyce). As a result, QBASIC became interesting because of Scratch, and *"I prefer the both"* (Joyce) but my friends *"found Scratch fun and they enjoy (ed) it more than QBASIC*" (Farai). One participant concluded by saying *"if one is serious after doing Scratch, now QBASIC is easy"* (Faith). On the contrary, Faith, Farai and Joy said they understood all aspects of Scratch which was taught. Mercy said, "*the Scratch, affect(ed) the QBASIC, because I did not really get that Scratch*" and *"I don't understand the order of precedence very well*" as well as *"working with costumes*" (Joyce).

Category 2.1.2 – Knowledge gained in programming

The participants discussed their perspectives on programming with regard to the knowledge gained in Scratch and QBASIC. All the participants, except Uchenna, noted that they understood the concepts learned in QBASIC programming. She claimed not understanding concepts, like expressions with order of precedence, *"because we were discussing a lot about mathematical aspect*" and "*because I just joined you*". This may explain why her performance in programming was low. With Scratch, participants noted that they understood topics learned because the topics were repeated. Farai supported this claim by saying, "*the topics we treated in Scratch we also treated in QBASIC, so being taught something repeatedly, you will get more understanding about the topic"*. Faith added, "*we were allowed to solve questions on our own, we were given projects in our groups and individually and that made us to have the experience of what programming is"*. However, Joyce, Peter and Mercy claimed that they did not understand order of precedence and algorithm with decision. For example, Joyce said "*I don't understand the order of precedence very well*". This could be because she lacked mathematic grounding. Other participants' reasons for a lack of understanding included depression and health issues. These are further discussed in subtheme 2.4.

Further discussion about knowledge gained in programming include: flowchart algorithm, running of programs, program writing, variables, repetition, control structure and understanding about QBASIC. With regard to flowchart algorithm and variables, some of the students can define variables and solve programming problems using the appropriate flowchart symbols. For instance, "*Group B2's solution to the problem shows they now understand the problem to some extent…they used the correct flowchart symbols to depict the solution. They made use of variable and could use good flowchart statements"* (classroom observation, 10 February 2017). In contrast, others did not grasp the importance of variables and therefore did not utilise them in their solution. For instance, "*Group C1 members, their solution also lacked variables"* and *"Group B1 used the correct flowchart*

symbols. Their flowchart lacked the use of variables. In group C2, members used the right flowchart symbols. They did not use variables in their solution…Group D, could use correct symbol…they did not use variables after the scaffold" (classroom observation, 10 February 2017). In addition, students' performances on algorithm during the test of individual concepts shows a low percentage score of 31% ,19% and 21% for questions 7, 8 and 10 respectively (see section 6.2.4.1). Reasons for these findings could be that the students were just learning to program. But towards the end of the semester, findings from a final test the students were exposed to reveal that students *"did not apply algorithmic techniques in writing program"* and the percentage correct answers for an algorithm type question solved during test shows a 36% score (see section 6.2.4.3). With variables, the percentage score during the test of individual concept II and final test were 56% but they could not identify string variable while 73% could identify both string and numeric variables. On repetition and control structure, Farai noted that *"the use of repetition on Scratch was made easy because we learnt more on repetition in QBASIC before moving to Scratch"* but "*students don't know when to use a decision*" in a program involving control structure.

Other programming knowledge gained relates to the running of programs. The participants claimed that they could run programs. Running their program enabled them to witness its output and make corrections, especially with Scratch which gives a visual output. Faith said, "*everybody likes the running of programs…I was able to run programs because I couldn't do that before…if we write a program and is not correct you won't know until you run the program in Scratch…after running the program you will know the place where you made mistake"*.

Category 2.1.3 – Previous experience aided programming knowledge gained

The participants claimed that the series of programming knowledge gained could be linked to their mathematics background and the lack of previous practical experience in secondary school. A classroom observation reveals that students "*cannot solve simple mathematical calculations, on expressions in QBASIC programming"*. In contrast, both Joy and Faith claimed they understood expressions using order of precedence in programming "*because I am a mathematics student and I know the rules of solving problems in mathematics"* (student reflective journal). Many students lacked practical experience in secondary school, one participant said "*that's what affected me in my secondary school, just explanation*" (Faith, interview excerpt). To corroborate the finding, "*majority of them could not locate the costumes, so I showed them how to locate it but because the students were not familiar with the computer system, it was difficult for them to find their way through*" (classroom observation, 15 February 2017).

Category 2.1.4 – Debugging skills

Scratch programming also enabled the participants to develop debugging skills in programming. One participant claimed that Scratch itself enabled her to correct her program whenever she made a mistake because "*you will find out that in Scratch, what you've written down may not be the correct one…when you are in Scratch, the Scratch will be correcting you*" (Joy). One is not sure whether Scratch helped to improve their debugging skills in QBASIC, but participants could identify errors and debug the program. Evidence from a classroom observation shows that, *"they run the program and noticed that there was error in the program. They noticed they did not include the 8! in the program, then they re-run the program again. As they did, they corrected the program and run it again but there was a syntax error such as 'overflow' which they corrected*" (classroom observation). Farai, while writing his program said, "*I changed the data errors that were in the program so to be able to get the right result"* and Faith, *"I tried checking it, I saw that I didn't put the semicolon so I corrected it"*. When the participants were exposed to a retrospective interview, evidence shows that many of them could not correctly detect, or debug, syntax errors found in the program tested. For example, while Joy was testing her program, she encountered some errors which she could not identify. She said, *"hmmm, I commanded it to stop because IF Counter = 5, I said THEN one person…is it not logical, that is logical error and digital error*". Also, with Benjamin, "*it was telling me that I should like…not to use different something like that sha. I went back to the program, I cleaned everything on that line and rewrite it again. It was still giving me the same thing*". Joyce declared a numeric string variable as a numeric constant. She experienced a syntax error with 'Gain' which she declared as a string variable. She explained her experience as, *"when it said duplicate definition I was like, and where it underlined it was under the gain and it was even in the program…So, I was now thinking that maybe this is where I am supposed to change the program. So, I now put eeem, I went to declare that gain as a variable. So, after declaring it I put the dollar sign in all my gain maybe…So, after that the thing now said, "variable expected" under the same word and there is variable in front of it"*. She did not detect this error until I told her at the end of her program.

Teacher: "Gain in that program, what kind of variable?"

Joyce: "Gain, it was supposed to be numeric variable."

Teacher: "And you declared it as a string variable…"

Joyce: "Haaaaaaa".

To support participants' experiences, evidence from a final test (involving debugging of a looping type question) shows the percentage of correct answers at the end of the test to be 34.6% (see section 6.2.4.3).

Category 2.1.5 – Word processing skills

One area relating to program writing which the participants explained, and which was observed in the classroom, was a lack of word processing skills which impacted negatively on the way in which the students program, "*I observed that they were looking for how to press a quotation mark for string".* Also, in another classroom observation *"group C members, said the program for question 4 was overwritten because they used the same name to save the same program but they already wrote it down on a paper"* (classroom observation, 7 April 2017). In addition, during a retrospective interview, Joy pressed a key on the keyboard that made her work disappear, "*I could say maybe because of eeem, I wasn't used with computer…So I was like what did I press and everything I was doing moved away…I was confused. I don't know what to press, and the help statement was not...maybe because I have not used the help statement before…I pressed tab, it was like the program disappeared…I wanted to press this eeem, greater than, so I was like I pressed the shift and it didn't come up. So, I was like maybe if I should try this, this tab"* (Joy, interview excerpt). Mercy's explanation also showed that she had no word processing skill which affected the program written, "*I did not press the normal thing I suppose to press. I want to press shift so I don't know how my hand just press another thing. I don't even know what I pressed"* (Mercy, interview excerpt). Towards effectively using this word processing skill, students suggested that initial exposure to word processing would be helpful.

Subtheme 2.2: Impact of programming knowledge on students

This subtheme discusses different factors which impacted on the programming knowledge students gained. These findings are discussed in the categories below.

Category 2.2.1 – Programming aids critical thinking

The impact programming had on the participants made them conclude that programming "*is a great course and is a kind of course that gives someone a kind of serious thinking…develops in the individual the way someone think and…handle his own things…"* (Peter, interview excerpt), and "*programming itself, it's very interesting and it involves thinking*" (Joy, interview excerpt). To support participants' statements, a classroom observation reveals the conversation between the teacher and a student experienced while programming.

"Teacher: So, what did you learn from that?'

"Group G, G1: I learnt that when we want to solve a problem, we should first think, understand the problem before we do it" (classroom observation, 1 March 2017).

Hence, Farai said *"since I joined the class and started this course, it has been of great help to me in the aspects of being creative, hardworking, thinking fast"* (whole brain feedback).

The participants said they engaged in critical thinking when solving any programming problem. They could engage in critical thinking and reflect on their learning because the teacher frequently (48%), almost always (38%) and occasionally (14%) encouraged them to do so (see section 6.2.2.1).

They see Scratch as a programming language with no serious thinking, mostly because of its low walls. Scratch does not require serious thinking, *"since all information needed are on the program, no need of serious thinking (word intensified and laughed). Am looking for goat, I go animal, just to pick the goat, maybe sprite 1, sprite 2…So, it's been a kind of eeh, creative thinking to a programmer"* (Peter, interview excerpt). Scratch does aid thinking while writing and running a QBASIC program. According to them, it also aided thinking in QBASIC, *"the thinking you are thinking in Scratch that makes you that okay you are seeing things that you are dragging, you now come to QBASIC you will want to express that thinking like okay, when I want to create it, in my own word is, and you have to follow the pattern like let me say follow the rules of QBASIC, the way you are creating it you will fall in the wall of QBASIC. So, it will help you"* (Benjamin, interview excerpt). They also noted that because they had developed the ability to think critically in the programming classroom, it would not be difficult to apply it to other areas of life because, *"most times you are programming, you are thinking, so when they give you other things, like other things from other, even though you don't know it your thinking mentality will like give you some ideas"* (Benjamin). Mercy concludes by saying, "*Scratch programming is very good programming but someone needs to be very wise and think ahead to be able to write programming well".*

Category 2.2.2 – Scratch built program writing skills

Scratch impacted positively on students' program writing skills. One participant said, referring to creating programs, "*anytime we want to write a program, if you can write it down step by step, you will be able to write the program in real sense in QBASIC"* (Farai, interview excerpt). Another participant said it aided program writing in that *"it helped…to write a program…orderly arrangement of program statements…so if you know how to arrange a program block very well, the QBASIC will not be difficult for you*" (Joyce, interview excerpt) and *"I need to follow the blocks in order so that I will not make mistake. So, it helped me the same way in QBASIC program especially when am writing a program. The rearrangement helps me to understand when program can be arranged in order in QBASIC program"* (Peter, interview excerpt). They further claimed that Scratch helped them to learn the order of precedence, "*like example was the arithmetic expression, we are like how are we going to first do this, do this. But by the time we work it on the Scratch in the class; we saw the usefulness of the order of precedence"* (Joy, interview excerpt). A classroom observation

supports the findings, "*they were asked to write an individual program which reads six students name they solved the problem and asked questions where they don't understand"* (classroom observation, 22 February 2017).

Category 2.2.4 – Programming boosts interests

Another impact the students felt relates to them learning both Scratch and QBASIC. Presentation, frequent practicals and interactive classrooms sessions resulted in them becoming interested in Scratch programming. This finding is supported with excerpts from participants' interviews and reflective journals, "*the day we started programming, everybody was happy. So, the next time they called us, we won't be sluggish about our moves, we ran up, because we are interested in Scratch*" (Faith), "*the class was very interesting. I can say that without the practical in Scratch, the QBASIC will just be explanatory…I think Scratch is more interesting than the QBASIC"* (Joy) and "*something that interest me most is the Scratch class*" (Faith). One such reason why students would prefer Scratch to QBASIC programming is because of its graphical feature.

During one classroom observation, the students developed "*creative in their project using different backgrounds and they learnt the concept of repetition by creating their own animations*" (classroom observation, 8 March 2017).

Category 2.2.4 – Programming impact on students

Participants voiced their feelings and made future career decisions based the impacts experienced in programming. Farai said, *"the way I see, with more understanding about it, it is easy to become a programmer. So, I will love to be a programmer".* Likewise, Joy said, "*I feel okay and…am bold of myself that am studying computer science in a college of education and be able to explain computer science*". However, others only expressed their feelings about programming and said *"I feel happy" and "I gained a lot in terms of algorithm…flowchart, in terms of READ statement, in terms of INPUT statement…arithmetic expression"* (Peter, interview excerpt). Joyce concluded by saying "*if you want to do anything about programming it is essential for you to sit down and think before venturing into programming because it is not that easy…look at the program before you solve it".*

Category 2.2.5 – Scratch inclusion to the curriculum

Based on their experience, they said Scratch programming should be included into the curriculum with foundation aspects being taught at both primary and secondary schools. The following interview excerpts support the findings, "*Scratch is going to be useful in every area of programming in computer*" and therefore "*should be taught in primary school, maybe*

peripheral aspect of it…I will suggest that programming should be taught not only in higher education level only but also in secondary school, maybe primary school, they can do the smaller aspect of it. Then secondary, it will be a kind of maybe…this one maybe primary their experience is lesser, maybe secondary is semi…so it will help Nigerians to transfer knowledge from one concept to another and explaining in Scratch will not be too so strange" (Peter, interview excerpt). He added that the computer science curriculum should be adjusted so that programming becomes a double major course. He stressed that "*there is this other one…Nigerian colleges most, even I will confess that there are (is) much load on students; thinking this one, thinking that one, as in course that students do in colleges of education. And this programming aspect of it, is a very wide course on its own, so I will suggest that for colleges, programming suppose be a kind of course on its own. At least there are these people English students; they are not doing any other course. English, English students, BASIC, BASIC students, so they should introduce that kind of".*

Subtheme 2.3: Impact of student factors on the programming knowledge gained

In this subtheme, different factors that impede students' learning of programming are discussed. Categories 2.3.1 to 2.3.3 presents the findings.

Category 2.3.1 – Students' strengths in programming and challenges faced

Findings show that one of the factors that impacted on the learning of programming was the strength and challenges faced while learning programming. These include challenges with complex research works, explained by one participant during classroom teaching, "*was a likeness for algorithm, and my strength is solving concrete algorithm, my challenge is having to research on a topic and solve complex task*" (classroom observation, 3 February 2017) and "*am saying if it is too much, if like, am a kind of person that if am doing the same thing, like the person is giving me more difficult, more difficult at the same time and not getting it. It will get to some extent that I will be fed up. I will not want to like to participate again"* (Benjamin, interview excerpt). Another challenge relates to encountering new experience/s which resulted in a participant becoming off balance during a retrospective session. He was asked to rewrite a program using a FOR…NEXT or Subroutine. He experienced a syntax error which he had never seen, although the work had been taught, and this caused him to comment, that *"this is a new experience. How can I solve this? So, I haven't encounter such problem before"* (Peter, interview excerpt). Joy likened her own challenge regarding new experiences to exposure to the laptop for programming. She said, *"I did many practical but not on my laptop...because there I have never made use of this help…I now figure out that it's the same thing. Maybe because I have not used a laptop before*" (Joy, interview excerpt). Farai's new experience relates to programming commands. He said *"…but the reason why*

I find it a bit difficult is because when we are using the IF…THEN statement, like I was taught…So I have not used more than two IF…THEN statements. So, in this program you have to do it for like three to four times".

Despite the benefits accrued from Scratch programming as an aid to learning QBASIC programming, the participants still experienced some challenges. Because Scratch was not a major course, "*my friend usually think Scratch is not necessary than the QBASIC because they believe that the Scratch is not going to come out in our exam or something, only the QBASIC*. *So, they were like why do I go to Scratch class…but me as a person, after the information, I like to see the visual…it makes me to understand it more"* (Joy, interview excerpt). Others stated that they feel "*frustrated like the work is too much. Some feel they don't like the class and some feel very happy about it. Most people like the eeeem, importing of costumes and some people like the programs we did like the QBASIC program*" (Faith), *"when someone is to look for a particular thing and couldn't find it"* (Peter, interview excerpt).

Category 2.3.2 – Student behaviour

Student behaviour also impacted on the learning programming. Student behaviour relates to those actions exhibited by the participants in the programming classroom. Desired behaviours, which in the long run help the learning of programming, are expected to be exhibited by the participants in the programming classroom. For example, during the first lesson, *"I observed that in group D, Peter was not talking, and then I said the group members should carry him along…At about 5 minutes when I checked the group again, Peter was seen fully participating in the group activity"* (classroom observation, 17 January 2017). In another lesson*, "I noticed that Group A students quickly rose up to present their project when called upon, unlike what happened in the previous class"* (classroom observation, 3 February 2017). In one of the classes, *"there was no board marker to write on the board and Alfred bought a marker for the class without being asked to".*

Undesired behaviour was exhibited by the participants. For example, when *"Group A was called upon to present but none of them came out to present whilst I have told them earlier to pick someone in their group. I have observed over time that that is their style"* (classroom observation, 1 February 2017). Students' impatience when experiencing problems on a particular task, "w*hile running the program, Alfred suggested that they close the program and move to the second question, but the other group members persisted and solved the problem* (classroom observation, 24 March 2017). Another behaviour exhibited by Alfred, and complained about by the group during a project presentation was, *"nonchalant and disrespectful behaviour*, but *Alfred also complained on non-availability of the group*

members. He said, 'they are not time conscious'…" (classroom observation, 3 February 2017*).* To support group members' claims, *"I also observed that Alfred is a type of student who was in his seventeen's and always want to voice his mind on every issue. He hisses uncontrollably, easily get angry, and shout at the top of his voice when he is angry. However, he has his good side"* (researcher's journal). They also had excessive arguments which did not encourage learning among groups. That was why Ogechi complained about her group members' behaviour during one of the programming classrooms, "*they argue, argue. That's what am saying, we do it separately then we join. How can we do it when this one will not agree with this one? Ha! Since we don't know it, let's try ourselves"* (classroom observation, 18 March 2017). Others are sleeping during the class, "*in the class today, student Joyce was sleeping"* (classroom observation, 24 March 2017) and using their phone while the teaching and learning was going on *"close to the end class, Bello was busy with the phone and so was not concentrating on the class activity. So, I collected the phone and returned to him after the class"* (classroom observation, 10 February 2017), lateness and negligence to assignment.

Category 2.3.3 – Student well-being

A student's well-being refers to his/her physical and emotional state which impacts on his/her learning, either positively or negatively. Their well-being determines their readiness for learning. In a case where they were unable to learn, they could balance their learning using other means. For example, during a QBASIC class, *"I did not concentrate in class because I was not feeling fine that day but based on previous knowledge, I was able to gain something"* (Faith, student reflective journal) and *"I was not in good mood and had a slight headache which really affected me"* (Joy, student reflective journal). Supporting Joy's claim, I also observed that *"from the students' faces I could see from their expressions that they were tired"* (classroom observation, 7 April 2017). Another participant noted, "*on our seventh class in Scratch...I had a terrible headache which did not allow me concentrate but I later practice on my own*" (Joy, student reflective Journal). Peter said, "*my understanding in the second and third class were the same because I was hungry and feel tired...on QBASIC arithmetic expression, I understand the topic but my rating dropped because I'm not well".* Peter was a slow learner and prone to depression, therefore *"working in QBASIC was like war for me"* (Peter, student reflective Journal).

Subtheme 2.4: Impact of contextual problem on programming teaching and learning

This subtheme contains contextual factors which affected the learning of programming during the semester. Categories 2.4.1 and 2.4.2 present the findings.

Category 2.4.1 – Contextual problem

In one of the Scratch programming classes students were to learn algorithm with decision and repetition but *"there was no light to project the lesson for the day. I was told the generator would not be on throughout the day"* (classroom observation, 1 February 2017). One of the participant said, *"I found it a bit difficult…insufficient supply of electricity*" (Peter, interview excerpt). Another contextual factor, noted by participants, as a barrier to the learning of programming was the lack of functional computers. This issue also relates to the wavering electricity supply, as discussed above. In one of the programming classes, *"because of the limited number of computers, two students were attached to a system, which was not part of the planned teaching for today's class. Even some of those that were attached to a system, their system broke down along the line, for example (group C2). They were given another system to work on*" and "*the projector developed a fault and the prepared slides could not be shown on the whiteboard"* (classroom observation, 8 February 2017). Peter therefore added that, "*the worst experience and dislike about the whole lesson was…we don't normally have enough time practicing"*.

Category 2.4.2 – Suggested solution to contextual problem

The participants suggested solutions to the identified contextual problems which were extensively discussed. These discussions included, "*programming is not just about sitting in class, explaining, but if we can practice everything we've been doing in the class, that will give us more knowledge about the programming language we are into…because this is computer, not other subjects"* (Faith). Supporting this statement, Benjamin said, "*the main thing that will make programming better is, I will like us to do practical…Out of the people you are teaching, there will be some people that their brain is sharp. They will understand what you are doing".* Another suggestion, by Farai, focused on the way in which the practical should done. He said, *"before going for the practical, we have to have little understanding in class before moving to practical. From there you have better understanding"* and "*it's not supposed to be 12-2pm, it supposed to be every time"* (Joy).

Another suggestion was based on the provision of more facilities, "*facilities like laptop or anything"* (Farai). Students "*should make use of their own system, not they will be…when they want to write they should write individually*" (Mercy) and "*should be an individual class, so that it will be you and the computer. So that what you do you see it yourself and no one making jest of you"* (Joy). On the issue of electricity in the college, the participants offered the following:

"adequate supply of light" (Peter);

"*there should be electricity because you are dealing with computer programming; you have to practice what you've done"* (Joyce);
"the light aspect, there should always be electricity…we choose, sometimes when you give us project, you find out that we will be like there is no light" (Joy).

In addition, the environment must be peaceful with a *"constant supply and air ventilation, there must be enough air coming in maybe through the window, fan or any air conditioner. It must be a cool environment so that when we are in any cool environment, student learns thing easily"* (Joyce) with *"pictorial signs or pictorial work that students can use to refer to a particular concept, for instance data type so on, that is programming suppose have its own environment...that if you enter, all what you will be seeing is all about programming. Even on the wall, even on the table, everything all what you will be seeing is all about programming. That is there supposed to be a programming environment in colleges of education*" (Peter). Also, the whiteboard should be visible to all students, because *"when you are writing sometimes on the board some students at the back will not be able to see what is going on in the front…they should use another; they should make use of where that everybody will be able to see what they are writing"* (Mercy). The participants lay emphasis on research works which they said, "*should be given…to make about the topic before coming to the class. If we research about it, if the teacher asks any question about the topic, we will be able to give her the required result and the lecturer won't be able to stress herself because she will believe we know everything"* (Faith, interview excerpt).

6.3.3.3 Theme 3: Mental representation and students' states during programming

This theme describes the content of the participants' comprehension activity obtained from the programming codes provided (Appendix A 5) and their affective and behavioural states during programming. The following subthemes: mental representation of the Block model, and students' affective and behavioural states during programming, are discussed. The thematic network is shown in Figure 6.24.

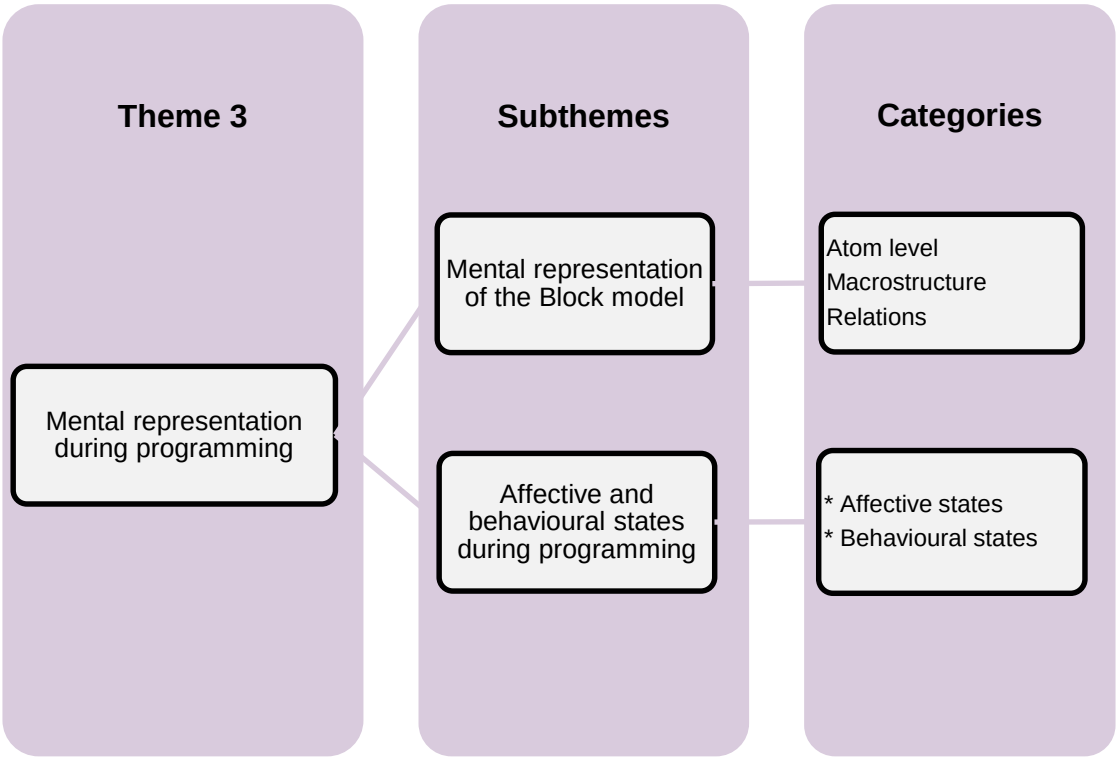


Figure 6.24: Thematic network of Theme 3

Subtheme 3.1: Mental representation of the Block model

This subtheme is divided into three different categories namely: atom, macro-structure and relations. They focus on the participants' mental representations on the knowledge dimension of the Block model.

Category 3.1.1 – Atom level

This description focuses on the atoms, the level at which participants first internalise the program elements. The atom level comprises two comprehension dimensions: the text surface and functions. On the *text surface*, participants could identify variables in the code presented. In code 1, Joyce and Farai said the variables in the code are "*Name$ and SALES*", while Farai added "*GAIN*" to the variables he listed. He explained further how he identified the variables. He said, "*they are the one that start with capital letter and they are not long and the string variables are the one that end with dollar sign. I looked for the one that have these attributes in them". In code 3,* Faith, Joy and Benjamin listed "*MATHSCORE, ENGSCORE, BIOSCORE, TOTSCORE*" while Faith added "*AVESCORE and DOUBLESCORE*" as variables in the code. Joy explained how she identified the variables. She said, "*I also identify that the total score is going to be a variable because it's going to contain the Math's score, the Biology score, and the English score*". Peter did not list the variables but only identified them as string and numeric variables. In support of the findings, percentage correct answers of students on the text surface level reveals that

percentage correct answers of the unistructural SOLO levels include test of individual concept II (41.35%), interim test (29.3%) and final test (51%).

On *functions,* where participants were expected to supply the meaning of 'GOTO' and 'COUNTER' in the context of the program, the participants' mental representations captured the function of each statement in the code. For example, in code 3, "*GOTO in the program is unconditional statement, we make use of GOTO instead of going back to the first step...we go back to the next step*" (Mercy). Benjamin said, *"if counter is counting student and counter count 1 student, it is the function of GOTO to direct the program back to the counter*" and Faith added, "*the counter is telling us the number of times we are to run the program and the GOTO is telling us the next step after calculating the total score*". Joy gave a comprehensive explanation, "*IF Counter is < 5, THEN GO TO 20. That means it is going to start over to calculate. Because the counter = 0. That means if the Counter is not yet 5, then is going to go to 20. That is, it will start calculating. Then IF Counter is < 5, THEN GO TO 90, Then GOTO 140 because it's going to END".* In code 1, Joyce said "*the counter count sales one after the other and the GOTO allows the counter to go to the next sales*", while Farai explained counter as, "*the count is used to count for 20 salesmen*". He also viewed COUNT as repetition structure, *"count is used when we have to calculate for not just one person. So, you make use of COUNT, Sometimes repetition. You have to start the first one before the next, then you start over again".*

Category 3.1.2 – Macrostructure

The second category is the macrostructure, where the participants gave a representation of the program goal. The comprehension dimension of this subtheme is functions. The different representations capture the essence of the program which they described as "*to know the gain of a salesman in different sales"* (Joyce) and *"to calculate/compute the sales and gain/income of worker*" (Peter). Farai said the goal was, *"to calculate the gain of a salesman".* He saw the goal as using the REM, *"So it's just like heading, that is REM, remark for the program. That is to compute the gain of a salesman".* In code 3, Faith's representation was "*to calculate the total score of a student in three subjects*", "*to input the subject that a student offer and the score of the student and the average total score*" (Mercy) and "*to calculate sum and average score of five students in English, Mathematics and Biology*" (Benjamin). Benjamin said, since the title of the program was not given, he traced the program to understand the code. He added, "*but, when I saw the total score, I saw SUM, and I saw average and I saw COUNTER, so I knew…the first thing that came to my mind was like…what will be the heading of the program. I was now like program to calculate the score of five students in Maths, Biology and English"*. Here, only Mercy's representation

differs. Supporting the findings, results from a final test shows that percentage correct answers of students' responses was 34% at the relational SOLO classification.

Category 3.1.3 – Relations

The *relations* require participants to explain what the program was doing by providing a line by line representation of the execution of the code in relation to the whole program. Participant representations in code 3 are described as, "*the program is calculating the Mathematics score, the English score, the Biology score and the average of the scores with the total of five students*" (Faith); "*the program calculated sum and average*" (Benjamin) and, "*the program is adding the score and the subject of the student, the average score and then introduce counter*" (Mercy). Also, in code 1, Peter said, "*the program tries to identify worker who has lesser sales and income, who has middle sales and income, and who has the highest sales and income*". Farai gave a complete representation, "*the program is calculating the gain of a salesman according to the sales made. IF the Sales is > 99,000, to calculate the gain for that, it will be 3/100 * (the specific sales made), IF Sales is > or = 69,000 but < 99,000 then the gain will be calculated as 25/100* (specific sale made) but IF Sale is greater than or equal to 25/100 * (the specific sales made)".* Joy added that "*the program allows us to know the gain of a salesman by introducing counter and the number of counts to work on*".

Subtheme 3.2: Students' affective and behavioural states during programming

This subtheme describes participants' affective and behavioural states when running a set of program codes on the computer system. These are based on observations made by the interviewer, as they ran the program, and questions asked retrospectively.

Category 3.2.1 – Affective states

They were all asked to rewrite their program using either a FOR…NEXT or a subroutine statement. The affective states centre on participants' sense of confusion when running the program, as well as how they decide to solve the problem and their recorded actions. One participant said she was confused by the new representation of line numbers in the program as well as how to write the FOR...NEXT program. She understood that line numbers are always represented in 10s. She said, "*like the one that you gave me it has 5, 10, 15 but the one of that QBASIC supposed to be 10, 20. It is stated in 10's not 5's"* (Joyce). Figure 6.25 is a sample of Joyce's program.

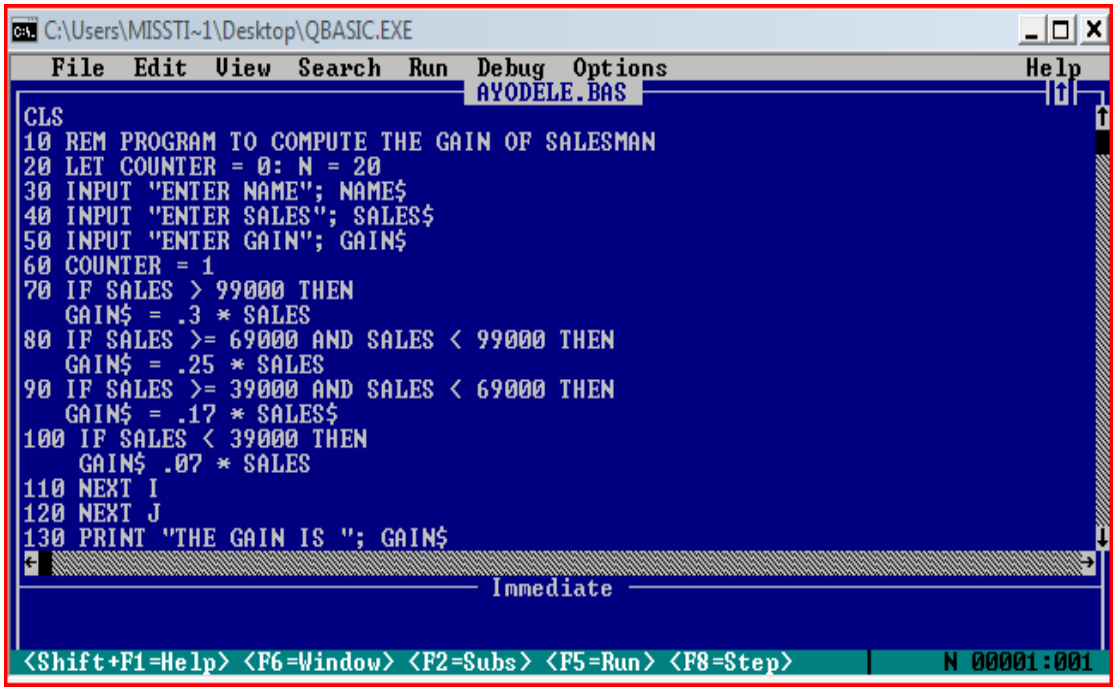


Figure 6.25: Screenshot of Joyce's program

She explained further, "s*o I don't know how am going to do it but I just use my imagination to do it. It doesn't even work, so I don't know…am sure it is wrong because I was to use FOR…NEXT statement and the program that I wrote there is no place for FOR, it is FOR I or FOR J, so I don't know where to put the…heen, I did not declare FOR, I was now like where am I going to put the FOR, maybe it is after declaring my variable or after my IF…THEN statement. I don't know where to put it exactly. So that's the big problem and I know that it is not right".* From the screenshot of Joyce's program above we can deduct that she used a FOR…NEXT statement without including the FOR aspect which starts the repetition.

Another participant was confused about how to write the program using the GOSUB statement. She said, *"when I was solving it, I think I didn't put the GOSUB where I was to put it. I was thinking should I put it here or I put it there, so I was confused*…I don't know what to do. *So, I tried correcting the program, it was still giving me the same result. So, I was looking at it what should I do. I was thinking should I do it like this or I should do it this way. The one I even wrote down in my note it was different from what I later run because the thing is not correct. The one I did in my sheet of paper it wasn't correct when I run it"* (Faith). The next screenshot shows that the participant knows everything needed in the program but could not correctly place the GOSUB to make the program work. Figure 6.26 presents a screenshot of Faith's program.

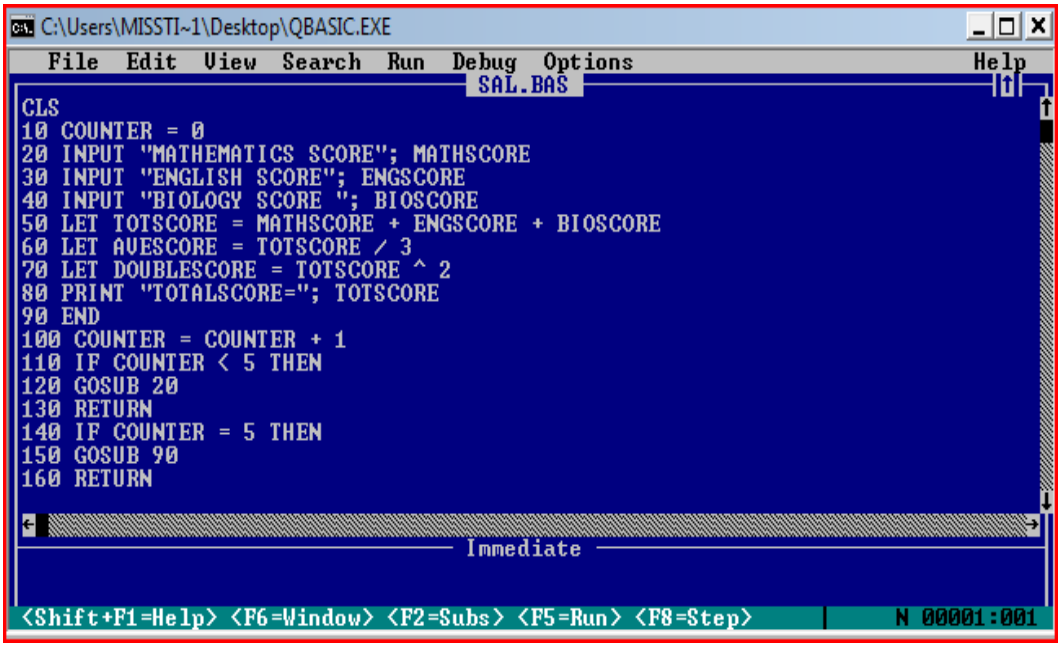


Figure 6.26: Screenshot of Faith's program

She did not state the number of the line where the GOSUB will return to. This contributed to the participant's confusion because the program was giving syntax errors. However, she experienced another syntax error on two different occasions and could identify and debug them. Peter added that while writing the program, he was *"thinking about…how to introduce the GOSUB. I want to use subroutine. I was thinking seriously, how to introduce that GOSUB to feed in, in the program. Even when I have not even started the program that is what I was thinking, that how can I introduce it…since subroutine program we are not going to make use of maybe END last, but maybe END in the second or third line, then followed by the GOSUB then the expression at the front. So that was what I was trying to introduce into the program…how do I follow these steps and get what I want. I discovered that the program keeps commenting, and you said we should use either subroutine or FOR…NEXT. I was like haha, why this thing is now giving me problem? I don't want to use another method. I want to use subroutine. I was cracking (racking) my brain seriously."* A sample of his program is represented in Figure 6.27.

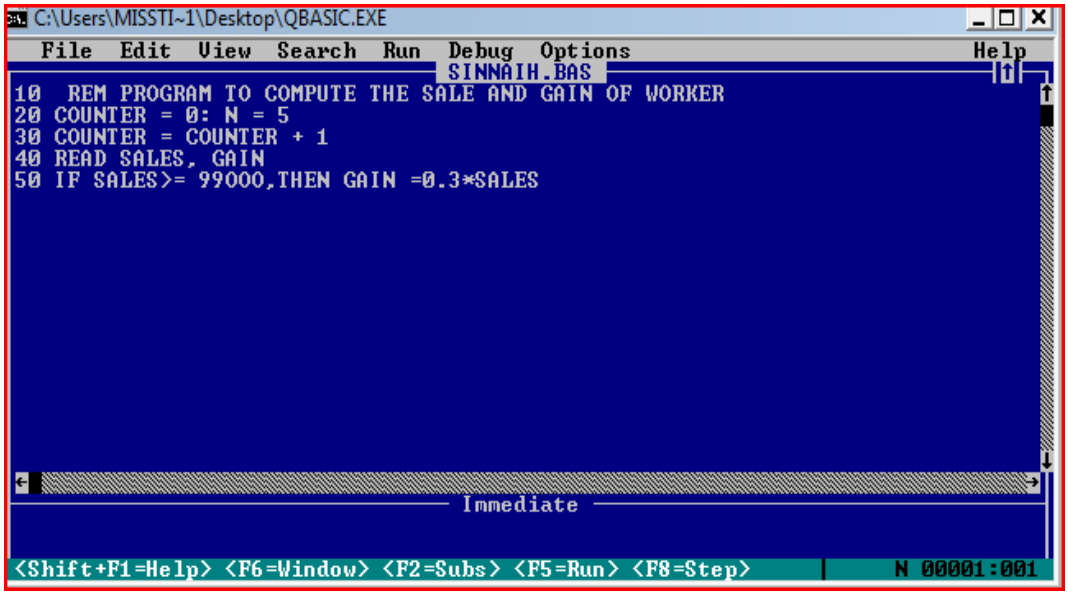


Figure 6.27: Screenshot of Peter's program

Faith and Peter's major confusion was that they did not really understand how data are executed in subroutines. Peter could not complete the program because he experienced syntax errors, while writing the program, which he could not correct.

The following discussion describes how they figured out their syntax errors during program writing. Firstly, when Benjamin was writing his program, I observed him experiencing some syntax errors which appeared continuously. He looked keenly at the syntax and tried to correct it, but to no avail. He said, "*I created variable for Maths, no error; variable for Biology, no error; and is (its) giving me error on variable for English. I don't know what led to the error. I was trying to figure it out but am not still able to…I went for help…it was telling me that I should like…not to use different something like that sha. I went back to the program, I cleaned everything on that line and rewrite it again. It was still giving me the same thing".* A screenshot of Benjamin's program is represented in Figure 6.28.

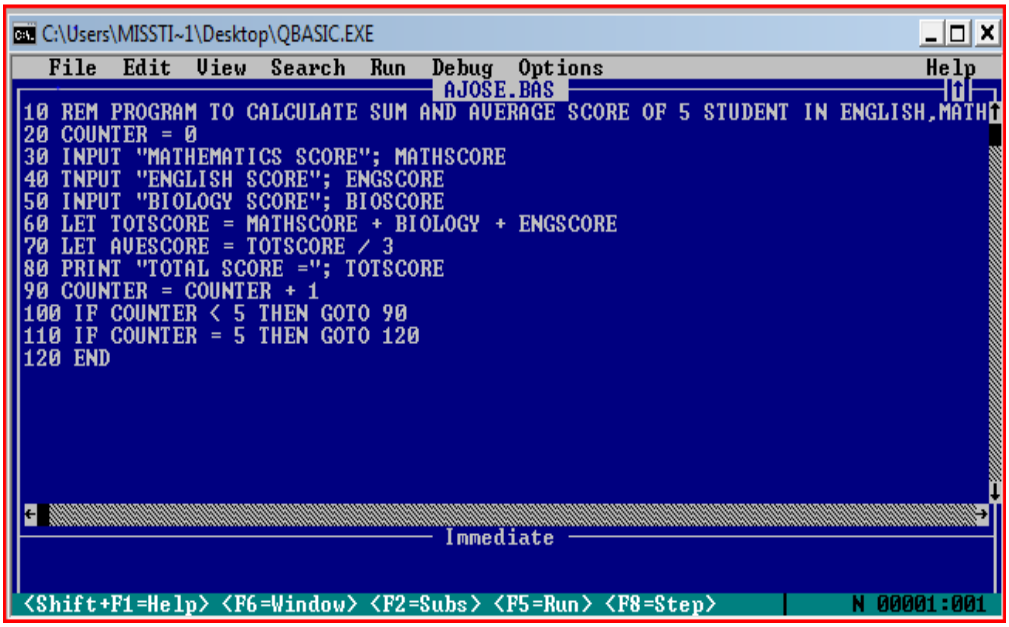


Figure 6.28: Screenshot of Benjamin's program

In line 40 of his program, Benjamin did not realise that he had typed 'TNPUT' instead of 'INPUT'. Although he used 'help' for assistance, he did not use the suggestions. Even if he had introduced the INPUT correctly, the program would run but would not have delivered the required result because of the position of the counter. It would only calculate the score of one student, instead of five. He could also not print the average.

Joy first traced the program and rearranged it correctly. Her explanation shows that she has knowledge of the purpose of the program, as well as the data and control flow. Although she experienced syntax errors, she could correct it. She said, "*I think I have problem but I figured it out...I used the LET without inputting the variable. I was supposed to input, is it score, and use the variable, so I just used the Let without using the variable'.* Also *"I read all the questions given to me, then I looked at it and figure out how to arrange it. Instead of this 10, counter = 0, it was 10 REM. It's going to have a heading like the pseudocode. Like what a task to work on. Then the input, then input the variables, then the LET. LET the Total score be the Maths score + the Biology score + the English score. Then before I print I must make sure the counter = 0. Then, maybe if it was just one person, we (are) not going to make use of counter but because it's for 5 person(s) or more than. If it were like to be 6 students it's going to be IF Counter < 6 or IF = 6".*

Another way participants addressed problems faced was, "*I tried checking what I have done and I saw that there was space between the double and the score. So, I removed the space*

in the program" and Farai, *"was looking at it, I was looking at it closely...I was reading the question…I looked at it if what I wrote made sense."*

Category 3.2.2 – Behavioural states

Behavioural state describes a participant's behaviour when testing their program. Observations made of Peter, while he was testing his program revealed, "*he was wiping his face and looking closely at the error while moving his lips. At a point he was seen supporting his waist with his two hands while looking at the error closely again. When he could not find solution to the syntax error, he shook his head, frowned his face and said 'hmmm' '*' (researcher's journal).

6.4 EXPERTS' REFLECTION ON THE TLPF

At the end of the second cycle, the final TLPF was designed (see Figure 7.2). It was then shown to experts in the field for evaluation and their reflections are reported here. The non-participant observers and one other lecturer in the department, who taught Scratch programming to first year programming students after data collection, were involved in the evaluation. I forwarded the $TLPF_3$ by e-mail to the evaluators (*q.v.* Appendix C 5) along with the description of its processes. Following this, a Google form containing three structured questions was sent to their individual e-mails. This they completed and consequently returned. From their evaluations, the non-participant observers supported the structure and design of phases of the framework with special emphasis on the design of instruction and based learning differentiation. They pinpointed that the $TLPF_3$ is suited as a framework for the teaching of programming because it is student centred. They supported the use of different teaching strategies which enable: social interaction, transfer of learning, problem solving learning strategies in the students. The evaluators therefore suggested training for teachers in the use of the framework in schools, colleges and universities. This would necessitate the incorporation of Scratch programming into the curriculum. Their responses are now presented.

What do you think about the phases of the TLPF?

Evaluator 1: *"The phases of the TLPF are well structured and organized. They accommodate common and effective learning style such as Kolb's learning style".*

Evaluator 2: "*The Phases of the Teaching and Learning Process Framework (TLPF) On the Analysis Phase, the TLPF was designed to identify the learner's previous knowledge in other to design and gathered information on the content of the instruction to learn. This phase promote effective learning style and enables cooperative learning and the students*

were motivated to learn QBASIC and Scratch Programming with the use of KLOBs learning style Instrument (KLSI) and Hermann Brain Dominance instrument (HBDI)".

Evaluator 3: *"The designed instructional model was well prepared towards teaching and learning process with comfortable for students to better understand programming language most especially for beginners using Kolbs learning style".*

What are your views about the teaching and learning strategies in the new TLPF for programming?

Evaluator 1: "*TLPF for programming is a welcomed technique. It simplifies programming learning skills. It also promotes peer-learning. It enables the students to learn programming by social interaction (students relate with one and another to solve classwork as groups, to assist the weak ones) and problem-solving strategies*".

Evaluator 2: *"DEVELOPMENTAL PHASE, The TLPF techniques for learning QBASIC and Scratch programming simplify the learning of programming. It also promotes peer tutoring and self-specialization among students. The developmental phase of this technique gives room for identification of individual learning styles and also allows students to discover things on their own and enables them to transfer acquired knowledge to solve problems in other areas of learning. The TLPF also enables the researcher to know the various kinds of methodology to be adopted during the teaching of QBASIC and Scratch programming at various interval for proper understanding of this language by students and it is students' centered approach because students were allow to do most of this things after proper guidance in class. The students' were evaluated using SOLO and Bloom's taxonomy method which include: project work, test, assignment, quize competition etc which arouse the interest of the learners".*

Evaluator 3: *"TLPF is well organized technique for programming which simplifies learning skills. It promotes collaborative learning among the students which enables the students to learn programming by social interaction and problem-solving techniques".*

What value do you think the TLPF will contribute to the teaching of programming in Nigerian colleges of education?

Evaluator 1: *"TLPF will contribute positively to the teaching of programming in Nigerian Colleges of Education, if seminars/workshops are periodically organized for the teachers in order to equip them with the new techniques before implementation"*.

Evaluator 2: *"If the TLPF method is well utilize in the Colleges of Education; it will aid learning and encourage learning of programming language among the students. Learning of this programming will also be of benefits to students if the Learning Curriculum of individual Country incorporates the programming language from the foundation this will*

enable them to have full knowledge of this language and students will be motivated to learn this program".

Evaluator 3: *"In Nigeria Colleges of education, TLPC will contribute immensely to teaching of programming to beginners, for instance, programming in Scratch, which requires interactive process with new techniques not applicable in other programming languages. Therefore, seminars and workshops can be organized for teachers teaching programming to better equip them".*

From the experts' reflections on the framework, it is clear that they are willing to use the framework to designing instruction for the teaching of programming. This calls for professional development in the next cycle of the action research study. Although it is *not* possible here because of time and space constraints, I do intend to advance this research.

6.5 META REFLECTION

This section focuses on my reflections on the second cycle action research and how this cycle has contributed to my continued professional development and teaching practice. The reflective practice of Schon (1983) for continuous professional development guided this cycle. These reflections were done by critically questioning my actions to thus improve my practice. As a result, in this cycle, I gained knowledge about myself, my students and facilitating learning within whole brain and constructivist teaching environment. I will firstly address my strengths and learning gained and secondly my weaknesses and areas where further improvement is needed. This will be done in relation to my students' and the non-participant observer's feedback on my teaching as well as my personal reflection.

I discovered that I maintained my strengths, as discussed in the first cycle, when facilitating instruction within the whole brain and constructivist environment (*q.v.* section 5.4). Concerning my learning journey, I gained more insight into the designing of tests using the revised Bloom's taxonomy and classification of students' responses to questions on terms of SOLO. Using the revised Bloom's and SOLO taxonomies for designing assessments, I learned that not only did I need to think critically about the question/s but that I also needed to be aware of the fact that a question can belong to two, or more, Bloom's classifications, depending on the assessment goal. Also, students' answers must be well studied to identify what they view as *simple* and *difficult*.

Planning and facilitating instruction, based on the whole brain, was not difficult for me. One thing I did however learn in this cycle was that my students did not really enjoy working in

groups. They prefer working in pairs. I realised that group work should not be used continuously as students become discouraged when it is overused. In this cycle I also realised that students wanted to explore, do research and be in control of their learning. The reflective feedback from my students in this cycle showed another aspect of my teaching which I had not realised in the first cycle. They see me as a teacher who explains herself in clear terms. Although students liked it when learning was facilitated to encourage the construction of meaning, they preferred that I give clear explanations.

I improved weaknesses, identified in the first cycle, by ensuring that I always linked each learning activity to real-life situations. This was noted by both my students and the non-participant observer. I also ensured that I involved the introverts in class discussions. The students expected that I maintain eye contact with them. Time management was not mentioned by the students, or the non-participant observer, but I strove to work within the limits of the lesson. I planned assessment questions based on the Block model before tests were administered. This was done to monitor learning progression on the blocks and to identify whether the tests were designed to ensure cross-referencing. There was still no cross-referencing at the end of the final test because the 'blocks level' questions were sparsely included in the planning of assessment. However, there was an improvement when compared to the first cycle design.

The insights gained regarding my students in this cycle correlated to what I had learned in the first cycle. They liked group projects but disliked it when some members of the group did not cooperate. They preferred to do practical work and developed an interest in Scratch programming. However, they felt the teaching of QBASIC programming should be the focus of the class since the exam will be based on QBASIC. Their reactions to the use of technology (computers) for practical purposes and paper-based programming made me realise that they are motivated by learning programming on the computer.

Finally, I realised that lecturers, in the context, supported the use of the TLPF for the teaching of programming. The is a need for the professional development of lecturers and this can be extended to other similar contexts in Nigeria.

6.5 CHAPTER SUMMARY

This chapter discussed the research results, research findings and meta reflections for the second cycle action research. The research results detailed data gathered during the *act* phase of the study. Observations made based on the data, including positive and negative accounts experienced during teaching, were also detailed. The findings section is a

reflection on the research results with supporting evidence from the data. I further discussed experts' review of the TLPF, as well as a meta reflection on the second phase of the teaching and learning of programming.

CHAPTER 7: LITERATURE CONTROL, RECOMMENDATIONS AND CONCLUSION

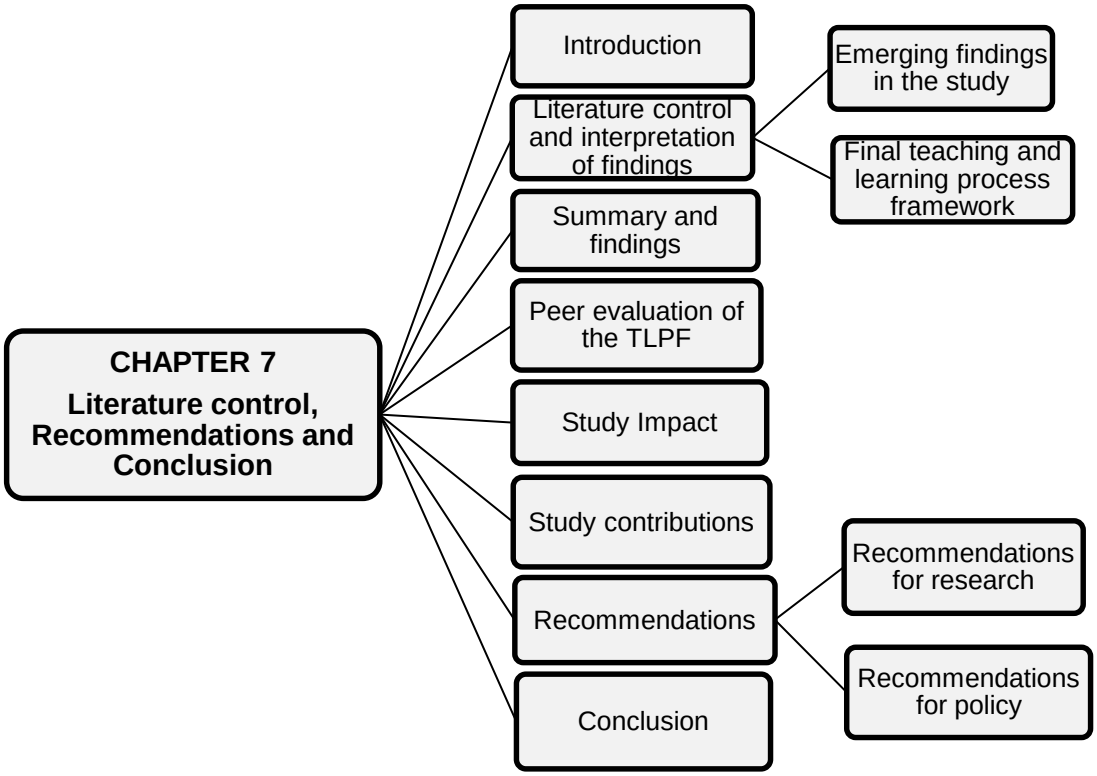


Figure 7.1 : Schematic representation of Chapter 7

7.1 INTRODUCTION

The purpose of this study was to determine how the integration of a visual programming environment informs the design of a new teaching and learning framework for programming in Nigeria. Chapters 5 and 6 focused on the results and findings from the first and second cycle action research. This chapter will synthesise the findings which emerged from the previous two chapters. I further integrated and interpreted the findings in relation to existing literature (literature control). This was done in an effort to answer the research questions and situate the study within the teaching and learning process framework (*q.v.* Chapter 3). Furthermore, I present the conclusion that emanated from the study, study implications and possible limitations of the research study. I conclude the chapter with discussions on the policy recommendations and recommendations for further research. This chapter commences with literature control and interpretation of findings.

7.2 LITERATURE CONTROL AND INTERPRETATION OF FINDINGS

I interpret the findings that emanated from the two cycles (see Table 7.1) towards providing answers to the research questions. Table 7.1 presents the themes, subthemes and

categories that helped in understanding and interpreting the secondary research questions (SRQ2, SRQ3 and SRQ4). Findings from the first secondary research question (SRQ1) are *not* represented on the table because they were not obtained through instruments. Details of the findings can be found in sections 5.2.1 and 6.2.1.

This section commences with a discussion of the secondary research questions to understand how the primary research question solved the problems researched. The research questions which guided the study are restated below. The primary research question asked: *How can the implementation of a visual programming environment inform the design of a new teaching and learning framework and as well support the learning of procedural programming skills in such a manner that teaching intervention planned upon students' learning approaches is positively experienced while promoting mental representations of the Block model?*

The secondary research questions, which answer the primary question, are:

Primary research question

The proposed study was guided by the following primary research question: *How can the implementation of a visual programming environment inform the design of a new teaching and learning framework and as well support the learning of procedural programming skills in such a manner that teaching intervention planned upon students' learning approaches is positively experienced while promoting mental representations of the Block model in first year college students?*

1.2.3.2 Secondary research questions

To investigate the primary research question, the following secondary research questions (SRQ) were addressed:

SRQ1: How do first year college students in the procedural programming classroom approach learning?

SRQ2: How do the participating students experience the teaching intervention designed based on their learning approaches with a view to promote the learning of procedural programming?

SRQ3 How does the visual programming environment (VPE) support learning of procedural programming in first year college students?

SRQ4 What are the mental representations of the Block model held by first year college students in the procedural programming classroom?

It should be noted that other findings also emerged from the study (see section 7.3). These findings do not directly support the questions guiding the study, but they are important in

that they may influence the TLPF. These findings are presented in blue (see Table 7.1) to differentiate them from the main findings of the study which are in black. The findings are discussed in the following sections.

Table 7.1: Findings from the AR cycles

Themes	Subthemes	Categories	
		AR cycle 1	**AR cycle 2**
Theme 1 **Strategies for teaching and learning programming (SRQ2)**	*Subtheme 1.1* Teaching strategies	Whole brain teaching	Whole brain teaching
		Constructivist teaching strategies	Constructivist teaching strategies
		Grouping benefits and negative impact on learning	Grouping benefits and negative impact on learning
		-	Students' perspectives and preferences for pair programming
		Teacher's personality	Teacher's personality
	Subtheme 1.2 Strategies students used for the leaning of programming	Cognitive learning strategies	Cognitive learning strategies
		Self-regulation of learning	Self-regulation of learning
		Social interaction	Social interaction of learning
		Problem solving strategies	Problem solving strategies
Theme 2 **Programming knowledge gained by students (SRQ3)**	*Subtheme 2.1* Programming knowledge gained by students	Perspectives about Scratch and QBASIC programming	Perspectives about Scratch and QBASIC programming
		Knowledge gained in Scratch and QBASIC programming	Knowledge gained in Scratch and QBASIC programming
		-	Previous experience aided programming knowledge gained
		-	Debugging skills
		Word processing skills	Word processing skills
	Subtheme 2.2 Impact of programming knowledge on students	Scratch boosts interest	Scratch boosts interest
		Programming impact on students	Programming impact on students
		-	Programming aids critical thinking
		-	Scratch builds program writing skills
		Scratch inclusion to the curriculum	Scratch inclusion to the curriculum
	Subtheme 2.3 Impact of student factors on the programming knowledge	Students' strengths and challenges faced in programming	Students' strengths and challenges faced
		Student behavior	Student behavior
		Student well being	Student well being
	Subtheme 2.4	Contextual problems	Contextual problems

	Contextual problems as impediment to the teaching and learning of programming	Suggested solution to contextual problem	Suggested solution to contextual problem
Theme 3 **Mental representation and affective states during programming (SRQ4)**	*Subtheme 3.1* Mental representation of the Block model	Atoms	Atoms
		Macro-structure	Macro-structure
		Blocks	Blocks
		-	Relations
	Subtheme 3.1 Students' affective and behavioural states during programming writing	Affective states	Affective states
		Behavioural states	Behavioural states

7.2.1 Research questions revisited

This section provides explanations to help solve the research questions posed in Chapter 1.

Research Question 1 - *How do first year college students in procedural programming classrooms approach learning?*

This research question investigated the learning approaches of students in the procedural programming classroom. The KLSI, simulated and validated HBDI instruments were used to determine students' learning approaches and establish findings. The rationale for using these instruments has been explained (*q.v.* sections 5.2.1 and 6.2.1). The findings are discussed based on the AR cycles. The breakdown of the analysis is represented in Table 7.2.

Table 7.2: Students' learning approaches for the AR cycles

AR cycle 1	KLSI indicators						Composite HBDI quadrants			
	Assimilating	Diverging	Converging	Accommodating			A	B	C	D
	27%	64%	4%	0%			48%	57%	52%	43%
AR cycle 2	Simulated HBDI quadrants						Composite HBDI quadrants			
	A	B	C	D	AC	AD	A	B	C	D
	19%	28%	38%	9%	3%	3%	77%	80%	77%	69%

Findings from AR Cycle 1 revealed that there were diverging (64%), assimilating (27%) and converging (4%) students in the programming classroom. This finding supports previous research studies by Campbell and Johnstone (2010) and Shaw (2012). This means that the programming classroom comprised diverse categories of students with *divergers* dominating the class. Therefore, first year programming students, who are *divergers,*

approached the learning of programming through group activities, class and group discussions and they received prompt feedback on class activities and assignments from the facilitator (Kolb and Kolb, 2009). The *assimilators* approached learning by responding to lectures on programming contents, reflecting on lessons taught, exploring programming content through practical sessions and responding to analytical ideas about programming problems. To accommodate the *converger*, programming content was offered through laboratory programming practicals, done during classwork and as assignments, and by trying new ideas. Although these findings did not record any data for *accommodating* learners, which may be due to the small number of students in the study sample, it still suggests that the programming classroom consists of students that approach the learning of programming differently.

Further findings show that the average composition of students' learning approaches in the procedural programming classroom, based on the HBDI, was B (57%), C (52%), A (48%) and D (43%). This finding seems to support previous studies, such as that of Bawaneh et al. (2011a) and le Roux (2011). They found that students were proportionately distributed to learning approaches in the four quadrants. In this study, however, quadrants B and C were well represented in the programming classroom. This implies that students in the programming classroom, who are majorly dominant in *B,* prefers learning that includes: organised, sequential, planned and detailed explanations of programming contents. *C* learns best through: interpersonal, emotional, feeling based and kinesthetic activities. These findings contradict the general belief in literature that programming students are purely left brained. Although there was a slight difference between the percentages of the A and C quadrants, bigger samples could produce better results.

In the second cycle, where the simulated HBDI instrument was used, findings show that the programming students were majorly dominant in B and C quadrants, confirming the results discussed above. Further analysis revealed that the percentage of quadrants represented in the programming classroom, based on the sample used, were: A (77%), B (80%), and C (77%) and D (69%). These findings show that there was a disparity between the simulated and validated HDBI in all the percentage scores of students' learning approaches. One possible reason is that the sample used for both instruments differed. In summary, one important point regarding students' approaches to learning is that B and C quadrants is overly represented in the two action research cycles. This means that programming students prefer instruction that involves detailed and sequential programming activities that are well planned and organised, with lessons designed to foster social interaction via interpersonal expressions, kinesthetic movements and emotional influences.

Research Question 2 - ***How do the participating students experience the teaching intervention designed based on their learning approaches with a view to promote the learning of procedural programming?***

To fully understand how students experienced the teaching intervention with a view to promote the learning of procedural programming, an extensive description is given. Findings that supports this question are teaching strategies and strategies students used for learning programming (Theme 1, *q.v.* Table 7.1). Table 7.3 presents a synopsis of the subtheme with corresponding categories, summary of findings and representative references supporting the findings.

Table 7.3: Findings supporting teaching and learning strategies

Findings	***Summary of findings***	***Representative references supporting findings***
Subtheme 1.1 – Teaching strategies		
Categories 1.1.1- Whole brain teaching strategies	Students had both peer and self-knowledge. They also learned in different ways from their peers in each quadrant group. This enabled them to move out of their comfort zones and also promoted positive attitudes to the learning of programming.	De Boer et al. (2013) and Rogers (2009)
1.1.2- Constructivist teaching strategies	Different constructivist strategies used for teaching fostered student engagement. However, grouping reduced students' participation in groups to confusion, lack of cooperation and conflict in groups. Students preferred pair programming to programming in groups because it enhanced cooperation, respect and taking responsibility for successes and failures.	Exeter et al. (2010); Schunk (2014); Micari and Pazos (2014) and Braught et al. (2011)
Subtheme 1.2 – Strategies students used for the leaning of programming		
1.2.1- Cognitive learning strategies	Students learned programming through assimilation and accommodation which they built upon as they learned new programing concepts and solved problems towards the construction of knowledge. This happened as students experienced disequilibrium.	Ma et al. (2007)
1.2.2- Self-regulation of learning	Self-regulation of learning was achieved using different metacognitive and cognitive strategies to monitor, control and regulate leaning towards the understanding of programming.	Fischer and Wallace (2016)

1.2.3-	Social interaction during learning	Meaning was constructed through interactions in groups. There was shared understanding among students. Introverts developed explaining skills.	Denton and McKinney (2006)
1.2.4-	Problem-solving strategies	Students gained self-confidence. They could solve programming problems in groups, through brainstorming, the use of analogies, arguing and devising group dynamics. Although some students could not solve some difficult problems individually after leaving the group.	Hmelo-Silver (2004)

At any level of education, much attention is lavished on the complexities of methods involved in teaching and learning, as well as assessment. Learning *can* occur without teaching, but if teaching is not honed, it is possible that teachers teach *without* making learning occur (Nilson, 2016). The use of different *teaching strategies* was informed by findings generated from students' learning approaches and this was used to design a whole brain teaching intervention (De Boer et al., 2013). Using the whole brain teaching model, the students learned programming in diverse ways. At the time of this research, whole brain teaching had not yet been used in the context of programming, but in other areas such as information literacy (De Boer et al., 2013), physics (Bawaneh et al., 2011b) and mathematics (Özgen et al., 2011). The use of *whole brain teaching strategies* enabled participants to obtain knowledge of themselves and their peers (le Roux, 2011). It also enabled students to move out of their comfort zones and to learn from their peers (De Boer, Bothma, & du Toit, 2011). In an effort to create a holistic and student-centred programming approach, *constructivist teaching* was combined with whole brain teaching. Findings revealed that studying programming within a constructivist environment was facilitated by strategies such as: cooperative learning, scaffolding, pair programming, research, project works and group learning. In addition, teaching with real-life application, teacher feedback, practical works, and self-paced learning were used. Combinations of these teaching strategies, therefore, promoted *student engagement* in programming. The use of *grouping* in the computer programming classroom resulted in both positive and negative effects on learning. While some students prefer its use, other students noted that it reduced their full participation and brought about confusion in groups (le Roux, 2011). A more pleasing teaching strategy, preferred by students, was *pair programming*. Reasons for this preference included: its strength in enhancing cooperation, responsibility for successes and failures and respect for individual opinions.

Whole brain teaching strategies

Using *whole brain teaching* entails a differentiation in learning accompanied with appropriate teaching methods that cater for students' differing learning approaches (De Boer et al., 2011) in the procedural programming classroom. For example, project presentation infused the class with life and interest and therefore aided in the understanding of data types in QBASIC. Through the poem, the use of whole brain teaching aided schema activation and the understanding of algorithm steps. Allowing students to do research afforded them the opportunity to explore programming topics from their perspective and to construct meaning through collaboration. A connection exists between teaching strategies and students' learning approaches (Rogers, 2009). Therefore, differentiating learning and thus importing this connection into the programming classroom, promotes positive attitudes towards the course material (le Roux, 2011). This is important because, as previously discussed, programming has high attritions rates and generally evokes negative attitudes in students. The teaching of programming, using *quadrant grouping,* takes into account students' learning approaches whilst facilitating their interaction with and learning from other group members (Coffield et al., 2004). The students also gain *knowledge of themselves* and their *group characteristics* in relation to the strengths and weaknesses they possess in their approach to learning. There seems to be no literature supporting this finding.

Constructivist teaching strategies

Constructivist teaching strategies, informed by the constructivist theory of learning, was used to facilitate programming as supported in literature (*q.v.* section 3.2.2). Teaching strategies used within a constructivist programming classroom include: cooperative learning, pair programming, scaffolding and group learning. Other strategies used include: teacher feedback, real-life application of learning and the use of research. *Cooperative learning* methods, such as think-pair-share and jigsaw (*q.v.* section 3.2.2), used to facilitate classroom activities, promote high levels of student engagement during programming activities enabling students to actively engage in the classroom. Engagement entails doing both individual and group work, the assessment of classwork and becoming involved in classroom discussions (Exeter et al., 2010). This study's findings are also in line with suggestions from literature to encourage active engagement of students in intellectual activities, collaboration and reflection on learning with metacognitive awareness of the learning experience (Gebre et al., 2014; Herrmann, 2013). Cooperative learning also enabled the students to become involved in decision-making and mental reasoning in the groups (Falkner and Palmer, 2009). Although cooperative learning was effective, the diverse composition of groups which contained different learning approaches, proved to distract and confuse some members, ultimately resulting in group conflict and a lack of

cooperation. This impacted negatively on some students' learning of programming concepts such as algorithm and pseudocode. These findings corroborate with existing literature stating that cooperative grouping does not always benefit all students (Micari and Pazos, 2014). No wonder Peter concluded that, *"Scratch programming is ineffective among group members".*

Furthermore, findings showed that *pair programming:* encouraged freedom of expression (Simon and Hanks, 2008b), fostered responsibility for actions and facilitated concentration and cooperation during programming (Braught et al., 2011). As a result, students preferred pair programming, because it built confidence during individual programming activities (Braught et al., 2011; Faja, 2014; Simon and Hanks, 2008b). Although pair programming provided these benefits, programming for understanding in pairs depends on the individuals involved. The behaviour of an unfocused partner can frustrate the efforts of the serious partner. This behaviour was not explicitly identified by participants, but it could be a topic for discussion in a further study. Cao and Xu (2005) ascertained that a student who is *not* performing well will agree with the decisions of a better performing student *without* weighing said decisions. In addition, Begel and Nagappan (2008) in their study *Pair programming: What is in it for me?* ascertained that even amongst experienced programmers, problems associated with pair programming include: disagreement, intrusion and personality differences, to name but a few.

Another finding from the constructivist teaching strategy shows that *scaffolding* was extensively used by the teacher and sometimes by the students (Schunk, 2014). Scaffolding helped to guide students during programming activities. This guidance was reduced in as much as the teacher ascertained that the students could solve problems on their own. Therefore, students could understand and complete some programming activities independently of the teacher, or other students, because of the scaffolding received. Therefore, some students reached the ZPD, while others could extend their ZPD during Scratch programming. Only a few students could move beyond the ZPD in procedural programming. This finding partially supports Awbi et al. (2015) who noted that scaffolding, when appropriately given, can extend students' ZPD.

R*eal-life examples* were used during programming teaching. Students, it was found, embraced the use of real-life activities in solving programming problems, especially in algorithm and decision making with repetition (*q.v.* section 6.3.3). The continuous use of *research activities* as a teaching strategy was emphasised by the students. These research activities inspired students to explore, interact and learn programming beyond the reach of

the classroom. *Assignments* were extensively used to keep students engaged and committed to the learning of programming, beyond the classroom. Findings revealed that students detested assignments when overused. Other explanations could be the high number of courses taken by students, each requiring a varying number of assignments. *Individual learning* and *practical class-work* created opportunities for students to take control of their learning. These findings, which align with the work of Schunk (2014), held certain practical implications for the use of the constructivist learning theory. *Teacher feedback* acquainted students with their progress and facilitated further decision making regarding learning. There seems to be limited literature on the use of these strategies in programming classrooms.

Cognitive learning strategies

Schmeck (2013) noted that learning strategies are series of procedures, or cognitive skills, which are used to accomplish learning. Programming itself involves several cognitive activities, where in the students plan solutions and engage in problem solving activities. The students learned programming through *assimilation* and *accommodation* (Pritchard, 2013) which they built upon as they learned new programing concepts and solved problems toward the construction of knowledge. The knowledge constructed by the students was aided through internal cognitive conflicts as regards the programming concepts learned. These findings corroborate those of Schunk (2008) and Ma et al. (2007). As stated by Piaget and Papert, the teaching-learning process must be organised in such a way as to allow for the construction of knowledge obtained through experience.

Self-regulation of learning

Self-regulated learning is the regulation of thoughts, feelings and actions by students, which relate to the attainment of goals (Schunk, 2014). Findings reveal that students regulated their learning by using different metacognitive and cognitive strategies to monitor, control and regulate their leaning towards understanding the programming course during the semester. They attached value to the learning of programming by challenging themselves to do more complex programming tasks and organised group tutorials without receiving a directive from the facilitator. They exhibited a focused goal attainment, sought help, transferred knowledge to other domains of learning, maintained a positive emotional outlook and displayed a multidimensional view of learning programming (Aleven, Stahl, Schworm, Fischer, & Wallace, 2003; Chang, 2005). The existence of a relationship between self-regulation and students' performance in programming has been ascertained (Bergin, Reilly, & Traynor, 2005). Even though *this* study did not examine such relationships, the findings

show that the self-regulatory strategies which students engaged in, augmented their learning of programming.

Social interaction during learning

Learning, through programming in pairs and cooperative groups, was encouraged during classroom teaching. As a result, students learned and constructed meaning in programming through social interaction/s. They discussed their learning during classroom interaction/s, solved problems together and shared understanding. Students noted that this process impacted on knowledge construction in programming which, without said interaction, they would *not* have understood. Social interaction fostered an enquiring and reflective state of mind in the students which improved introverts' level of interaction. Findings from this study align with the social constructivist view of learning (Amineh and Asl, 2015) and support previous findings of (King, 1990; McKinney and Denton, 2006b).

Problem solving strategies

Jonassen (2011a) asserts that problem solving is a complex activity which is schema-based. It requires students to continuously practice their skills within the learning domain. Findings from this study show that, due to different programming problems which students were exposed to, some of them gained the self-confidence to solve programming problems individually. They constructed knowledge and computationally solved problems through visual representations in Scratch and transferred knowledge to solving problems in QBASIC programming. These findings support the work of Grover, Pea, and Cooper (2015) and Fee and Holland-Minkley (2010). Students also developed problem-solving strategies such as brainstorming, the use of analogies, arguing and devising group dynamics to solve programming problems (Hazzan et al., 2015). Hmelo-Silver (2004) explained that as students discuss work in groups, *knowledge* is constructed through the activation of prior information and the processing of new data. However, some students could not express their problem-solving ability on complex problems after leaving the group. Many factors may have contributed to this. For example, (Jonassen, 2011a) noted that a students' problem-solving ability may be affected by: prior knowledge, previous experiences of problems, cognitive skills and epistemological knowledge.

Research Question 3 - ***How does the visual programming environment (VPE) support learning of procedural programming in first year college students?***

This research question aims to explain the possibilities of Scratch programming for improving students' learning of procedural programming. The findings for this research question relate to programming knowledge gained by students and, as such, impacts on

programming (Theme 2, *q.v.* subthemes 2.1 to 2.3, Table 7.1). Table 7.4 presents a summary of the subthemes answering the research question with references supporting the findings.

Table 7.4: Findings supporting how VPE supported the learning of procedural programming

Findings	**Summary of findings**	***Representative references supporting findings***
Subtheme 2.1 – *Programming knowledge gained by students*		
Categories		
2.1.1- Perspectives about Scratch and QBASIC	Scratch programming was easy to learn. Students found similarities between QBASIC and Scratch programming and therefore it aided the easy understanding of QBASIC programming concepts.	Wolz et al. (2009); (Cetin, 2016)
2.1.2- Knowledge gained in programming	Students can define and identify variables for a programming problem. They could represent Scratch algorithms in QBASIC codes and understood the concept of repetition. However, they could not apply variables for designing solutions and were unable to think algorithmically when solving problems in Scratch and QBASIC programming.	(Meerbaum-Salant et al., 2013); Mladenović et al. (2017a)
2.1.3 - Debugging skills	Scratch did not aid students' debugging skills.	Weintrop and Wilensky (2015)
Subtheme 2.2 - *Impact of programming knowledge on students*		
Categories		
2.2.1- Programming aids critical thinking	Thinking about problem-solving in Scratch was not that serious, unlike QBASIC where they could think critically about a problem.	Mladenović et al. (2016)
2.2.2- Scratch builds program writing skills	Students could solve expression problems and write a program in QBASIC due to the knowledge of blocks arrangement Scratch programming.	Brennan and Resnick (2012)
2.2.3 - Scratch boosts interest	Students interest in programming were boosted due to their exposure to Scratch programming.	Ouahbi et al. (2015)
2.2.5 - Programming impact on students	Students made future career decisions in programming because of the Scratch influence. They also applied the learning of programming to their daily planning activities.	Mladenović et al. (2017b)
Subtheme 2.3 - Impact of student factors on programming knowledge		
Category		

2.3.1- Students' strengths in programming and challenges faced	Students faced challenges relating to the Scratch blocks. Some students did not like the informal introduction of Scratch.	Monig et al. (2015)

Perspectives about Scratch and QBASIC

Students experienced Scratch as *fun* and *easy to learn,* because of the interactive nature of the animations and graphical composition of the Scratch environment. This finding corroborates with Weintrop and Wilensky (2015) who found that students perceive block-based programming to be *easier* due to the simplicity of the block labels. It also supports the work of Wolz et al. (2009) which noted that transitioning from Scratch to Java, or C programming, was easier for students. Students discovered that there were *similarities* in the use of some Scratch and QBASIC concepts. They learned QBASIC repetition structures, as well as the use of variables and operators through Scratch programming. Therefore, students noted that transitioning from Scratch to QBASIC programming aided their understanding of programming concepts (Cetin, 2016). The interrelationship between the application of concepts in Scratch and QBASIC resulted in the students viewing Scratch as a *foundation* which underpins procedural programming. Scratch facilitated a background knowledge of programming concepts *before* students moved to QBASIC programming, thereby deepening their understanding. Repetition of each programming concept in Scratch and QBASIC heightened students' understanding of programming concepts. There seems to be no research work related to these findings with respect to QBASIC, but it does confirm the finding of Malan and Leitner (2007) which reported that 76% of the students rated Scratch as a positive influence when exposed to text-based programming. Scratch also *motivated* students to attend programming classes. This *confirms* Ouahbi et al. (2015) and Sáez-López et al. (2016) but *disconfirms* Martínez-Valdés, Velázquez-Iturbide, and Hijón-Neira (2017). Therefore, students' preconceived ideas about the complexity of programming were removed *after* the experience. This does not mean that students did not identify QBASIC programming as complex, but they had enough confidence to continue with programming in the future.

Knowledge gained in programming

Programming concepts such as algorithms, variables, looping and conditionals learned in Scratch provided students with a better understanding of these concepts when they transitioned to QBASIC programming (Cetin, 2016). These programming concepts supported the learning of QBASIC. Findings, based on *knowledge of variables,* reveal that students could correctly identify and define a variable for programs in QBASIC. This is

because the use of variable formation in Scratch is quite similar to text-based programming (Meerbaum-Salant et al., 2013). Scratch, however, may allow spaces in defining a variable but this does not affect the way in which students define variables in QBASIC programming. These findings disagree with previous findings of Swidan et al. (2017) which suggested that students find the use of variables challenging when transitioning to text-based programming. However, not all students applied their knowledge of variables in designing solutions and writing programs in QBASIC. The implication of these findings is that students did not relate their experiences gained in Scratch to the solutions designed in QBASIC. For a student to attain a global view of a program, they must be able to use variables and understand their application in a program (Lister et al., 2010). Teachers can use the techniques of *bridging* and *hugging* to bring about a mediated transfer from the two programming languages, as discussed by (Mladenović et al., 2016).

In designing solutions to answers, knowledge gained on *algorithm* in the Scratch classroom was applied during problem-solving in QBASIC (*q.v.* section 5.2.3) with the guidance of the teacher. This was achieved, having been emphasised, in both Scratch *and* QBASIC class. Learning algorithms for solution development consistently in the two classes deepened students' understanding. This was revealed in their ability to develop a QBASIC algorithm from a Scratch script. Regrettably, students did not always apply algorithmic techniques in designing solutions to problems in individual, or group assignments. These findings confirmed the findings of Meerbaum-Salant et al. (2011) who found that students solve problems in Scratch by dragging blocks that fit together to form a script without thinking algorithmically. This is a common habit in Scratch and is termed *bricolage*. It is not surprising that students transferred this habit to QBASIC programming. However, for students to program well they need to think systematically (Kafai and Burke, 2013).

The understanding gained of the concept of looping or *repetition* learned in Scratch programming using the REPEAT, FOREVER and FOREVER IF enabled students to identify, interpret and apply repetition structures correctly in QBASIC. The findings of this study confirmed the study of Mladenović et al. (2017a) and (Lewis, 2010). However, most students' programs were written with teacher guidance. On *control structures*, findings show that majority of students did not understand the concept and therefore could *not* conveniently solve problems that required decision making in QBASIC. One reason may be the limited emphasis afforded the concept because of time constraints.

Debugging skills

Debugging was not an issue in Scratch programming because Scratch helped students to address their errors. Students detect errors in Scratch when the wrong blocks are dragged to design a program as they do not *snap* together. It has been established that Scratch reduces the cognitive load found in text-based programming (Cetin, 2016). Therefore, students could focus on the algorithm instead of learning syntax. However, students tended to debug based on trial and error with an understanding that *unsnapped* blocks depict an incorrect code. This finding corroborates with Weintrop and Wilensky (2015) who used Snap! (a variant of Scratch) with high school students. This habit was transferred to QBASIC programming as students' debug identified program error based on trial and error without reflecting and thinking about the cause of the problem. Therefore, Scratch did not help students to debug in QBASIC programming.

Programming aids critical thinking

Programming in Scratch helped students to *critically think* about programming solutions and thus carefully arrange program blocks through dragging and dropping them. This action in Scratch helped them to carefully consider the arrangement of program codes towards the desired result in QBASIC (Sáez-López et al., 2016). However, according to students, programming in Scratch does not require serious thinking. With Scratch, students were unable to fully express their ideas because it involved dragging predesigned blocks, but with QBASIC they could express their thinking and ideas well. This finding confirms a previous study by Mladenović et al. (2016) which stated that students found the Scratch environment *limiting* after having been exposed to programming. This also supports the study of Weintrop and Wilensky (2015), although their work focuses on Snap!.

Scratch builds program writing skills

Scratch helped the students to gain and apply knowledge regarding the *arrangement of blocks* to QBASIC (*q.v.* subtheme 3, sections 5.3.2 and 6.3.3). Students attributed the orderly arrangement of blocks in Scratch programming to the arrangement of program codes in QBASIC during program writing. According to the findings, incorrectly placed blocks render wrong results in Scratch and wrong outputs in QBASIC. There seems to be no research report that supports this finding. One term in literature which does, however, relate to this finding is *sequencing*. Bers, Flannery, Kazakoff, and Sullivan (2014) explained sequencing as a form of planning involved when arranging program codes or blocks in an orderly fashion to achieve the desired result. Brennan and Resnick (2012) identified *sequences* as one of the computational thinking concepts in Scratch because programs written in Scratch are written in *sequences* rather than in a *procedural* manner. Therefore,

students could easily transfer programming sequences learnt in Scratch to QBASIC programming since programs written in QBASIC are also written in sequences. However, this bottom-up approach to programming should *not* be encouraged (Meerbaum-Salant et al., 2011). Students related *knowledge of expression* gained in Scratch programming to BEDMAS in mathematics, which was applied in solving simple mathematical expressions in QBASIC (Brennan & Resnick, 2012).

Scratch programming boosts interest

Scratch programming was found to increase students' interest in programming. Students' first attempt at programming in Scratch resulted in an enthusiastic approach to programming. Students affirmed that QBASIC class would have been easy to understand, but exposure to learning concepts through graphical images boosted their interest in QBASIC. These findings support the study of Ouahbi et al. (2015). In addition, they see Scratch as more interesting than QBASIC.

Programming impacts on students

Students' experiences in the programming classroom led them to conclude that programming is worthwhile. Some of them even indicated that they believed it would be easy to become a programmer. This finding supports Mladenović et al. (2017b). Therefore, the simplicity of programming with Scratch motivated students to view programming as a possible career option. In addition, the experiences gained in programming helped them to plan and structure their daily lives and activities relating to other courses of study in the college.

Students' strengths in programming and challenges faced

Despite the positive contribution of Scratch to QBASIC programming, students experienced challenges relating to: the difficult location of blocks and costumes, long scripts producing small results, mastering of program blocks and their application in a problem. In addition, Scratch wastes time due to the frequent searching for program blocks. This confirms the study of Monig et al. (2015) who ascertained that for experienced programmers the searching and assembling of blocks waists time and interrupts their flow of thought. Although the identified challenges faced were frustrating, students claimed they learned from their experiences. In addition, because Scratch programming did not form part of the curriculum or a major course for graduation, some students did not attend Scratch class but preferred to only attend QBASIC class, which was the major course. This might have contributed to some students *not* understanding the concept of expression, as discussed

before. This finding supports the work of Martínez-Valdés et al. (2017) who claimed that an informal introduction to Scratch may be unsatisfactory to students.

Research question 4 - *What are mental representations of the Block model held by first year college students?*

This research question describes findings obtained from students' mental representation of QBASIC program code designs based on the Block model. The following explanations provide answers to the research as found in Theme 3, subtheme 3.1, *q.v.* Table 7.3.

Atom level

The first mental representation constructed by participants was the concept of a *variable*. As Jimoyiannis (2011) stated, building a correct mental representation of a variable is very important in understanding other programming constructs. It is not surprising that participants could identify and explain how they formed their representation of variables in the code at the *text surface.* For students to proceed, it is important to gain understanding at the atom level (Schulte et al., 2010). However, most participants could not differentiate a *string variable* from a *numeric variable*. This can affect how they apply variables in a program because they have not yet built a mental representation of the concept (Eckerdal et al., 2006; Jimoyiannis, 2011). It should also be noted that the tension of the interview could have contributed to participants' incorrect representations. At the *program execution* level, the participants could explain their representation of the execution of an assignment statement (*q.v.* section 5.3.2 and 6.3.2) in the program. One of the participants had a misconception as to the difference between a variable as a program element and a variable in execution. However, if a student could *not* build the local perspective needed for comprehending a variable, the holistic perspective, needed to understand the whole program, would be difficult to obtain (Sheard et al., 2008). At the *functions level*, students could explain and mentally represent QBASIC statements (INPUT, GOTO and COUNT) and explain their functions in the code. These findings are not surprising as it is expected that participants start by reading the program text in a bottom-up fashion. Knowledge gained at the atom is believed to help them in the comprehension process. Nevertheless, if a participant's representation at this stage is fragile, moving upwards the block might prove to be increasingly difficult and frustrating. One possible solution, suggested in literature, is that teachers teach novices the roles of variables (Sajaniemi, Ben-Ari, Byckling, Gerdt, & Kulikova, 2006) in programming.

Blocks level

The Blocks model requires a deeper understanding of the program constructs where the participants build a mental model of the program (Schulte et al., 2010). They explained the code and the understanding gained from their representation show that they could mentally supply a correct representation of the content IF…THEN block (*q.v.* section 5.2.3) which was syntactically and semantically built with the program codes. This confirms the argument of Mow (2008) that a novice programmer must develop a mental model of how execution takes place. A possible contribution to this could be the extensive explanation of the IF…THEN statement using analogies from real life situations and careful reading of the program before the explanation. This supports findings that reading improves a novice's understanding of an abstract concept (Busjahn and Schulte, 2013, November, 14-17; Kumar, 2013).

Macrostructure

The comprehension dimension of the macrostructure was tested at the functions level. Findings show that students formed a representation of the program goal in two ways. Firstly, by using the REM statement which helped them in constructing an understanding of the program. The majority of the participants did this on the surface without carefully studying the program to determine the goal. Therefore, representations by surface learners did not capture the essence of the code. From the findings, it can be argued that the representation these students formed depicted surface knowledge (Shi, Cui, Zhang, & Sun, 2018). Secondly, the goal was determined by first tracing the program and identifying the condition the program was meant to test (*q.v.* sections 5.3.2 and 6.3.2) to obtain a global view. This shows that they understood the program with a viable representation.

Relations

The relations describe mental representation of students' comprehension of program at the program execution. The students' representation did not capture the essence of the data and control flow of the program. Only one of the participants could give a complete presentation of the program. This means that the students could not understand the interrelationship between the sub-goals and the goals of the program. It can be interpreted that they only saw the *trees* and not the *forest* (Lister et al., 2006). The findings of this study confirm Jimoyiannis (2011) views which stated that students approach programming problems on a line to line basis and lack a detailed mental model of the program.

7.2.2 Emergent findings in the study

This section describes the new findings in the study *(q.v.* Table 7.1*)*. They are: teacher's personality, word processing skills, Scratch inclusion in the curriculum, student behaviour, contextual factor and affective and behavioural states. The *teachers' personality* and attitudes towards teaching contribute to effective teaching. The participants noted that the teacher was: passionate and committed to the teaching of programming, competent and always present in class. She also showcased good attitudes toward teaching and expressed herself well to the students. These findings corroborate with Liakopoulou (2011). He stated that a teacher's teaching strategies depends on the fusion of his/her personality traits and knowledge acquired on the subject matter. In relation to program writing, some of the students noted that they gained *word processing skills* due to constant exposure to programming practical. Even though possession of word processing skills was not emphasised in the classroom, it influenced the way in which the students programmed, especially students who *lacked* the skill. Students became confused when the wrong keys were pressed. Therefore, the students claimed that an extensive practical in word processing skills should be emphasised. Due to the experiences and knowledge gained in programming, *Scratch inclusion in the curriculum* was emphasised by the students. They noted that Scratch should be a foundation course to all programming courses in colleges of education. Implementation of the elementary aspects of Scratch programming in primary and secondary schools should also be included in the curriculum, prior to students' exposure to it in the college or university. *Student behaviour* also impacted on the learning of programming. The students exhibited *desired* and *undesired* behaviours during the programming. Desired behaviours exhibited enabled them to support their peers and address their bad behaviour. Undesired behaviours (including lateness, negligence to do assignments, sleeping and lack of respect for group members) impacted negatively on the learning of programing. It was also found that the *student well-being* in relation to their mood, personal and health issues influenced how they behaved or learned programming in the class. Because students sometimes dealt with these issues, their participation in the class was limited and this affected their understanding of a concept. Different problems (including a lack of electricity, electricity failure, faulty computers and projectors, practical issues, positioning of the whiteboard and laboratory arrangement and student enrollment) were identified as *contextual factors* which hindered the teaching and learning of programming during the semester. However, students gave suggestions as to how these problems could be solved which included provision of constant electricity, time management by lecturers, encouraging kinesthetic activities, increased practical time, new and well-resourced laboratory environment for the teaching and learning of programming. During

programming, the students exhibited both affective and behavioural states. The *affective states* of the students revealed that they were confused about the use of repetition structure FOR…NEXT, in writing a program. They did not understand the concept and continuous trials yielded no result. Added to this, the pressing of wrong keyboard functions contributed to their confusion. Other students noted that they were able to figure out solutions but that their solution to the program did not produce the needed skill. They could possibly figure out the problem through trial and error. Because of their affective states, the students developed *behavioural states* such as wiping their face or moving their lips.

7.2.3 Final teaching and Learning Process Framework

Prior to the empirical study, a teaching and learning process framework (TLPF) was designed to serve as a conceptual guide towards deepening and channelling understanding of first year college students' programming experiences from *visual* to *procedural* (*q.v.* Figure 3.10 and Figure 6.2). The final framework ($TLPF_3$), as the product of this study and focus of the primary research question, is herewith presented based on the findings from the study, as presented in the following section.

Primary research question – *How can the implementation of a visual programming environment inform the design of a new teaching and learning framework and as well support the learning of procedural programming skills in such a manner that teaching intervention planned upon students' learning approaches is positively experienced while promoting mental representations of the Block model in first year college students?*

The primary research question was discussed by elaborating on the secondary research questions and aligning each of the findings to the $TLPF_3$ (see Figure 7.2.1). The $TLPF_3$ will now be used as a guide to explain how it aligns with the findings. The $TLPF_3$ will now be presented and parallels between the different aspects of the TLPF and research themes and subthemes will be discussed.

Reflecting on the findings provided by the TLPF

Drawing from the concepts of the TLPF and the findings that emerged from the study, links were established between them to aid in the answering of the primary research question. These findings helped in the adaptation and improvement of TLPF constructs.

Learning approaches

The first question deals with students' learning. From the findings, the researcher argues that integrating a visual programming environment, as a support for the teaching programming to first year college students, required that one firstly determines students' approaches to learning. It was discovered that students in the first year programming class possessed different learning styles. Therefore, students' learning approaches (which were sorted into quadrants A, B, C and D) and termed *convergers*, *assimilators* and *divergers,* were analysed in terms of the TLPF. Although *need analysis* and *task analysis* do not appear in the findings of this study, the researcher argues that they *are* implicit practices employed by the teacher before the learner analysis are discovered and are therefore viewed as informing the new TLPF.

Teaching and learning strategies

Theme 2 addresses the second research question. The findings align with the designed TLPF. Different teaching strategies were utilised to meet students' learning approaches, as identified previously. The whole brain teaching strategy allowed for the grouping of students in accordance with their approaches to and differentiation in learning. The grouping of students, based on learning approaches, facilitated their movement beyond the scope of their normal comfort zones to learn from their peers and discover inherent knowledge about themselves. It is important to know that students learn programming to construct knowledge, therefore *constructivist* teaching strategies which encourage learning include: cooperative learning, pair programming and scaffolding. These teaching and learning strategies align with the designed TLPF and can be found in the *implement phase* of the TLPF.

Other constructivist teaching strategies used in the programming classroom are: real-life examples, research activities, assignments and project presentation. In addition, individualised learning, extensive practical and frequent teacher feedback are also used. In addition, students learned programming through: cognitive learning strategies, social interaction, problem-solving strategies and self-regulated learning. These strategies supported the teaching and learning of programming. Therefore, these teaching and learning strategies inform the design of a new TLPF for improving the teaching of procedural programming in Nigeria. It is important to note that, before the teaching strategies were implemented on the TLPF, two concepts of the TLPF (*design* and *develop*) were considered by the teacher. Parallel links to the evaluation block also existed, meaning students' knowledge was evaluated during, and after, teaching and learning took place. The researcher holds that these were essential practices that informed the use of teaching strategies in the programming classroom. Parallels are drawn to show the connection

between the teaching and learning strategies (implement), develop and design concepts of the TLPF.

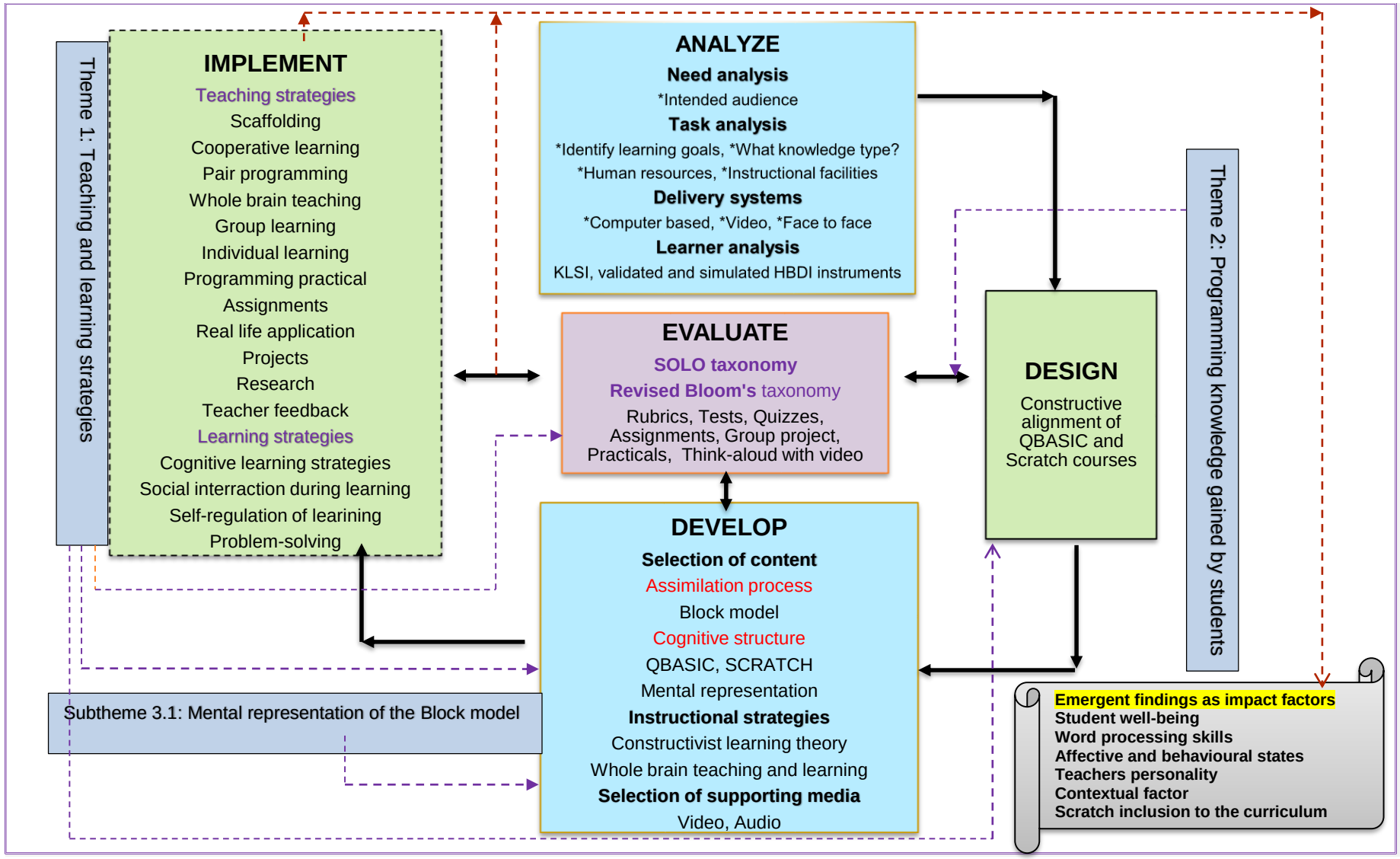


Figure 7.2: Final teaching and learning process framework ($TLPF_3$)

VPE supported learning

Theme 3 (programming knowledge gained by students) discusses the third research question. Students found Scratch to be fun, easy to learn, motivating, building interests in programming and encouraging critical thinking. Scratch supported the learning of concepts and students could confidently transition to QBASIC because of concepts learned in Scratch. Students, therefore, believed being a programmer will be easier with continuous exposure to Scratch programming. However, students are deficient in the application of variables to design solutions in QBASIC, application of algorithmic thinking during program planning in Scratch and QBASIC as well as understanding the concept of control structure. In addition, students debug in Scratch and QBASIC based on trial and error. They also faced challenges relating to *time wasting* due to the exploration of Scratch program blocks and the mastering of said blocks. The findings from this research question answered the primary research question because Scratch supported the learning of procedural programming. However, areas which students found challenging need to be revisited, while the drawbacks of Scratch need to be considered during teaching. I, therefore, argue that the findings of the study align with the TLPF. Parallels show that the theme directly connects with the *develop phase.* The develop block, in turn, connects with teaching and learning strategies (Theme 1.1) of the TLPF. In addition, the TLPF concepts connect and influence each other.

Mental representation

Theme 3 (mental representation of the Block model) provides answers to the fourth research question. Students' mental representation of the Block model knowledge dimension of atom, block, macrostructure and relation captured the essence of the codes studied. However, findings show that most students could not differentiate between a string and numeric variable at the atom level. At the macrostructure, most of the participants' mental representations of the goal of a program were done on the surface *without* tracing the code, as expected. As per relations, however, students did not have the mental model of the notional machine for the program. The understanding gained from this discussion was that the integration of a VPE only supported the learning of QBASIC programming concepts. It did not aid the students in building a concrete mental representation for QBASIC programming. Explanation/s for this could be rooted in how the teaching was structured as it did not emphasise the determination of students' representation in Scratch programming. These findings, which are thus open to further investigation, align with the conceptual framework and, as such, determines students' understanding of the developed procedural programming course. Figure 7.2 depicts parallels which point to the *develop* and *evaluation* phases of the TLPF.

7.3 SUMMARY OF FINDINGS

Findings from this study reveal that first year college students held different learning approaches and thus approached learning differently. Teaching strategies used for programming teaching promoted student engagement towards the learning of programming. Students learned based on their learning approach and moved beyond their comfort zones to learn from their peers and discover new things about themselves. Pair programming was preferred to cooperative grouping as it fostered cooperation amongst the students. Nevertheless, students could learn programming through: social interaction, self-problem-solving activities and self-regulation of learning. The visual programming environment supported the learning of procedural programming concepts. However, it did *not* help students to think algorithmically. Essentially, students dealt with programming errors based on trial and error. Students also struggled with the Scratch environment with respect to: the memorisation of blocks, the location of costumes, time used in dragging blocks and an informal introduction of Scratch. Students held a concrete representation of *some* aspects of the Block model. Representations of the program execution dimension of the *relations and functions* dimension of the macrostructure were fragile. This study also found that learning programming is dependent on the emotional state of the student. They exhibit affective and behavioural states when they do not understand programming concepts with repetitive structures. The teacher's personality indirectly influences students' commitment and motivation to learning. Contextual factors affected the teaching and learning of programming, but practical solutions to curbing problems have been identified in the study. It is important that some knowledge of word processing skills are acquired prior to the learning of programming as it influences the way in which students program. There was a strong move for the inclusion of Scratch programming in the computer science curriculum as well as in primary and second schools.

7.4 STUDY IMPACT

This study impacted significantly on study context where data was collected. Due to students' motivation and eagerness to learn programming through Scratch, it was internally decided (at a departmental level) that all first year students should be exposed to Scratch programming. It was also decided that it would be a '0' unit course for students. In effect, during the following 2017/2018 academic session, first year students were taught Scratch programming for a semester. A lecturer in the department facilitated the teaching of Scratch, although the TLPF framework was not used for this purpose. However, students were motivated to learn programming through Scratch.

7.5 STUDY CONTRIBUTIONS

This study makes three contributions to the field of programming education. These are theoretical, methodological and practical contributions, as forthwith discussed.

7.5.1 Theoretical contribution

Theoretically, this study contributes to the existing literature in four dimensions. Firstly, it adds to the existing literature on programming and program comprehension, especially in applying the Block model for designing comprehension questions in QBASIC programming. Secondly, it adds to the literature on whole brain theory which has not been used before in programming education (*q.v.* section 7.2). Thirdly, the study provides evidence and supports the body of existing research which advocates the use of constructivist teaching strategies for programming teaching (*q.v.* section 7.2). Fourthly, the final $TLPF_3$ serves as the research outcome which adds to the body of literature on instructional design model for the teaching of programming courses.

In the light of the above, the study contributes to the literature on teaching programming to first year college students in Nigeria. The study's theoretical contribution is hereby presented since less attention is afforded to the teaching of programming to first year college students in Nigeria. In response to improving both *the teaching of programming* by the teacher, and *the learning of programming* by students, the contributions made are considered applicable.

7.5.2 Methodological contribution

As for the methodological dimension, this study contributes to the field of programming education by using the practical action research strategy (involving two cycles) to study students' experiences in programming. Lecturers at both universities and colleges of education will find this methodological approach useful for studying their practice and thereby augment their professional growth. The action research plan designed for this study (Chapter 4, Figures 4.9 and 4.10) can be replicated as a methodological guide in another context. A lecturer can study his/her practice by first identifying a problem, or asset, which can then be improved upon. The first cycle (Figure 4.9) can then be used as a guide along with the $TLPF_3$ as the two work together. It would thus be implemented with the action plan. A further iteration can follow using the second cycle action plan (Figure 4.10). Depending on the outcome of the study, another action plan can be designed.

7.5.3 Practical contribution

Practically, the TLPF was designed as a framework for facilitating programming instruction at schools, colleges and universities. In using the framework, the simulated HBDI instrument (Appendix A15) will help teachers to determine students' learning approaches. The designed whole brain walk around (Chapter 5, Figure 5.7) and its implementation (Tables 5.1 and 6.1) in the programming classroom can be wholly adapted, or adopted, by teachers globally for designing programming courses which meet diverse students' need. In addition to designing programming courses, a designed constructive alignment for QBASIC and Scratch programming (Appendix B 1 and B2) can be applied to the teaching of Scratch and QBASIC programming courses in different contexts. This study also made an important contribution towards the assessment of students' learning using the combined Bloom's and SOLO taxonomies for designing and assessing formative programming test questions. It will serve as useful guide for teachers teaching QBASIC as an introductory programming course.

7.6 STUDY LIMITATION

In this section, I reflect on the limitations of the study. The research strategy I used was action research. I acknowledge the non-generalisability of action research studies and therefore the findings of this study cannot be generalised, but only replicated in another context. During the study, my role as a participant observer may have resulted in observer error and bias on the data collected in the programming classroom. To prevent observer error, I watched the recorded videos several times to fully understand the situation. Further reflection on the observation/s helped me to interpret the situation better. I also cross-checked the recorded observations with the participants, non-participant observer and the whole brain feedback form from both students and observers to prevent observer bias. I acknowledge that I might have been biased in the selection of students for the interview since the selection was based on those who had completed the HBDI instrument. The interview was voluntary.

7.7 RECOMMENDATION

Due to the complex nature of programming, recommendations are made for similar contexts like the one used in this study. The recommendations are provided in line with research and policy.

7.7.1 Recommendation for research

- *Research on the $TLPF_3$:* As an important product of this study, I recommend the new Teaching and Learning Process Framework ($TLPF_3$), as a basis for further research in universities and colleges of education in Nigeria and in other similar contexts (*q.v.* Figure 7.2). A further research study can improve on this study by conducting a lesson study amongst secondary school teachers using the $TLPF_3$ and investigating its appropriateness for programming lessons. Both in-service and pre-service professional development can be conducted for teachers using $TLPF_3$.
- *Improved teaching on programming skills:* Lack of algorithmic and debugging skills, and mediated transfer in both QBASIC and Scratch programming, were identified in the study. I recommend that further research should focus on designing programming activities rich in algorithm design by the students. Programming activities that encourage mediated transfer between Scratch and QBASIC programming can also be structured on teaching programming concepts and problem-solving with an emphasis on arrays and control structures.
- *Teaching the roles of variables:* The study found that students held misconceptions regarding variables. Further research on QBASIC programming in schools, colleges and universities can help to structure teaching on the roles of variables to students, as suggested in the literature.
- *Cognitive tests based on programming taxonomies*: The use of test questions in determining students' level of programming understanding is one of the practices employed in this study. I recommend that lecturers in colleges of education and universities do research into students' learning in introductory programming using the combined revised Bloom's and SOLO taxonomies as well as the Block model.
- *Programming assessment:* I recommend that the BRACELet accepted programming skills of reading, tracing and writing be used by lecturers in colleges and universities for conducting research which focuses on assessing programming courses in Nigerian colleges and universities.

7.7.2 Recommendation for policy

One of the reasons for supporting the learning of procedural programming was to support learning. Since findings from the study reveal that Scratch programming aids critical thinking, builds program writing skills and boost interests in programming. I, therefore, recommend the inclusion of Scratch programming in the computer science curriculum at colleges of education in Nigeria. The move for this policy was also influenced by findings from the study (see sections 5.3.3 and 6.3.3). Since pre-service teachers are trained to

teach students at primary and secondary schools, they need to be well equipped for the challenges ahead. Therefore, policy can be tailored to address this need. On the one hand, Scratch programming can be featured as a first semester course (CSC 102) where students have ample time to internalise and learn programming concepts visually. On the other hand, QBASIC programming (presently a first semester course) can be moved to the second semester as CSC 121. Including Scratch programming into the curriculum will require the provision of necessary textbooks and software for teaching and learning. For implementation, in-service and pre-service teachers in schools and colleges need to be trained and retrained.

7.8 CONCLUSION

The purpose of this study was to explore how a visual programming environment support first year college students' programming skills in procedural programming. The findings from this study indicated that students in the programming classroom were dominant in quadrants B and C. Consequently, programming instruction was facilitated using differentiated instruction that met students' needs. This necessitated the design of a teaching and learning framework for facilitating the programming content. Findings from the study revealed that the teaching intervention which used the framework promoted student learning and engagement in diverse forms. Using the VPE for supporting learning enabled students to: learn programming concepts, build their interest in programming and promote their motivation in procedural programming. Understanding gained in Scratch helped most students to build a correct mental representation of the Block model for procedural programming, except for *relations*. The study also established that students learned programming using different means such as: accommodation and disequilibrium, social interaction, self-regulation and problem-solving techniques. Students, therefore, suggested that Scratch programming be included in the computer science curriculum for colleges of education in Nigeria. However, contextual issues were found to influence the learning of programming and possible solutions to these problems were identified in the study.

In the light of this study, it can, therefore, be concluded that a visual programming environment can be used to support the teaching of procedural programming for first year college students.

"Reasoning draws a conclusion, but it does not make the conclusion certain, unless the mind discovers it by the path of experience" - Roger Bacon

REFERENCES

Ackerman, E. (2001). Piagets constructivism, Paperts constructionism: whats the differences. *Constructivism: Uses and Perspectives in Education, 12*(2), 85-94.

Adenubi, A. O., Lateef, U. O., & Bukonla, A. O. (2007). Developing a simple pedagogical programming language interpreter for junior secondary school students in Nigeria. *Journal of Science and Information Technology, 3*, 1-15.

Adeyemi, A. (2017, January 9). UN priortises ICT skills for economic development *The Guardian* Retrieved from https://guardian.ng/technology/un.priortises-ict-skills-for-economic-development

Ado, I. B. (2016). A field trip approach in teaching mathematics students flowcharts for program writing skills development in Yenagoa local government area of Bayelsa State, Nigeria. *International Journal of Educational Benchmark, 5*(4), 63-75.

Agbetuyi, P., & Oluwatayo, J. (2012). Information and communication technology (ICT) in Nigerian educational system. *Mediterranean Journal of Social Sciences, 3*(3), 41-45.

Aina, J. K. (2015). Analysis of integrated science and computer science students'academic performances in Physics in colleges of education, Nigeria. *International Journal of Education and Practice, 3*(1), 28-35.

Ajayi, A., Olajubu, E. A., Ninan, D., Akinboro, S., & Soriyan, H. A. (2010). Development and testing of a graphical Fortran learning tool for novice programmers. *Interdisciplinary Journal of Information, Knowledge, and Management, 5*, 277-291.

Akinola, O. S., & Nosiru, K. A. (2014). Factors influencing students' performance in computer programming: A fuzzy set operations approach. *International Journal of Advances in Engineering and Technology, 7*(4), 1141-1149.

Aleksic, V., & Ivanovic, M. (2016). Introductory programming subject in European higher education. *Informatics in Education, 15*(2), 163-182.

Aleven, V., Stahl, E., Schworm, S., Fischer, F., & Wallace, R. (2003). Help seeking and help design in interactive learning environments. *Review of educational research, 73*, 277-320.

Alhojailan, M. I. (2012). Thematic analysis: A critical review of its process and evaluation. *West East Journal of Social Sciences, 1*, 39-47.

Altadmri, A., & Brown, N. C. (2015). *37 million compilations: Investigating novice programming mistakes in large-scale student data.* In D. Adrienne, E. Kurt, A. Carl, & T. Jodi, Proceedings of the 46th ACM Technical Symposium on Computer Science Education,Kansas City, MO, USA 4th-7th, March, 2015 (pp. pp. 522-527). ACM.

Ambrosio, A. P., Costa, F. M., Almeida, L., , & Franco, A., & Macedo, J. (2011). Identifying Cognitive abilities to improve CS1 outcome. *Frontiers in Education Conference (FIE)*, 12-15.

Amineh, R. J., & Asl, H. D. (2015). Review of constructivism and social constructivism *Journal of Social Sciences, Literature and Languages, 1*, 9-16.

Anderson, L. W., Krathwohl, D. R., & Bloom, B. S. (2001). *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives*: Allyn & Bacon.

Angelakopoulou, M. (2016). *Scaffolding in classroom: Analysing diagnosing in teacher-student interactions from both a quantitative and a qualitative perspective.* (Doctoral dissertation), Retrieved from https://openaccess.leidenuniv.nl/

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to "Real" programming. *ACM Transactions on Computing Education, 14*(4), 137-151.

Attride-Stirling, J. (2001). Thematic networks: an analytic tool for qualitative research. *Qualitative research, 1*(3), 385-405.

Awbi, N. K., Whalley, J. L., & Philpott, A. (2015). Scaffolding, the zone of proximal development, and novice programmers. *Journal of Applied Computing and Information Technology, 91*(1), 1-2.

Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable programming: blocks and beyond. *Communications of the ACM, 60*(6), 72-80.
Bawaneh, A. K. A., Abdullah, A. G. K., Saleh, S., & Yin, K. Y. (2011a). Jordanian students' thinking styles based on Herrmann whole brain model. *International Journal of Humanities and social science, 1*(9), 89-97.
Bawaneh, A. K. A., Zain, A. N. M., & Saleh, S. (2011b). The effect of Herrmann whole brain teaching method on students' understanding of simple electric circuits. *European Journal of Physics Education, 2*(2), 1-23.
Bayman, P., & Mayer, R. E. (1988). Using conceptual models to teach BASIC computer programming. *Journal of Educational Psychology, 80*(3), 291-298.
Beck, L., & Chizhik, A. (2013). Cooperative learning instructional methods for CS1: Design, implementation, and evaluation. *ACM Transactions on Computing Education, 13*(3). doi:http://dx.doi.org/10.1145/2492686
Begel, A., & Nagappan, N. (2008). *Pair programming: what's in it for me?* In D. Rombach, S. Elbaum, & J. Munch, Proceedings of the second ACM-IEEE international symposium on Empirical software engineering and measurement,Kaiserslautern, Germany, October 9-10, 2008 (pp. 120-128). ACM.
Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching, 20*(1), 45-74.
Bennedsen, J., & Caspersen, M. E. (2007). Failure rates in introductory programming. *ACM SIGCSE Bulletin, 39*(2), 32-36.
Bergersen, G. R. (2015). *Measuring programming skill-construction and validation of an instrument for evaluating Java developers.* (Doctoral dissertation), University of Oslo, Retrieved from https://www.duo.uio.no/handle/10852/48583
Bergersen, G. R., & Gustafsson, J.-E. (2015). Programming skill, knowledge, and working memory among professional software developers from an investment theory perspective. *Journal of Individual Differences, 32*, 201-209.
Bergin, S., Reilly, R., & Traynor, D. (2005). *Examining the role of self-regulated learning on introductory programming performance.* In R. Anderson, S. A. Fincher, & M. Guzdial, Proceedings of the first international workshop on Computing education research, October 01-02, 2005 (pp. 81-86). New York, USA: ACM.
Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers and Education, 72*, 145-157.
Bhattacharjee, J. (2015). Constructivist approach to learning–An effective approach of teaching learning. *International Research Journal of Interdisciplinary & Multidisciplinary Studies, 1*(6), 65-74.
Bichelmeyer, B. (2005). *The ADDIE model: A metaphor for the lack of clarity in the field of IDT*. Retrieved from http://www.indiana.edu/~idt/index.html
Biggs, J. (1996). Enhancing teaching through constructive alignment. *Higher Education, 32*(3), 347-364.
Biggs, J. (2003). Aligning teaching and assessing to course objectives. *Teaching and Learning in Higher Education: New Trends and Innovations, 2*, 13-17.
Biggs, J. (2012). Enhancing learning through constructive alignment. In J. R. Kirby & M. J. Lawson (Eds.), *Enhancing the quality of learning: Dispositions, instruction, and learning processes* (pp. 117-136). New York, USA: Cambridge University Press.
Biggs, J., & Tang, C. (2007). Teaching for Quality Learning at University Open University Press. In.
Biggs, J. B., & Collis, K. F. (1982). *Evaluating the quality of learning: the SOLO taxonomy (structure of the observed learning outcome)*. New York: Academic Press.
Biggs, J. B., & Tang, C. (2011). *Teaching for quality learning at University: What the student does* (4th ed.). New-York: McGraw-Hill Education.
Bishka, A. (2010). Learning styles fray: Brilliant or batty? *Performance Improvement, 49*(10), 9-13.

Brabrand, C., & Dahl, B. (2008). Constructive Alignment and the SOLO Taxonomy. *Koli Calling Proceedings*.
Brabrand, C., & Dahl, B. (2009). Using the SOLO taxonomy to analyze competence progression of University science curricula. *Higher Education, 58*(4), 531-549.
Branch, R. M. (2010). *Instructional design: The ADDIE approach*. USA: Springer Science and Business Media.
Braught, G., Wahls, T., & Eby, L. M. (2011). The case for pair programming in the Computer science classroom. *ACM Transactions on Computing Education, 11*(1), 1-21.
Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative research in psychology, 3*(2), 77-101.
Brennan, K., & Resnick, M. (2012). *New frameworks for studying and assessing the development of computational thinking*. Paper presented at the the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada. Retrieved from http://scratched.gse.harvard.edu.ct/files/AERA2012.pdf
Brooks, R. (1983). Towards a theory of the comprehension of computer programs. *International Journal of Man-machine studies, 18*(6), 543-554.
Brooks, R. A. (1990). Elephants don't play chess. *Robotics and autonomous systems, 6*(1), 3-15.
Brown, A. H., & Green, T. D. (2015). *The essentials of instructional design: Connecting fundamental principles with process and practice*. NewYork: Routledge.
Burkhardt, J.-M., Détienne, F., & Wiedenbeck, S. (2002). Object-oriented program comprehension: Effect of expertise, task and phase. *Empirical Software Engineering, 7*(2), 115-156.
Burton, K. J. (2006). Designing criterion-referenced assessment. *Journal of Learning Design, 1*(2), 73-82.
Burton, P. J., & Bruhn, R. E. (2003). Teaching programming in the OOP era. *ACM SIGCSE Bulletin, 35*(2), 111-114.
Busjahn, T., & Schulte, C. (2013, November, 14-17). *The use of code reading in teaching programming.* Paper presented at the 13th Koli Calling International Conference on Computing Education Research. Retrieved from http://dx.doi.org/10.1145/2526968.2526969
Cain, A., & Woodward, C. J. (2013). *Examining student reflections from a constructively aligned introductory programming unit.* In A. Carbobe & J. Whalley, Proceedings of the Fifteenth Australasian Computing Education Conference,Adelaide, Australia, 14-17, November, 2013 (pp. 127-136). New South Wales: Australian Computer Society, Inc.
Çakıroğlu, Ü. (2014). Analyzing the effect of learning styles and study habits of distance learners on learning performances: A case of an introductory programming course. *The International Review of Research in Open and Distributed Learning, 15*(4), 161-185.
Campbell, V., & Johnstone, M. (2010). *The significance of learning style with respect to achievement in first year programming students.* In J. Noble & C. Fridge, 21st Australian Software Engineering Conference,Auckland, New Zealand 6-10 April, 2010 (pp. 165-170). IEEE. 10.1109/ASWEC.2010.33.
Cao, L., & Xu, P. (2005). *Activity patterns of pair programming.* In Proceedings of the 38th Annual Hawaii International conference on system sciences Big Island, USA, 6th January, 2005 (pp. 1-10). IEEE. 0-7695-2268-8/05$20.00.
Carlisle, M. C. (2009). Raptor: a visual programming environment for teaching object-oriented programming. *Journal of Computing Sciences in Colleges, 24*(4), 275-281.
Cetin, I. (2016). Pre-service teachers' introduction to Computing: Exploring utililization of Scratch. *Journal of Educational Computing Research, 54*(7), 997-1021. doi:https://doi.org/10.1177/0735633116642774
Chang, M.-M. (2005). Applying self-regulated learning strategies in a web-based instruction—an investigation of motivation perception. *Computer Assisted Language Learning, 18*(3), 217-230.

Chao, P.-Y. (2016). Exploring students' computational practice, design and performance of problem-solving through a visual programming environment. *Computers and Education, 95*, 202-215.

Chen, P., & Du, X. (2012). *Improving software comprehension process by Adoption of Cognitive Theories in large-scale complex software maintenance - An empirical research of cognitive theories in software maintenance*. Kandidatuppsatser, Retrieved from Gupea database. Gothenbutg University publications electronic archive.

Cheng, W. F. J. (2010). *Teaching and learning to program: a qualitative study of Hong Kong sub-degree students.* (Doctoral dissertation), University of Technology, Sydney. Retrieved from http://hdl.handle.net/10453/20328

Choi, I., Lee, S. J., & Kang, J. (2009). Implementing a case-based e-learning environment in a lecture-oriented anaesthesiology class: Do learning styles matter in complex problem solving over time? *British Journal of Educational Technology, 40*(5), 933-947.

Clear, T., Edwards, J., Lister, R., Simon, B., Thompson, E., & Whalley, J. (2008a). *The teaching of novice computer programmers: bringing the scholarly-research approach to Australia.* In H. Simon & M. Hamilton, Proceedings of the tenth Australiasian Computing education conference, 01 January 2008 (pp. 63-68). New South Wales: Australian Computer Society, Inc.

Clear, T., Whalley, J., Lister, R., Carbone, A., Hu, M., Sheard, J., . . . Thompson, E. (2008b). *Reliably classifying novice programmer exam responses using the SOLO taxonomy.* In 21st Annual Conference of the National Advisory Committee on Computing Qualifications,Auckland (pp. 23-30). National advisory committee on computing

Coffield, F. (2012). Learning Styles: unreliable, invalid and impractical and yet still widely used. In P. Adey & J. Dillon (Eds.), *Bad Education: Debunking Myths in Education.* (pp. 215-230). Berkishire: Open University Press.

Coffield, F., Moseley, D., Hall, E., & Ecclestone, K. (2004). *Learning styles and pedagogy in post 16 learning: a systematic and critical review*. London: The Learning and Skills Research Centre.

Coghlan, D., & Brannick, T. (2014). Epistemology. In: *The Sage enclyclopedia of Action research*. (Vol. 1,pp. 302-305). London: Sage.

Coghlan, D., & Brydon-Miller, M. (2014). Hermeneutics. In: *The Sage encyclopedia of action research*. (Vol. 1,pp. 404-405). London: Sage Publications Ltd.

Cohen, L., Manion, L., & Morrison, K. (2007). *Research methods in Education*. New York: Routledge.

Cohen, L., Manion, L., & Morrison, K. (2011). *Research methods in education*. New York: Routledge.

Corritore, C. L., & Wiedenbeck, S. (2001). An exploratory study of program comprehension strategies of procedural and object-oriented programmers. *International Journal of Human-Computer Studies, 54* (1), 1-23.

Cozby, P. C., & Bates, S. C. (2012). *Methods in Behavioral Research*. New York: McGraw-Hill Education.

Craik, K. J. (1943). *The nature of explanation.* Cambridge, UK: Cambridge University Press.

Creswell, J. W. (2007). *Qualitative enquiry and research design: Choosing among five approaches* (2nd ed.). New Delhi: Sage.

Creswell, J. W. (2012). *Educational research: Planning conducting, and evaluating quantitative and qualitative research* (4th ed.). Upper Saddle river, New Jersey: Pearson Education Inc.

Creswell, J. W. (2014). *Reseach Design: Quantitative, Qualitative and Mixed methods approaches* (4th ed.). Upper Saddle river, New Jersey: Pearson Education Inc.

Crotty, M. (1998). *The foundations of of social research: Meaning and perspective in the research process*: Thousand Oaks, CA, Sage.

Cuevas, J. (2015). Is learning styles-based instruction effective? A comprehensive analysis of recent research on learning styles. *Theory and Research in Education, 13*(3), 308-333.

Curry, L. (1987). *Integrating concepts of cognitive or learning style: A review with attention to psychometric standards*. Ottawa: Learning styles network.

Dagdilelis, V., Satratzemi, M., & Evangelidis, G. (2004). Introducing secondary education students to algorithms and programming. *Education and Information Technologies, 9*(2), 159-173.

Daniel, R. T. (2010). *Action Research for Educators* (2nd ed.). Lanham, Maryland Rowman & Littlefield Education.

Dasgupta, C. (2010). *That is not my program: investigating the relation between program comprehension and program authorship.* In H. C. Cunningham, Proceedings of the 48th Annual Southeast Regional Conference,New York, USA, April 15-17, 2010 (pp. 103). ACM. 10.1145/19000008.1900142.

De Boer, A.-L., Bothma, T., & du Toit, P. (2011). Enhancing information literacy through the application of whole brain strategies. *Libri, 61*(1), 67-75.

De Boer, A.-L., Du Toit, P., Scheepers, D., & Bothma, T. (2013). *Whole Brain® Learning in higher education: Evidence-based practice*. New Delhi: Chandos Publishing.

De Boer, A., Steyn, T., & Du Toit, P. (2001). A whole brain approach to teaching and learning in higher education. *South African Journal of Higher Education, 15*(3), 185-193.

De Corte, E. (2010). Historical developments in the understanding of learning. In H. Dumont, D. Istance, & F. Benavides (Eds.), *The nature of learning. Using research to inspire practice* (pp. 35-67). Paris: Center for educational research and innovation.

Dehnadi, S., & Bornat, R. (2006). The camel has two humps (working title). *Middlesex University, UK*, 1-21.

Denner, J., Werner, L., Campe, S., & Ortiz, E. (2014). Pair Programming: Under what conditions is it advantageous for middle school students? *Journal of Research on Technology in Education, 46*(3), 277-296.

Denner, J., Werner, L., & Ortiz, E. (2012). Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers and Education, 58*(1), 240-249.

Dillon, E., Anderson, M., & Brown, M. (2012). *Comparing mental models of novice programmers when using visual and command line environments.* In R. K. Smith & S. V. Vrbsky, Proceedings of the 50th Annual Southeast Regional Conference,Tuscaloosa, Alabama, March 29 - 31, 2012 (pp. 142-147). New York: ACM. 10.1145/2184512.2184546.

Dooly, M. (2008). *Telecollaborative language learning: A guidebook to moderating intercultural collaboration online*. Switzerland: Peter Lang AG, International Academic Publishers.

du Boulay, B., O'Shea, T., & Monk, J. (1981). The black box inside the glass box: presenting computing concepts to novices. *International Journal of Man-machine studies, 14*(3), 237-249.

Du Toit, P. (2010). *An Action Research approach for monitoring one's professional development as a manager*. Pretoria: Foundation for professional devlopment.

Du Toit, P. H. (2009). *An action research approach to monitoring one's professional development as manager*. Pretoria: Foundation for Professional Development.

Du Toit, P. H. (2012). Using action research as process for sustaining knowledge production: A case study of a higher education qualification for academics. *South African Journal of Higher Education, 26*(6), 1216-1233.

Du Toit, P. H., De Boer, A.-L., Bothma, T., & Scheepers, D. (2010). *Multidisciplinary collaboration: A necessity for curriculum innovation*. Paper presented at the Proceedings of the World Library and Information Congress: 76th IFLA General Conference and Assembly, Gothenburg, Sweden. Retrieved from http://www.ifla.org/en/ifla76.

Ebersöhn, L., Elof, I., & Ferreira, R. (2007). First stpes in action research. In K. Maree (Ed.), *First stpes in research* (1st ed., pp. 123-143). Pretoria: Van Schaik Publishers.
Ebersöhn, L., Eloff, I., & Ferreira, R. (2016). First steps in action research. In K. Maree (Ed.), *First steps in research* (2nd ed., pp. 134-159). Pretoria: Van Schaik Publishers.
Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M., Sanders, K., & Zander, C. (2006). *Putting threshold concepts into context in computer science education.* In M. Goldweber & P. Salomoni, ACM SIGCSE Bulletin,Bologna, Italy 26–28 June 2006 (pp. 103-107). ACM. 1-59593-055-8/06/0006.
Efron, S. E., & Ravid, R. (2013). *Action research in education: A practical guide*. New York: Guilford Press.
Eid, C., & Millham, R. (2012). Which introductory programming approach is most suitable for students: Procedural or visual programming? *American Journal of Business Education, 5*(2), 173-178.
Entwistle, N., McCune, V., & Hounsell, J. (2002). Approaches to studying and perceptions of University teaching-learning environments: Concepts, measures and preliminary findings. *Enhancing Teaching and Learning Environments in Undergraduate Courses: Occasional Report 1, 1*, 1-19.
Exeter, D. J., Ameratunga, S., Ratima, M., Morton, S., Dickson, M., Hsu, D., & Jackson, R. (2010). Student engagement in very large classes: The teachers' perspective. *Studies in Higher Education, 35*(7), 761-775.
Eysenck, M. W., & Keane, M. T. (2000). *Cognitive psychology: A student's handbook*. USA: Taylor & Francis.
Ezziane, Z. (2007). Information technology literacy: Implications on teaching and learning. *Journal of Educational Technology and Society, 10*(3), 175-191.
Faja, S. (2014). Evaluating effectiveness of pair programming as a teaching tool in programming courses. *Information Systems Education Journal, 12*(6), 36-44.
Falkner, K., & Palmer, E. (2009). Developing authentic problem solving skills in introductory computing classes. *ACM SIGCSE Bulletin, 41*(1), 4-8.
Federal Ministry of Education, F. (1998). *Report on National Policy on Computer Education*. Lagos: NERDC Press.
Federal Ministry of Education, F. (2004). *Ministerial initiative on e – education for Nigerian educational system*. Abuja: NERDC Press.
Fee, S. B., & Holland-Minkley, A. M. (2010). Teaching computer science through problems, not solutions. *Computer Science Education, 20*(2), 129-144.
Feilzer, M. Y. (2010). Doing mixed methods research pragmatically: Implications for the rediscovery of pragmatism as a research paradigm. *Journal of Mixed Methods Research, 4* (1), 6–16.
Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers and Education, 63*, 87-97.
Fincher, S., Cooper, S., Kölling, M., & Maloney, J. (2010). *Comparing Alice, Greenfoot and Scratch.* In G. Lewandowski, S. Wolfman, T. J. Cortina, E. L. Walker, & D. R. Musicant, Proceedings of the 41st ACM technical symposium on Computer science education Wisconsin, March 10–13, 2010 (pp. 192-193). New York: Association for Computing Machinery, Inc. 978-1-60558-885-8.
Fitzgerald, S., Lewandowski, G., McCauley, R., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: finding, fixing and flailing, a multi-institutional study of novice debuggers. *Computer Science Education, 18*(2), 93-116.
Forehand, M. (2010). Bloom's taxonomy. In M. Orey (Ed.), Emerging perspectives on learning, teaching, and technology (pp. 41-44). Retrieved from http://www.palieducationsociety.org/images/ebooks%20(13).
Fosnot, C. T., & Perry, R. S. (Eds.). (2013). *Constructivism: Theory, perspectives, and practice* (2nd ed.). New York: Teachers College Press.
Fox, N. J. (2008). Induction in qualitative research. In: L. M. Given, *The SAGE encyclopedia of qualitative research methods*. (Vols. 2,pp. 86-89). New York: Sage.

Frankish, K., & Ramsey, W. (2012). *The cambridge handbook of cognitive science*: Cambridge University Press.

Frey, B. B., Schmitt, V., L, & Justin, P. A. (2012). Defining authentic classroom assessment. *Practical Assessment, Research and Evaluation, 17*(2), 1-18.

Fuller, U., Johnson, C. G., Ahoniemi, T., Cukierman, D., Hernán-Losada, I., Jackova, J., . . . Riedesel, C. (2007). *Developing a computer science-specific learning taxonomy.* In Proceedings of the 12th annual conference on innovation and technology in Computer science education,Dundee, Scotland, June 23 - 27, 2007 (pp. 152-170). New York: ACM.

Gadamer, H.-G. (Ed.) (1997). *Truth and method* (2nd ed.). New York: Continuum.

Gao, R. (2011). *Reforming to improve the teaching quality of computer programming language.* In Proceedings of the 6th International Conference on Computer Science and Education,Singapore, August 3-5, 2011 (pp. 1267-1269). IEEE.

Gasaymeh, A.-M., AlJa'afreh, I. A., Al-Dmour, A., & Alrub, M. A. (2016). Higher education students' preferences for applying the principles of constructivism in learning programming languages with the use of ICTs. *Journal of Studies in Education, 6*(3), 168-187.

Gebre, E., Saroyan, A., & Bracewell, R. (2014). Students' engagement in technology rich classrooms and its relationship to professors' conceptions of effective teaching. *British Journal of Educational Technology, 45*(1), 83-96.

Geelan, D. (2007). *Weaving narrative nets to capture classrooms: Multimethod qualitative approaches for educational research*. Netherlands: Springer.

Gellenbeck, E. M., & Cook, C. R. (1991). *An investigation of procedure and variable names as beacons during program comprehension.* In J. Koenemaa-Belliveau, T. G. Moher, & S. P. Robertson, Proceedings of the Fourth Workshop on Empirical Studies of Programmers,New Brunswick, Dec 7-9, 1991 (pp. 65-79). New Jersey: Ablex publishing corporation

George, C. E. (2000). *Experiences with novices: The importance of graphical representations in supporting mental models.* In A. F. Blackwell & E. Bilotta, 12th annual workshop of the psychology of programming interest group, Cozenza, Italy (pp. 33-44). Citeseer.

Ginat, D. (2004). On novice loop boundaries and range conceptions. *Computer Science Education, 14*(3), 165-181.

Girvan, C., Tangney, B., & Savage, T. (2013). SLurtles: supporting constructionist learning in second life. *Computers and Education, 61*, 115-132.

Given, L. M. (2008). *The Sage encyclopedia of qualitative research methods*. London: Sage Publications.

Godbole, S. (2014). *Impact of a visual programming experience on the attitude toward programming of introductory undergraduate students.* (Masters dissertation), Purdue University Graduate School, West Lafayette, Indiana. Retrieved from https://docs.lib.purdue.edu/open_access_theses/327/

Gomes, A., & Mendes, A. J. (2007). *An environment to improve programming education.* In B. Rachev, A. Smrikarov, & D. Dimov, Proceedings of the 2007 international conference on Computer systems and technologies and workshop for PhD students in computing,Bulgaria, 14-15 June, 2007 (pp. 193-196). ACM.

Gouws, L. A., Bradshaw, K., & Wentworth, P. (2013). *Computational thinking in educational activities: an evaluation of the educational game light-bot.* In J. Carter, Proceedings of the 18th ACM conference on Innovation and technology in computer science education,Canterbury, England July 01 - 03, 2013 (pp. 10-15). USA: ACM.

Govender, I. (2009). *Learning to program, learning to teach programming: pre-and in service teachers' experiences of an object-oriented language.* (Doctoral dissertation), University of South Africa, Retrieved from http://hdl.handle.net/10500/1495

Grandell, L., Peltomäki, M., & Salakoski, T. (2005). *High school programming—a beyond-syntax analysis of novice programmers' difficulties.* In T. Salakoski, Proceedings of

the Koli Calling 2005 Conference on Computer Science Education,Finland (pp. 17-24).
Grix, J. (2004). *The foundations of research* (2nd ed.). New York: Palgrave Macmillan.
Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational researcher, 42*(1), 38-43.
Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education, 25*(2), 199-237.
Guba, E., G, & Lincoln, Y. S. (2005). *Competing Paradigmatic controversies, contradictions and emerging confluences* Thousand Oaks: Sage.
Guibert, N., Girard, P., & Guittet, L. (2004). *Example-based Programming: A pertinent visual approach for learning to program.* In M. F. Costabile, Proceedings of the working conference on Advanced visual interfaces,Gallipoli, Italy May 25 - 28, 2004 (pp. 358-361). USA: ACM.
Gurudatt, K., & Sathyaraj, R. (2014). A survey on graphical programming systems. *An International Journal of Advanced Computer Technology, 3* (4), 709-703.
Hari, O. P., Bhushan, P., R, Venkataraman, S., & Geeta, V. (2013). Integrated visual programming environment. *International Journal of Modeling and Optimization, 3*(3), 256-260.
Hartfield, P. J. (2010). Reinforcing constructivist teaching in advanced level biochemistry through the introduction of case-based learning activities. *Journal of Learning Design, 3*(3), 20-31.
Harvey, B., & Mönig, J. (2010). Bringing “no ceiling” to Scratch: Can one language serve kids and computer scientists. *Constructionism, 1*, 1-10.
Hawk, T. F., & Shah, A. J. (2007). Using learning style instruments to enhance student learning. *Decision Sciences Journal of Innovative Education, Volume 5* (1), 1-9.
Hazzan, O., Lapidot, T., & Ragonis, N. (2015). *Guide to teaching computer science: An activity-based approach* (2nd ed.). London: Springer.
Heidegger, M. (1962). *Being and Time*: San Francisco: Harper & Row, Inc.
Heit, E., & Rotello, C. M. (2010). Relations between inductive reasoning and deductive reasoning. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 36*(3), 805-812.
Held, C., Vosgerau, G., & Knauff, M. (Eds.). (2006). *Mental models and the mind: Current developments in cognitive psychology, neuroscience and philosophy of mind* (Vol. 138). Netherlands: Elsevier.
Hendricks, C. (2006). *Improving Schools through Action Research: A comprehensive guide for educators*. United States of America: Pearson.
Hendricks, C. (2013). *Improving schools through Action Research: A reflective practice approach*. United States of America (USA): Pearson Education Inc.
Herrington, A., & Herrington, J. (Eds.). (2006). *Authentic learning environments in higher education*. Hersley: Information Science Publishing.
Herrmann, K. J. (2013). The impact of cooperative learning on student engagement: Results from an intervention. *Active Learning in Higher Education, 14*(3), 175-187.
Herrmann, N. (1996). *The whole brain business book*: McGraw Hill Professional.
Hmelo-Silver, C. E. (2004). Problem-based learning: What and how do students learn? *Educational psychology review, 16*(3), 235-266.
Hofstedt, P. (2011). *Multiparadigm constraint programming languages*. London: Springer Heidelberg Dordrech.
Howard-Jones, P. A. (2014). Neuroscience and education: myths and messages. *Nature Reviews Neuroscience, 15*(12), 817-824.
Hsieh, S.-W., Jang, Y.-R., Hwang, G.-J., & Chen, N.-S. (2011). Effects of teaching and learning styles on students’ reflection levels for ubiquitous learning. *Computers and Education, 57*(1), 1194-1201.

Hwang, G.-J., Sung, H.-Y., Hung, C.-M., Huang, I., & Tsai, C.-C. (2012a). Development of a personalized educational computer game based on students' learning styles. *Educational technology research and development, 60*(4), 623-638.

Hwang, W.-Y., Shadiev, R., Wang, C.-Y., & Huang, Z.-H. (2012b). A pilot study of cooperative programming learning behavior and its relationship with students' learning performance. *Computers and Education, 58*(4), 1267-1281.

Hyde, K. F. (2000). Recognising deductive processes in qualitative research. *Qualitative market research: An international journal, 3*(2), 82-90.

Iji, C. (2005). Information and communication technology (ICT) in human resource development: Issues in Nigeria certificate in education (NCE) Computer science curriculum. *The Nigerian Academic Forum: A Multidisciplinary Journal, 9*(2-5), 102-109.

Illeris, K. (2007). *How we learn: Learning and non-learning in school and beyond*. London: Taylor and Francis.

Illeris, K. (2009). *Contemporary theories of learning: learning theorists in their own words*. Park Square, Milton Park, Abingdon: Taylor and Francis.

Ismail, M. N., Ngah, N. A., & Umar, I. N. (2010). Instructional strategy in the teaching of computer programming: a need assessment analyses. *The Turkish Online Journal of Educational Technology, 9*(2), 125-131.

Isong, B. (2014). A methodology for teaching computer programming: First year students' perspective. *International journal of modern education and computer science, 6*(9), 15-21.

Jegede, P. O. (2009). Predictors of Java programming self efficacy among engineering students in a Nigerian University. *International Journal of Computer Science and Information Security, 4*(1 and 2), 1-7.

Jimoyiannis, A. (2011). Using SOLO taxonomy to explore students' mental models of the programming variable and the assignment statement. *Themes in Science and Technology Education, 4*, 53-74.

Johnson-Laird, P. N. (1985). Mental models: Towards a cognitive science of language, inference, and consciousness. *Linguistic Society of America, 61*(4), 897-903.

Johnson, D. W., & Johnson, R. T. (2009). An educational psychology success story: Social interdependence theory and cooperative learning. *Educational researcher, 38*(5), 365-379.

Johnson, R. B., & Onwuegbuzie, A. J. (2004). Mixed methods research: A research paradigm whose time has come. *Educational researcher, 33*(7), 14-26.

Jonassen, D. (2009). Reconciling a new cognitive architecture In T. Sigmund & D. M. Thomas (Eds.), *Constructivist instruction: Success or Failure?* London: Routlege, Taylor and Francis.

Jonassen, D. (2011a). *Learning to solve problems: A handbook for designing problem solving learning environments*. New York: Routledge

Jonassen, D. H. (1991). Objectivism versus constructivism: Do we need a new philosophical paradigm? *Educational technology research and development, 39*(3), 5-14.

Kafai, Y. B., & Burke, Q. (2013). Computer programming goes back to school. *Phi Delta Kappan, 95*(1), 61-65.

Kafai, Y. B., & Resnick, M. (1996). Introduction. In Y. B. Kafai & M. Resnick (Eds.), *Constructionism in practice: Designing, thinking and learning in a digital world* (pp. 1-8). NewYork: Routledge.

Kafle, N. P. (2013). Hermeneutic phenomenological research method simplified. *Bodhi: An Interdisciplinary Journal, 5*(1), 181-200.

Katai, Z. (2011). Multi-sensory method for teaching-learning recursion. *Computer Applications in Engineering Education, 19*(2), 234-243.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR), 37*(2), 83-137.

Kelvin, E. (2016, November 13). Frank: No conscious effort to develop ICT in Nigeria. *The Guardian* Retrieved from https://guardian.ng/saturday-magazine/cover/fank-no-conscious-efforts-to-develop-ict-in-nigeria

Kemmis, S., McTaggart, R., & Nixon, R. (2014). *The Action Research Planner: Doing Critical Participatory Action Research*. Singapore: Springer Science Business Media.

Khalil, M. K., & Elkhider, I. A. (2016). Applying learning theories and instructional design models for effective instruction. *Advances in Physiology Education, 40*, 147-156. doi:doi:10.1152/advan.00138.2015

King, A. (1990). Enhancing peer interaction and learning in the classroom through reciprocal questioning. *American Educational Research Journal, 27*(4), 664-687.

Kinnunen, P., & Malmi, L. (2008). *CS minors in a CS1 course.* In R. Lister, M. E. Caspersen, & M. Clancy, Proceedings of the fourth international workshop on Computing education research,Sydney, Australia, September 06 - 07, 2008 (pp. 79-90). New York: ACM.

Kirstein, M., & Kunz, R. (2016). A Whole Brain® learning approach to an undergraduate auditing initiative – an exploratory study. *Meditari Accountancy Research, 24*(4), 527-544. doi:https://doi.org/10.1108/MEDAR-02-2014-0029

Klein, H. K., & Myers, M. D. (1999). A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS quarterly, 23*(1), 67-93.

Kolb, A. Y., & Kolb, D. A. (2005a). The Kolb learning style inventory–version 3.1 2005 technical specifications. *Experience Based Learning Systems, Inc., 200*, 1-72.

Kolb, A. Y., & Kolb, D. A. (2005b). Learning styles and learning spaces: Enhancing experiential learning in higher education. *Academy of management learning and education, 4*(2), 193-212.

Kolb, A. Y., & Kolb, D. A. (2009). The learning way: Meta-cognitive aspects of experiential learning. *Simulation and Gaming, 40*(3), 297-327.

Kolb, A. Y., & Kolb, D. A. (2012). Experiential learning theory. In S. N.M. (Ed.), *Encyclopedia of the Sciences of Learning* (pp. 1215-1219). Boston: Springer.

Kolikant, Y. B.-D., & Mussai, M. (2008). "So my program doesn't run!" Definition, origins, and practical expressions of students'(mis) conceptions of correctness. *Computer Science Education, 18*(2), 135-151.

Koorsse, M., Cilliers, C., & Calitz, A. (2015). Programming assistance tools to support the learning of IT programming in South African secondary schools. *Computers and Education, 82*, 162-178.

Kordaki, M. (2012). Diverse categories of programming learning activities could be performed within Scratch. *Procedia-Social and Behavioral Sciences, 46*, 1162-1166.

Koshy, V. (2009). *Action research for improving educational practice: A step-by-step guide*. India: Sage.

Kosky, V. (2009). *Action Research for improving Educational Practice: A step-by-step (2nd ed.)*. India: Sage Publications.

Kothiyal, A., Majumdar, R., Murthy, S., & Iyer, S. (2013). *Effect of think-pair-share in a large CS1 class: 83% sustained engagement.* In B. Simon, S. Diego, A. Clear, & Q. Cutts,San Diego, San California, August 12 - 14, 2013 (pp. 137-144). New York: ACM.

Koulouri, T., Lauria, S., & Macredie, R. D. (2015). Teaching introductory programming: a quantitative evaluation of different approaches. *ACM Transactions on Computing Education, 14*(4), 1-28.

Krathwohl, D. R. (2002). A revision of Bloom's taxonomy: An overview. *Theory into practice, 41*(4), 212-218.

Krpan, D., Mladenović, S., & Zaharija, G. (2017). *Mediated transfer from visual to high-level programming language.* In proceedings of the 40th international convention on information and communication technology, electronics and microelectronics Opatija, Croatia, 22-26 May 2017 (pp. 800-805). IEEE. 10.23919/MIPRO.2017.7973531.

Kuhn, K.-A. L., & Rundle-Thiele, S. R. (2009). Curriculum alignment: Exploring student perception of learning achievement measures. *International Journal of Teaching and Learning in Higher Education, 21*(3), 351-361.

Kumar, A. N. (2013). *A study of the influence of code-tracing problems on code-writing skills.* In J. Carter, I. Utting, & A. Clear,Canterbury, England, July 01 - 03, 2013 (pp. 183-188). New York: ACM.

Kynigos, C. (2015). Constructionism: Theory of learning or theory of design? In S. Cho (Ed.), *Selected Regular Lectures from the 12th International Congress on Mathematical Education* (pp. 417-438). Cham: Springer.

Lahtinen, E. (2007). *A categorization of Novice Programmers: a cluster analysis study.* In Proceedings of the 19th annual workshop of the psychology of programming interest group,Joensuu, Finnland (pp. 32-41). Citeseer.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H.-M. (2005). *A study of the difficulties of novice programmers.* In J. Cunha, W. Fleischman, & J. L. Viera Proulx, Innovation and Technology in Computer Science Education,Caparica, Portugal June 27 - 29, 2005 (pp. 14-18). New York: ACM.

Langdridge, D. (2007). *Phenomenological psychology: Theory, research and method*. England: Pearson Education.

Langenhoven, C. (2015). *User experiences of learners in technology-facilitated learning in a resourced deprived context.* (Masters dissertation), University of Pretoria, Pretoria. Retrieved from http://hdl.handle.net/2263/52939

Lau, W., & Yuen, A. (2010). Promoting conceptual change of learning sorting algorithm through the diagnosis of mental models: The effects of gender and learning styles. *Computers & Education, 54*(1), 275-288.

Lau, W., & Yuen, A. (2011). Modeling programming performance: beyond the influence of learner characteristics. *Computers and Education, 57*(1), 1202-1213.

Laverty, S. M. (2003). Hermeneutic phenomenology and phenomenology: A comparison of historical and methodological considerations. *International Journal of Qualitative Methods, 2*(3), 21-35.

Law, K. M., Lee, V. C., & Yu, Y.-T. (2010). Learning motivation in e-learning facilitated computer programming courses. *Computers and Education, 55*(1), 218-228.

le Roux, I. (2011). New large class pedagogy: developing students' whole brain thinking skills. *Procedia-Social and Behavioral Sciences, 15*, 426-435.

Letovsky, S. (1987). Cognitive processes in program comprehension. *Journal of Systems and software, 7*(4), 325-339.

Lewis, C. M. (2010). *How programming environment shapes perception, learning and goals: logo vs. scratch.* In G. Lewandowski, S. Wolfman, T. J. Cortina, E. L. Walker, & D. R. Musicant, Proceedings of the 41st ACM technical symposium on Computer science education,Milwaukee, Wisconsin, March 10 - 13, 2010 (pp. 346-350). New York: ACM.

Li, Z. Z., Cheng, Y. B., & Liu, C. C. (2013). A constructionism framework for designing game-like learning systems: Its effect on different learners. *British Journal of Educational Technology, 44*(2), 208-224.

Liakopoulou, M. (2011). The professional competence of teachers: Which qualities, attitudes, skills and knowledge contribute to a teacher's effectiveness. *International Journal of Humanities and social science, 1*(21), 66-78.

Lin, Y.-T., Wu, C.-C., Hou, T.-Y., Lin, Y.-C., Yang, F.-Y., & Chang, C.-H. (2016). Tracking students' cognitive processes during program debugging—An eye-movement approach. *IEEE transactions on education, 59*(3), 175-186.

Lister, R., Clear, T., Bouvier, D. J., Carter, P., Eckerdal, A., Jacková, J., . . . Seppälä, O. (2010). Naturally occurring data as research instrument: analyzing examination responses to study the novice programmer. *ACM SIGCSE Bulletin, 41*(4), 156-173.

Lister, R., Fidge, C., & Teague, D. (2009). Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. *ACM SIGCSE Bulletin, 41*(3), 161-165.

Lister, R., Simon, B., Thompson, E., Whalley, J. L., & Prasad, C. (2006). Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. *ACM SIGCSE Bulletin, 38*(3), 118-122.

Littman, D. C., Pinto, J., Letovsky, S., & Soloway, E. (1986). Mental Model and Software Maintenance. *Journal of Systems and software, 7*(4), 341-355.

Lodico, M. G., Spaulding, D. T., & Voegtle, K. H. (2011). *Methods in Educational Research: From theory to practice* (2nd ed.). San Francisco: Jossey-Bass.

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). *Relationships between reading, tracing and writing skills in introductory programming.* Paper presented at the Proceedings of the Fourth international Workshop on Computing Education Research. Retrieved from

Louden, K. C., & Lambert, K. A. (2011). *Programming languages: principles and practices* (3rd ed.). Boston: Course Technology, Cengage Learning.

Luo, D. (2005). Using constructivism as a teaching model for Computer science. *The China Papers, 5,* 36-40.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior, 41*, 51-61.

Ma, L., Ferguson, J., Roper, M., & Wood, M. (2007). Investigating the viability of mental models held by novice programmers. *ACM SIGCSE Bulletin, 39*(1), 499-503.

Ma, L., Ferguson, J. D., Roper, M., Ross, I., & Wood, M. (2008). Using cognitive conflict and visualisation to improve mental models held by novice programmers. *ACM SIGCSE Bulletin, 40*(1), 342-346.

MacLaurin, M. B. (2011). The design of Kodu: A tiny visual programming language for children on the Xbox 360. *ACM Sigplan Notices, 46*(1), 241-246.

Malan, D. J., & Leitner, H. H. (2007). Scratch for budding computer scientists. *ACM SIGCSE Bulletin, 39*(1), 223-227.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch programming language and environment. *ACM Transactions on Computing Education, 10*(4), 1-15.

Manktelow, K., & Chung, M. C. (2004). *Psychology of reasoning: theoretical and historical perspectives*. New York: Psychology Press.

Manolis, C., Burns, D. J., Assudani, R., & Chinta, R. (2013). Assessing experiential learning styles: A methodological reconstruction and validation of the Kolb Learning Style Inventory. *Learning and individual differences, 23*, 44-52.

Maree, K., & Pietersen, J. (2007). Survey and the use of Questionnaires. In K. Maree (Ed.), *First steps in Research* (1st ed., pp. 154-156). Pretoria, South Africa: Van Schaik.

Martin, S. (2010). Teachers using learning styles: Torn between research and accountability? *Teaching and teacher education, 26*(8), 1583-1591.

Martínez-Valdés, J. A., Velázquez-Iturbide, J. Á., & Hijón-Neira, R. (2017). *A relatively unsatisfactory experience of use of Scratch in CS1.* In J. M. Dodero, M. S. I. Sáiz, & I. R. Rube, Proceedings of the 5th international conference on technological ecosystems for enhancing multiculturality,Cádiz, Spain, October 18 - 20, 2017 (pp. 1-7). New York: ACM.

Mason, R., Cooper, G., & de Raadt, M. (2012). *Trends in introductory programming courses in Australian universities: languages, environments and pedagogy.* In M. d. Raadt & A. Carbone, Proceedings of the fourteenth australasian computing education conference,Melbourne, Australia, January 31 - February 03, 2012 (pp. 33-42). Australia: Australian Computer Society, Inc.

Mbogo, C. C. (2015). *Scaffolding Java programming on a mobile phone for novice learners.* (Doctoral dissertation), University of Cape Town, Retrieved from http://hdl.handle.net/11427/16609

McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: a review of the literature from an educational perspective. *Computer Science Education, 18*(2), 67-92.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y., . . . Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin, 33*(4), 125-180.

McGill, T. J., & Volet, S. E. (1997). A conceptual framework for analyzing students' knowledge of programming. *Journal of research on Computing in Education, 29*(3), 276-297.

McKinney, D., & Denton, L. F. (2006a). *Developing collaborative skills early in the CS curriculum in a laboratory environment.* In (pp. 138-142). ACM.

McKinney, D., & Denton, L. F. (2006b). Developing collaborative skills early in the CS curriculum in a laboratory environment. *ACM SIGCSE Bulletin, 38*(1), 138-142.

McNiff, J., & Jack, W. (2010). *You and your action research project* (3rd ed.). London and NewYork: Routledge.

McNiff, J., & Whitehead, J. (2006a). *All you need to know about action research*. New York: Sage

McNiff, J., & Whitehead, J. (2006b). *All you need to know about Action Research: An Introduction*. London: SAGE Publications.

McNiff, J., & Whitehead, J. (2010). *You and your action research project* (3rd Ed.). New York: Routledge.

McVee, M. B., Dunsmore, K., & Gavelek, J. R. (2005). Schema theory revisited. *Review of educational research, 75*(4), 531-566.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2011). *Habits of programming in Scratch.* In Proceedings of the 16th annual joint conference on Innovation and technology in computer science education,Darmstadt, Germany, June 27 - 29, 2011 (pp. 168-172). ACM.

Meerbaum-Salant, O., Armoni, M., & Ben-Ari, M. (2013). Learning computer science concepts with scratch. *Computer Science Education, 23*(3), 239-264.

Mentz, E., Van der Walt, J., & Goosen, L. (2008). The effect of incorporating cooperative learning principles in pair programming for student teachers. *Computer Science Education, 18*(4), 247-260.

Mertens, D. M. (2010). *Reasearch and evaluation in education and psychology: Integrating diversity with quantitative, qualitative and mixed methods* (3rd ed.). London: SAGE.

Mertler, C. A. (2008). *Action research: Teachers as researchers in the classroom*. United Kongdom: Sage.

Micari, M., & Pazos, P. (2014). Worrying about what others think: A social-comparison concern intervention in small learning groups. *Active Learning in Higher Education, 15*(3), 249-262.

Miles, M. B., Huberman, A. M., & Saldana, J. (2014). *Qualitative data analysis*. New Delhi: Sage.

Millis, B. J. (2010). *Cooperative learning in higher education: Across the disciplines, across the academy*. Sterling: Stylus publishing

Mills, G. E. (2011). *Action Research. A guide for the teacher researcher* (4th ed. ed.). Upper Saddle River, NJ: Pearson/Allyn Bacon.

Milne, I., & Rowe, G. (2002). Difficulties in learning and teaching programming—views of students and tutors. *Education and Information Technologies, 7*(1), 55-66.

Mladenović, M., Boljat, I., & Žanko, Ž. (2017a). Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level. *Education and Information Technologies*, 1-18.

Mladenović, M., Krpan, D., & Mladenović, S. (2017b). *Learning programming from Scratch.* In International conference on new horizons in education (pp.

Mladenovic, M., Rosic, M., & Mladenovic, S. (2016). Comparing elementary students' programming success based on programming environment. *International journal of modern education and computer science, 8*(8), 1-10.

Mladenović, S., Krpan, D., & Mladenović, M. (2016). Using games to help novices embrace programming: from elementary to higher education. *International journal of engineering education, 32*(1), 521-531.

Monig, J., Ohshima, Y., & Maloney, J. (2015). *Blocks at your fingertips: Blurring the line between blocks and text in GP.* In F. Turbak, D. Bau, J. Gray, C. Kelleher, & J. Sheldon, Blocks and Beyond Workshop,Atlanta, Georgia, 22 October, 2015 (pp. 51-53). USA: Institute of Electrical and Electronics Engineers, Inc. 10.1109/BLOCKS.2015.7369001.

Moreno, J., & Robles, G. (2014). *Automatic detection of bad programming habits in scratch: A preliminary study.* In (pp. 1-4). IEEE.

Mössenböck, H. (2012). *Object-oriented programming in Oberon-2* (2nd ed.). Berlin: Springer Science and Business Media.

Mow, I. C. (2008). Issues and difficulties in teaching novice computer programming. In I. M. (Ed.), *Innovative techniques in instruction technology, e-learning, e-assessment, and education* (pp. 199-204). Dordrecht: Springer.

Munn-Giddings, C., & Winter, R. (2013). *A handbook for action research in health and social care*. London and NewYork: Routledge.

Myller, N., Bednarik, R., Sutinen, E., & Ben-Ari, M. (2009). Extending the engagement taxonomy: Software visualization and collaborative learning. *ACM Transactions on Computing Education, 9*(1), 7.

Nanja, M., & Cook, C. R. (1987). *An analysis of the on-line debugging process.* In (pp. 172-184). Ablex Publishing Corp.

NCCE. (2008). *Nigeria Ceritificate in Education minimum standards for Sciences.* Garki, Abuja: National Commission for Colleges in Education.

NCCE. (2012). *Nigeria Certificate in Education minimun standards for Sciences.* . Garki, Abuja: National Commission for Colleges of Education.

Ngozo, B. P. (2012). *Dynamics of learning style flexibility in teaching and learning.* (Masters dissertation), University of Pretoria,

Nieuwenhuis, J. (2014a). Analysing qualitative data. In M. Kobus (Ed.), *First Steps in Research* (1st ed.). Pretoria: Van Schaik.

Nieuwenhuis, J. (2014b). Qualitative research and data gathering techniques. In K. Maree (Ed.), *First Steps in Research* (pp. 67-97). Pretoria: Van Schaik Publishers.

Nilson, L. B. (2016). *Teaching at its best: A research-based resource for college instructors* (4th ed.). San Fransisco: John Wiley & Sons.

Nuraminah, R., & Shukor, S. M. F. (2008). The Effects of Pair Programming in Programming Language Subject *IEEE*.

O'donoghue, T. (2007). *Planning your qualitative research project: An introduction to interpretivist research in education*. New York: Routledge.

O'Leary, T., & O'Leary, L. (2013). *Computing essential: Making IT work for you*. London: McGraw-Hill international

Olelewe, C., & Fakorede, S. (2009). Information and communication Technology: A tool for reforming instructional processes in tertiary institutions. *Academic Staff Union Journal Eha-Amufu Chapter, 1*(1), 54-62.

Olelewe, C. J. (2016). *Strategies for improving the teaching and learning of Qbasic programming language in colleges of education in Enugu, Ebonyi and Anambra states.* (Doctoral dissertation),

Olelewe, C. J., & Agomuo, E. E. (2016). Effects of B-learning and F2F learning environments on students' achievement in QBASIC programming. *Computers and Education, 103*, 76-86.

Ololube, N. P. (2007). The relationship between funding, ICT, selection processes, administration and planning and the standard of science teacher education in Nigeria. *Asia-Pacific Forum on Science Learning and Teaching, 8*(1), 1-29.

Onwuegbuzie, A. J., & Leech, N. L. (2005). On becoming a pragmatic researcher: The importance of combining quantitative and qualitative research methodologies. *International Journal of Social Research Methodology, 8*(5), 375-387.

Opdecam, E., Everaert, P., Van Keer, H., & Buysschaert, F. (2014). Preferences for team learning and lecture-based learning among first-year undergraduate accounting students. *Research in Higher Education, 55*(4), 400-432.

Ormrod, J. (1999). *Human learning* (3rd ed.). New Jersey: Upper Saddle River.
Otuniyi, A. K. (2013). *Enhancing teacher quality: The COL-NCCE collaborative expereince* Paper presented at the International conference of the Seventh Pan-Commonwealth forum on Open learning Abuja. Retrieved from https://scholar.google.com/scholar?q=Enhancing+Teacher+Quality%3A+The+COL-NCCE+collaborative+expereince+&submit3.x=0&submit3.y=0.
Ouahbi, I., Kaddari, F., Darhmaoui, H., Elachqar, A., & Lahmine, S. (2015). Learning Basic Programming Concepts by Creating Games with Scratch Programming Environment. *Procedia-Social and Behavioral Sciences, 191*, 1479-1482.
Owolabi, J., Adebayo, A. O., Amao-Kehinde, A. O., & Olayanju, T. A. (2015). Effects of technology selected laboratory-based instructional approaches on computer students' self efficacy in Visual-Basic.NET computer programming. *Global Journal of Education, 1*(1), 60-67.
Owolabi, J., Olanipekun, P., & Iwerima, J. (2014). Mathematics ability and anxiety, computer and programming anxieties, age and gender as determinants of achievement in Basic programming. *GSTF Journal on Computing, 3*(4), 109-114.
Özgen, K., Tataroglu, B., & Alkan, H. (2011). An examination of multiple intelligence domains and learning styles of pre-service mathematics teachers: Their reflections on mathematics education. *Educational Research and Reviews, 6*(2), 168-182.
Özmen, B., & Altun, A. (2014). Undergraduate students' experiences in programming: Difficulties and obstacles. *Turkish Online Journal of Qualitative Inquiry, 5*(3), 9-25.
Paivio, A. (1990). *Mental representations: A dual coding approach*. USA: Oxford University Press.
Panselinas, G., & Komis, V. (2009). 'Scaffolding'through talk in groupwork learning. *Thinking Skills and Creativity, 4*(2), 86-103.
Papert, S., & Harel, I. (1991). Situating constructionism. *Constructionism, 36*(2), 1-11.
Pashler, H., McDaniel, M., Rohrer, D., & Bjork, R. (2008). Learning styles: Concepts and evidence. *Psychological science in the public interest, 9*(3), 105-119.
Patton, M., Q. (2002). *Qualitative Research and Evaluation Methods*. United Kingdom: Sage.
Pea, R. D. (1986). Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research, 2*(1), 25-36.
Pelton, R. P. (2010). *Action research for Teacher candidates: Using classroom data to enhance instruction*. Maryland: Rowman & Littlefield Education.
Pennington, N. (1987). Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive psychology, 19*(3), 295-341.
Perkins, D., & Martin, F. (1986). *Fragile knowledge and neglected strategies in novice programmers.* In E. Soloway & S. Iyengar,Washinton, DC, June 5-6, 1986 (pp. 213-229). First workshop on empirical studies of programmers on empirical studies of programmers.
Phillips, D. K., & Kevin, C. (2014). *Becoming a Teacher through Action Research: Process, Context, and Self-Study* (3rd ed.). New York: Routledge.
Philpott, A., Robbins, P., & Whalley, J. (2007). *Assessing the steps on the road to relational thinking.* In S. Mann & N. Bridgeman, Proceedings of the 20th annual conference of the national advisory committee on computing qualifications,July 8-11, 2007 (pp.
Piaget, J. (1964). Part I: Cognitive development in children: Piaget development and learning. *Journal of research in science teaching, 2*(3), 176-186.
Pietersen, J., & Maree, K. (2016). Standardisation of a questionnaire. In K. Maree (Ed.), *First steps in Research* (2nd ed., pp. 237-247). Pretoria: Van Schaik
Piggot-Irvine, E. (2015). *Goal Pursuit in Education Using Focused Action Research*. United states of America: Palmgrave Macmillan.
Pine, G. J. (2008). *Teacher action research: Building knowledge democracies*. London: Sage.
Pintrich, P. R. (2002). The role of metacognitive knowledge in learning, teaching, and assessing. *Theory into practice, 41*(4), 219-225.

Potgieter, E. (1999). Relationship between the whole brain creativity model and Kolb's experiential learning model. *Curationis, 22*(4), 9-14.
Pring, R. (2000). The 'false dualism'of educational research. *Journal of Philosophy of Education, 34*(2), 247-260.
Pritchard, A. (2013). *Ways of learning: Learning theories and learning styles in the classroom*. USA and Canada: Routledge.
Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education, 18*(1), 1-24.
Quaye, A. M., & Dasuki, S. I. (2017). A computational approach to learning programming using visual programming in a developing country University. In Rich P. & H. C. (Eds.), *Emerging Research, Practice, and Policy on Computational Thinking* (pp. 121-134): Springer.
Qudah, A. (2009). *The effect of teaching using the McCarthy method on the biology achievement for the fourth learning styles of tenth female graders in Ajloun.* (Master dissertation), Yarmouk University, Irbed, Jordan.
Raes, A., Schellens, T., De Wever, B., & Vanderhoven, E. (2012). Scaffolding information problem solving in web-based collaborative inquiry learning. *Computers and Education, 59*(1), 82-94.
Reid, G. (2005). *Learning styles and inclusion*. London: Sage publications.
Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., . . . Silverman, B. (2009). Scratch: programming for all. *Communications of the ACM, 52*(11), 60-67.
Resnick, M., Ocko, S., & Papert, S. (1988). LEGO, Logo, and design. *Children's Environments Quarterly, 5*(4), 14-18.
Ricoeur, P. (Ed.) (1981). *Hermeneutics and the human sciences: Essays on language, action, and interpretation*. United Kingdom: Cambridge University Press
Rizvi, M., & Humphries, T. (2012). Scratch programming for undergraduates. *Journal of Computing Sciences in Colleges, 27*(3), 107-107.
Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education, 13*(2), 137-172. doi:10.1076/csed.13.2.137.14200
Rogers, K. M. A. (2009). A preliminary investigation and analysis of student learning style preferences in further and higher education. *Journal of Further and Higher Education, 33*(1), 13-21.
Sáez-López, J.-M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers and Education, 97*, 129-141.
Sajaniemi, J., Ben-Ari, M., Byckling, P., Gerdt, P., & Kulikova, Y. (2006). Roles of variables in three programming paradigms. *Computer Science Education, 16*(4), 261-279.
Sajaniemi, J., & Kuittinen, M. (2008). From procedures to objects: A research agenda for the psychology of object-oriented programming education. *Human Technology: An Interdisciplinary Journal on Humans in ICT Environments*.
Saldaña, J. (2015). *The coding manual for qualitative researchers*. London: Sage.
Sankey, M., Birch, D., & Gardiner, M. (2012). The impact of multiple representations of content using multimedia on learning outcomes across learning styles and modal preferences. *International Journal of Education and Development using ICT, 7*(3), 18-35.
Saunders, M., Lewis, P., & Thornhill, A. (2009). *Research methods for business students* (5th ed.). England: Pearson Education Limited.
Saunders, M., Lewis, P., & Thornhill, A. (2012). *Research methods for business students* (6th ed.). Harlow: Pearson Education Limited.
Schdmit, L. K. (2006). *Understanding Hermeneutics*. Durham: Acumen Publishing Limited.

Schmeck, R. R. (2013). An introduction to learning strategies and learning styles. In R. R. Schmeck (Ed.), *Learning strategies and learning styles*. New York Springer Science and Business Media.

Schön, D. A. (1983). *The reflective practitioner: How professionals think in action*. USA: Ashgate publishing.

Schulte, C. (2008). *Block Model: An educational model of program comprehension as a tool for a scholarly approach to teaching.* In R. Lister, M. E. Caspersen, & M. Clancy, Proceedings of the Fourth international Workshop on Computing Education Research,Sydney, Australia, September 6-7, 2008 (pp. 149-160).

Schulte, C., Clear, T., Taherkhani, A., Busjahn, T., & Paterson, J. H. (2010). *An introduction to program comprehension for computer science educators.* In A. Clear & L. R. Dag, Proceedings of the 2010 ITiCSE working group reports,Ankara, Turkey June 28 - 30, 2010 (pp. 65-86). New York: ACM.

Schunk, D., H. (2008). *Learning theories: An educational perspectives* (4th ed.). United States of America Pearson Prentice Hall.

Schunk, D., H. (2014). *Learning theories: An educational perspective* (6th ed.). United states of America: Pearson education limited.

Scott, C. (2010). The enduring appeal of 'learning styles'. *Australian Journal of Education, 54*(1), 5-17.

Scott , D., & Morrison, M. (2005). *Key ideas in educational research* NewYork: Continuum International Publishing Group.

Scott, D., & Usher, R. (2011). *Researching Education Data, Methods and Theory in Educational Enquiry*. NewYork: Continuum International Publishing Group.

Sefotho, M. M. (2015). A researcher's dilemma: Philosophy in crafting dissertations and theses. *Journal of Social Sciences, 42*(1), 23-26.

Serrano-Cámara, L. M., Paredes-Velasco, M., Alcover, C.-M., & Velazquez-Iturbide, J. Á. (2014). An evaluation of students' motivation in computer-supported collaborative learning of programming concepts. *Computers in Human Behavior, 31*, 499-508.

Shapiro, R. B., & Ahrens, M. (2016). Beyond blocks: Syntax and semantics. *Communications of the ACM, 59,* 39-41.

Shaw, R.-S. (2012). A study of the relationships among learning styles, participation types, and performance in programming language learning supported by online forums. *Computers and Education, 58*(1), 111-120.

Sheard, J., Carbone, A., Lister, R., Simon, B., Thompson, E., & Whalley, J. L. (2008). Going SOLO to assess novice programmers. *ACM SIGCSE Bulletin, 40*(3), 209-213

Shi, N., Cui, W., Zhang, P., & Sun, X. (2018). Evaluating the effectiveness roles of variables in the novice programmers learning. *Journal of Educational Computing Research, 56*(2), 181-201.

Shneider, E., & Gladkikh, O. (2006). *Designing questioning strategies for information technology courses.* In S. Mann & N. Bridgeman, Proceedings of the 19th Annual Conference of the National Advisory Committee on Computing Qualifications: Preparing for the Future — Capitalising on IT Wellington (pp. 243-248). Australia: National Advisory Committee on Computing Qualifications.

Shuhidan, S., Hamilton, M., & D'Souza, D. (2009). *A taxonomic study of novice programming summative assessment.* In M. Hamilton & T. Clear, Proceedings of the Eleventh Australasian Conference on Computing Education,Wellington, New Zealand, January 01 2009 (pp. 147-156). Australia: Australian Computer Society, Inc.

Sigmund, T., & Thomas, M. (2009). The Success or Failure of Constructivist Instruction An Introduction. In T. Sigmund & M. Thomas (Eds.), *Constructivist Instruction Success or Failure?* London: Routledge, Taylor and Francis.

Simon, B., & Hanks, B. (2008a). First-year students' impressions of pair programming in CS1. *Journal on Educational Resources in Computing, 7*(4), 1-28.

Simon, B., & Hanks, B. (2008b). First-year students' impressions of pair programming in CS1. *Journal on Educational Resources in Computing (JERIC), 7*(4), 5.

Smith, J. A., Flowers, P., & Larkin, M. (2009). Interpretative phenomenological analysis: theory, method and research. In G. M. Breakwell (Ed.), *Doing Social Psychology Research* (pp. 229-254). London: Blackwell Publishing Ltd.

Soloway, E., & Ehrlich, K. (1984). Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*(5), 595-609.

Sorva, J. (2013). Notional machines and introductory programming education. *ACM Transactions on Computing Education, 13*(2), 1-31.

Spohrer, J. C., & Soloway, E. (1986). Novice mistakes: Are the folk wisdoms correct? *Communications of the ACM, 29*(7), 624-632.

Stachel, J., Marghitu, D., Brahim, T. B., Sims, R., Reynolds, L., & Czelusniak, V. (2013). Managing cognitive load in introductory programming courses: A cognitive aware scaffolding tool. *Journal of Integrated Design and Process Science, 17*(1), 37-54.

Starr, C. W., Manaris, B., & Stalvey, R. H. (2008). Bloom's taxonomy revisited: specifying assessable learning objectives in computer science. *ACM SIGCSE Bulletin, 40*(1), 261-265.

Stolin, Y., & Hazzan, O. (2007). Students' understanding of computer science soft ideas: the case of programming paradigm. *ACM SIGCSE Bulletin, 39*(2), 65-69.

Stringer, E. T. (2007). *Action Research in Education* (2nd ed.). United States of America: Sage Publications, Inc.

Stringer, E. T. (2014). *Action Research in education (2nd ed.)*. Harlow, Essex: Pearson Education Limited.

Su, A., Yang, S. J., Hwang, W. Y., Huang, C. S., & Tern, M. Y. (2014). Investigating the role of computer-supported annotation in problem-solving-based teaching: An empirical study of a Scratch programming pedagogy. *British Journal of Educational Technology, 45*(4), 647-665.

Sullivan, A., & Bers, M. U. (2016). Girls, boys, and bots: Gender differences in young children's performance on robotics and programming tasks. *Journal of Information Technology Education: Innovations in Practice, 15*, 145-165.

Swidan, A., Serebrenik, A., & Hermans, F. (2017). *How do Scratch programmers name variables and procedures?* In Proceedings of International Working Conference onSource Code Analysis and Manipulation,Shangai, China (pp. 51-60). Tokyo: Institute of electronic engineers.

Tafliovich, A., Campbell, J., & Petersen, A. (2013). *A student perspective on prior experience in CS1.* In Proceeding of the 44th ACM technical symposium on Computer science education,Denver, Colorado March 06 - 09, 2013 (pp. 239-244). New York: Association for Computng Machinery.

Tan, G., & Venables, A. (2010). Wearing the assessment 'BRACElet'. *Journal of Information Technology Education: Innovations in Practice, 9*, 25-34.

Tan, H., Wilson, A., & Olver, I. (2009). Ricoeur's theory of interpretation: An instrument for data interpretation in hermeneutic phenomenology. *International Journal of Qualitative Methods, 8*(4), 1-15.

Tan, P.-H., Ting, C.-Y., & Ling, S.-W. (2009). *Learning difficulties in programming courses: undergraduates' perspective and perception.* In (pp. 42-46). IEEE.

Taylor, G. R., & MacKenney, L. (2008). *Improving human learning in the classroom: Theories and Teaching Practices*. United Kinddom: R&L Education.

Tedlock, B. (2005). The observation of participation and the emergence of public ethnography In N. K. Denzin & Y. S. Lincoln (Eds.), *The Sage handbook of Qualitative Research* (3rd ed.). New Delhi: Sage.

Thompson, E., Luxton-Reilly, A., Whalley, J. L., Hu, M., & Robbins, P. (2008). *Bloom's taxonomy for CS assessment.* In S. Hamilton & M. Hamilton, Proceedings of the tenth conference on Australasian Computing education,Wollongong, Australia January 01 - 01, 2008 (pp. 155-161). Australia: Australian Computer Society, Inc.

Thota, N., & Negreiros, J. G. (2015). Introducing educational technologies to teachers: Experience report. *Journal of University Teaching and Learning Practice, 12*(1), 5.

Thota, N., & Whitfield, R. (2010). Holistic approach to learning and teaching introductory object-oriented programming. *Computer Science Education, 20*(2), 103-127.
Tiantong, M., & Teemuangsai, S. (2013). The four scaffolding modules for collaborative problem-based learning through the computer network on moodle LMS for the Computer Programming course. *International Education Studies, 6*(5), 47-55.
Toker, S., & Moseley, J. L. (2013). The mental model comparison of expert and novice performance improvement practitioners. *Performance Improvement Quarterly, 26*(3), 7-32.
Topalli, D., & Cagiltay, N. E. (2018). Improving programming skills in engineering education through problem-based game projects with Scratch. *Computers and Education, 120*, 64-74.
Tulbure, C. (2011). Do different learning styles require differentiated teaching strategies? *Procedia-Social and Behavioral Sciences, 11*, 155-159.
Turesky, E. F., & Gallagher, D. (2011). Know thyself: Coaching for leadership using Kolb's experiential learning theory. *The Coaching Psychologist, 7*(1), 5-14.
Vaismoradi, M., Turunen, H., & Bondas, T. (2013). Content analysis and thematic analysis: Implications for conducting a qualitative descriptive study. *Nursing and Health Sciences, 15*(3), 398-405.
Van de Pol, J., Volman, M., & Beishuizen, J. (2010). Scaffolding in teacher–student interaction: A decade of research. *Educational psychology review, 22*(3), 271-296.
van de Pol, J., Volman, M., & Beishuizen, J. (2012). Promoting teacher scaffolding in small-group work: A contingency perspective. *Teaching and teacher education, 28*(2), 193-205.
van de Pol, J., Volman, M., Oort, F., & Beishuizen, J. (2014). Teacher scaffolding in small-group work: An intervention study. *Journal of the Learning Sciences, 23*(4), 600-650.
Van Manen, M. (1991). *The tact of teaching: The meaning of pedagogical thoughtfulness*. New York: Althouse Press.
Van Manen, M. (2002). Writing Phenomenology. In M. Van Manen (Ed.), *Writing in the dark – phenomenological studies in interpretive inquiry* (1st ed., pp. 1-8). London: The Althouse Press
Van Manen, M. (2016). *Researching lived experience: Human science for an action sensitive pedagogy* (2nd ed.). NewYork: Routledge.
van Oordtand, M. L., van Oordt, T., & du Toit, P. H. (2014). Are two teachers better than one? *Meditari Accountancy Research, 22*(2), 165-185. doi:https://doi.org/10.1108/MEDAR-01-2013-0003
Van Zyl, S., Mentz, E., & Havenga, M. (2016). Lessons learned from teaching Scratch as an introduction to object-oriented programming in Delphi. *African Journal of Research in Mathematics, Science and Technology Education, 20*(2), 131-141.
Vasilopoulos, I. V. (2014). *The design, development and evaluation of a visual programming tool for novice programmers: psychological and pedagogical effects of introductory programming tools on programming knowledge of Greek students.* (Doctoral dissertation), Teesside University, United Kingdom. Retrieved from http://hdl.handle.net/10149/347149
Verdú, E., Regueras, L. M., Verdú, M. J., Leal, J. P., de Castro, J. P., & Queirós, R. (2012). A distributed system for learning programming on-line. *Computers and Education, 58*(1), 1-10.
Verenikina, I. (2008). Scaffolding and learning: Its role in nurturing new learners. In Peter Kell, Wilma Vialle, D. Konza, & G. Vogl (Eds.), Learning and The Learner: Exploring Learning for New Times (pp. 161-180): Faculty of Education, University of Wollongong.
Von Glasersfeld, E. (1998). Cognition, construction of knowledge, and teaching. In *Constructivism in science education* (pp. 11-30): Springer.
Von Mayrhauser, A., & Vans, A. M. (1995). Program understanding: Models and experiments. *Advances in computers, 40*, 1-38.

Von Mayrhauser, A., & Vans, A. M. (1996). Identification of dynamic comprehension processes during large scale maintenance. *IEEE Transactions on Software Engineering, 22*(6), 424-437.

von Mayrhauser, A., & Vans, A. M. (1997). *Program understanding behavior during debugging of large scale software.* In S. Wiedenbeck & J. Scholtz, Proceeding of the Workshop on Empirical studies of programmers,Alexandria, Virginia, June 1997 (pp. 157-179). USA: ACM.

Vrachnos, E., & Jimoyiannis, A. (2017). Secondary education students' difficulties in algorithmic problems with arrays: An analysis using the SOLO taxonomy. *Themes in Science and Technology Education, 10*(1), 31-52.

Vujošević-Janičić, M., & Tošić, D. (2008). The role of programming paradigms in the first programming courses. *The Teaching of Mathematics, 11*(2), 63-83.

Vygotsky, L. S. (1978). *Mind in Society*. Cambridge: Harvard University Press.

Wassermann, U., Noome, C., Gentle, E., Gibson, K., Macmillan, P., & Zeeman, M. (2011). *IT is gr8 @ 10*. Dorandia: Study Opportunities.

Watkins, K. Z., & Watkins, M. J. (2009). Towards minimizing pair incompatibilities to help retain under-represented groups in beginning programming courses using pair programming. *Journal of Computing Sciences in Colleges, 25*(2), 221-227.

Weintrop, D., & Wilensky, U. (2015). *To block or not to block, that is the question: students' perceptions of blocks-based programming.* In M. Lee & T. Strelevitz, Proceedings of the 14th International Conference on Interaction Design and Children,Boston, Massachusetts June 21 - 24, 2015 (pp. 199-208). New York: Association for Computing Machinery.

Weintrop, D., & Wilensky, U. (2017). Comparing block-based and text-based programming in high school computer science classrooms. *ACM Transactions on Computing Education, 18*(1), 1-25.

Werner, L. L., Hanks, B., & McDowell, C. (2004). Pair-programming helps female Computer science students. *Journal on Educational Resources in Computing, 4*(1), 1-8.

Whalley, J., & Kasto, N. (2013). *Revisiting models of human conceptualisation in the context of a programming examination.* In A. Carbone & J. Whalley, Proceedings of the Fifteenth Australasian Computing Education Conference,Adelaide, Australia, January 29 - February 01, 2013 (pp. 67-76). Australian Computer Society, Inc.

Whalley, J., & Katso, N. (2014). *A qualitative think-aloud study of novice programmers' code writing startegies.* In Proceedings of the Innovation and Technology in Computer Science Education Conference,Uppsala, Sweden, June 21 - 25, 2014 (pp. 279-284). New York: Association for Computing Machinery.

Whalley, J. L., Clear, T., & Lister, R. (2007). The many ways of the BRACElet project. *Journal of Applied Computing and Information Technology, 5*(1), 1-16.

Whalley, J. L., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, P., & Prasad, C. (2006). *An Australasian study of reading and comprehension skills in novice programmers, using the bloom and SOLO taxonomies.* In (pp. 243-252). Australian Computer Society, Inc.

Whalley, J. L., & Robbins, P. (2007). Report on the fourth BRACElet workshop. *Bulletin of Applied Computing and Information Technology, 5*(1), Retrieved from https://www.citrenz.ac.nz/bacit/0501/2007Whalley_BRACELET_Workshop.htm.

White, G., & Sivitanides, M. (2005). Cognitive differences between procedural programming and object oriented programming. *Information Technology and management, 6*(4), 333-350.

Wiedenbeck, S. (1999). The use of icons and labels in an end user application program: an empirical study of learning and retention. *Behaviour and Information Technology, 18*(2), 68-82.

Wiedenbeck, S., Labelle, D., & Kain, V. N. (2004). *Factors affecting course outcomes in introductory programming.* In (pp. 97-109).

Wiggins, G. (April, 2011). A true test: Toward more authentic and equitable assessment. *Phi Delta Kappan, 92,* 81-93.

Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. *IEEE software, 17*(4), 19-25.
Williams, N., I. (2015). *Assessing the quality of computer science courses using taxonomy tools.* (Doctoral dissertation), Flinders University,South Australia, Adelaide.
Willig, C. (2008). *Introducing Qualitative Research in Psychology (2nd ed.)*. England: Open University Press.
Willingham, D. T., Hughes, E. M., & Dobolyi, D. G. (2015). The scientific status of learning styles theories. *Teaching of Psychology, 42*(3), 266-271.
Willis, J. W., & Jost, M. (2007). *Foundations of qualitative research: Interpretive and critical approaches*. London: Sage.
Wing, J. M. (2006). Computational thinking. *Communications of the ACM, 49*(3), 33-35.
Winslow, L. E. (1996). Programming pedagogy—a psychological overview. *ACM SIGCSE Bulletin, 28*(3), 17-22.
Wolz, U., Leitner, H. H., Malan, D. J., & Maloney, J. (2009). Starting with Scratch in CS 1. *ACM SIGCSE Bulletin, 41*(1), 2-3.
Yeh, K.-C. (2009). *Toward understanding the cognitive processes of software design in novice programmers.* (Doctoral dissertation), The Pennsylvania State University, Retrieved from https://etda.libraries.psu.edu/catalog/9775
Yin, R. K. (2011). *Qualitative research from start to finish*. New York: Guilford Publications.
Yurdugül, H., & Aşkar, P. (2013a). Learning Programming, Problem Solving and Gender: A Longitudinal Study. *Procedia - Social and Behavioral Sciences, 83*, 605-610. doi:10.1016/j.sbspro.2013.06.115
Yurdugül, H., & Aşkar, P. (2013b). Learning programming, problem solving and gender: A longitudinal study. *Procedia-Social and Behavioral Sciences,, 83,*, 605-610.
Zacharis, N. Z. (2011). The effect of learning style on preference for web-based courses and learning outcomes. *British Journal of Educational Technology, 42*(5), 790-800.
Zhang, Y. (2007). *An Ontology-based Program Comprehension Model.* (Doctoral dissertation), Concordia University, Canada. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.137.6466&rep=rep1&type=pdf
Zuber-Skerritt, O. (1996). *New Directions in Action Research*. London: The Falmer Press.
Zuber-Skerritt, O. (Ed.) (2001). *Action Learning and Action Research: Paradigm, praxis, and programs'*. Lismore: Southern Cross University Press.

Appendix A 1: Letter of Authorization from the site

INTERNAL MEMORANDUM

From: The Registrar

To: Tijani, (Miss.) Fatimah Yetunde.
Lecturer III, Dept. of Comp. Science.
School of Sciences.
Ufs: Dean, School of Sciences.

MOCPED/REG/SPF/481/VOL.2/244 Date: February 23, 2016.

RE: AUTHORIZATION TO SERVE IN A RESEARCH STUDY

I write to acknowledge the receipt of your letter on the above subject matter and to inform you that the Provost, on behalf of the College Management has approved your request to conduct your research work in the Computer Science Department of the College and as well make use of other relevant technologies available for teaching and learning in School of Sciences.

It is hoped that you will reciprocate the good gesture of the College Management by making judicious use of the College facilities and rendering more valuable services to the College.
Thank you

Olaleye, A.T. (Mrs.)
Deputy Registrar, Personnel, SSM.
For : Registrar

Appendix A 2: Letter of permission to use Kolb's instrument

Subject: RE: Signature page

From: Joe.McDonald@haygroup.com
To: tijanifatima_2000@yahoo.com
Date: Monday, January 11, 2016, 6:19:26 PM GMT+2

Dear Fatimah,

Congratulations! Your LSI research has been approved! Attached you will find the following documents:

- MCB200C - This is a copy of the LSI 3.1 test. You may print of copy this as needed for your research.
- MCB200D - The profile sheet contains the answer key for the test as well as the profiling graphs for plotting scores. This document may be produced as necessary for your research. The AC-CE score on the Learning Style Type Grid is obtained by subtracting the CE score from the AC score. Similarly, the AE-RO score is AE minus RO.

These files are for your data collection only. This permission does not extend to include a copy of the files in your research paper. It should be sufficient to source it.

We wish you luck with your research and look forward to hearing about your findings. Please send a completed copy of your research to Joe.McDonald@haygroup.com or you can mail a hardcopy to:

LSI Research Contracts

c/o Joe McDonald

Hay Group, Inc.

399 Boylston Street

4th Floor, Suite 400

Boston, MA 02116

Please let me know if you have any questions.

Kind regards,

Joe

Appendix A 3: Letter of permission to use HBDI instrument

Subject: Re: Appointment for consultation

From: Pieter.duToit@up.ac.za
To: tijanifatima_2000@yahoo.com
Date: Friday, November 20, 2015, 9:19:40 AM GMT+2

We can meet on Thursday at 14:30. I have diarised it provisionally

>>> tijanifatima_2000 <tijanifatima_2000@yahoo.com> 2015/11/20 09:08 AM >>>
Good morning sir,

I will be in school in Wednesday but I don't know if I will be available by 2pm.
I am defending my proposal on that wed and the time is not yet confirmed.

I will get back to you or can we fix Thursday preferably.

Kind regards,
Fatimah

Sent from Samsung tablet

-------- Original message --------
From: Pieter duToit

Date:20/11/2015 07:50 (GMT+02:00)
To: tijanifatima_2000@yahoo.com
Subject: Re: Appointment for consultation

Good morning

Thank you for the payment

I would like to discuss the outcome with you. May we meet on Wednesday 2 November at 11:00 in my office.

Please confirm your availability
Prof Du Toit
>>> fatimah yetunde <tijanifatima_2000@yahoo.com> 2015/11/02 02:36 PM >>>
Good day sir,
I am the student whose request was sent to you from the Hermann international on HBDI.

I will like to make a consultation tomorrow by 3.30pm. I want to be sure if you will be around.
Hoping to hear from you.

kind regards,
Fatimah

Appendix A 4: Interview protocol

Participant Demographic Information	1. What is your name? 2. How old are you? 3. Give a brief description of your educational background. 4. What motivate you to study computer science? 5. Why did you choose this college amongst other colleges of tertiary institutions? 6. Do you have any prior knowledge in programming? If yes, what are your past experiences in the learning of programming?
Research question 1	7. What questions do you have prior to the administration of the HBDI? 8. What learning did you gain from the HBDI and its relation to your thinking profile? 9. Why do you think the intervention boost your thinking preference? 10. How do your friends feel about the HBDI?
Research question 2	11. Tell me what you understand about variables in procedural programming? 12. How can you differentiate between a variable and a string? 13. Please describe your experience in the procedural programming classroom. 14. What is your own perception of programming with visual program environments? a. …Enjoyment of programming. b. ….As in the usefulness of Scratch from transition to BASIC d. ….The most frustrating thing about Scratch e. …Its usefulness for learning concepts 15. Explain to me the relationship that exists between (VPE) and procedural programming. 16. How do your friends feel about visual programming environments (VPE)? 17. How do you feel about programming as a course? 18. What overall experiences have you gained in procedural programming classroom? 19. How has VPE enhanced the learning of procedural programming?
Research question 3	20a. Please give a detailed description of what you were thinking when solving the problem. 2bb. Please give a detailed description of how you solve each problem and why you think your answer is wrong? 21. What procedure did you use in solving the problem? 22. What did problems you encountered when solving the problem? 23. What general information do you want to share?
Primary research question	What suggestions or recommendations do you have with regards to the design of a new teaching and learning framework for programming?

Appendix A 5: Retrospective interview questions

Code 1

```
CLS
10    COUNTER =0:N = 20
15    INPUT "Enter name"; NAM$
20    INPUT "What is the sales ="; SALES
25    COUNTER = COUNTER + 1
30    IF SALES > 99000 THEN GAIN = .3 * SALES
35    IF SALES >= 69000 AND SALES < 99000 THEN GAIN = .25 * SALES
40    IF SALES >= 39000 AND SALES < 69000 THEN GAIN = .17 * SALES
45    IF SALES < 39000 THEN GAIN = .07 * SALES
50    PRINT "The gain is "; GAIN
55IF COUNTER <= 20 THEN GOTO 15
60    END
```

Code 2

```
10 CLS
20 INPUT NUM
40 Sum = 0: Counter = 0
50 Counter = Counter + 2
60 Sum = Sum + Counter
70 IF Counter <= 100 THEN GOTO 50
80 PRINT "The sum is"; Sum
90 END
```

Code 3 (errors are introduced)

```
CLS
10 COUNTER = 0
20 INPUT "MATHEMATICS SCORE="; MATHSCORE
30 INPUT "ENGLISH SCORE="; ENGSCORE
40 INPUT "BIOLOGY SCORE="; BIOSCORE
40 LET TOTSCORE = MATHSCORE + BIOSCORE + ENGSCORE
50 LET AVESCORE = TOTSCORE / 3
60 LET DOUBLESCORE = TOTSCORE ^ 2
0 PRINT "TOTAL SCORE="; TOTSCORE
0 COUNTER = COUNTER + 1
70 IF COUNTER < 5 THEN GOTO 20
80 IF COUNTER = 5 THEN GOTO 90
90 END
```

1.	Identify the variables in the code (Code 1-3)
2.	Which condition makes the control move to line 15 (code 1) The statement IF Counter < 100 THEN GO TO 50 means what in the code? (code 2) How do you understand the two statements in line 70 and 80 (code 3)
3.	What is the work of INPUT in the program? (code 1)
4.	Count = Count + 1 represents? (code 1) Counter = 0, what does it mean (code 2 & 3)
5.	What is the goal of the program? (code 1-3)
6.	Rewrite the program using a FOR...NEXT statement. (Code 1-3)

Appendix A 6: Test of individual concepts (Cycle 1)

DEPARTMENT OF COMPUTER SCIENCE

FIRST SEMESTER 2015/2016

INSTRUCTION: Answer ALL questions **Time: 25mins**

1. What is a stage?
2. From which area do you find the program blocks?
3. Name two (2) program blocks you can use to communicate with the user and identify where each can be found in the Scratch control blocks.
4. Study the algorithm below and answer the questions that follow.

```
BEGIN
        Ask 'Length of the hall?'
        Get Length
        Ask 'Width of the hall?'
        Get Width
        Area = Length x Width
        Cost = Area x 15.26
Deposit = Cost x 0.5
        Balance = Cost-Deposit
Display 'Cost is: N', Cost
    Display 'Deposit is: N', Cost
    Display 'Balance is: N', Cost
    END
```

(i) Identify the variables in the code.
(ii) Explain in plain English what it does.
(iii) Assuming the Length is 8 and Width is 11, calculate the Cost, Deposit and Balance using what you have learnt?

5. Which repetition structures have the potential never to end (be an endless loop)? Explain why?
6. State three repetition structures in Scratch you know and explain their functions?
7.

```
BEGIN
        Counter = 0
        Sum = 0
        While another number DO
            Ask 'what is the number?'
            Get Number
            Counter = Counter + 1
            Sum = Sum + 1
        EndWhile
        Average = Sum/Counter
        Display Counter
        Display Average
END
```

The above pseudocode represents the algorithm to calculate and display the average of a few numbers.
(i) Identify the code that is repeated.
(ii) Prepare a flowchart to illustrate the sequence of events.
(iii) State whether the algorithm is concrete or abstract.

Appendix A 7: Interim test I

DEPARTMENT OF COMPUTER SCIENCE

FIRST SEMESTER 2015/2016

INTERIM TEST 1 Time: 30mins

INSTRUCTION: Answer ALL questions

1. Provide the value for A and C given three different values of the parameter R (20, -10, 0)

```
10      REM     Program to computer the area and circumference of a circle
20      Pi = 3.141592
30      INPUT “Radius of circle is”; R
40      A = R ^ 2 * Pi
50      C = 2 * Pi * R
60      PRINT “The areas is”; A
70      PRINT “The circumference is”; C
```

2. Study and debug this program. You are to rewrite the correct code.

```
10      REM     Program to computer the area and circumference of a circle
20      Pi = 3.141592
30      INPUT “Radius of circle is” R
40      A = R ^ 2 * Pi
20      C = 2 * Pi * r
60      Print “The areas is”; a
70      PRINT “The circumference is”; C
```

3. Look at this code and explain in plain English what it does.

```
Cls
10      INPUT “Enter the length”; L
20      INPUT “enter the breadth”;B
30      INPUT “Enter the height”; H
40      Let Volume of box= L*B*H
50      PRINT “The volume of box =”; Volume
```

4. Write a QBASIC program for the pseudocode below.

```
BEGIN
        Ask ‘Length of the room?’
        Get Length
        Ask ‘Width of the room?’
        Get Width
        Area = Length x Width
        Cost = Area x 10.50
Deposit = Cost x 0.8
        Balance = Cost-Deposit
Display ‘Cost is: N’, Cost
    Display ‘Deposit is: N’, Cost
    Display ‘Balance is: N’, Cost
    END
```

Appendix A 8: Interim Test II and III

DEPARTMENT OF COMPUTER SCIENCE

FIRST SEMESTER 2015/2016

INTERIM TEST II

1. What is the goal of this program below?

```
10  INPUT "Enter the length"; L
20  INPUT "enter the breadth"; B
30  INPUT "Enter the height"; H
40  Let Volume of box= L*B*H
50  PRINT "The volume of box ="; Volume
```

(3 marks)

2. Write a single line of code to perform each of the following tasks

i. Y = X^ 2-2*X-4

ii. Display the value of Y

iii. ?= Name$, Tel$, Age$, Sex$, Matricno

?= "Olamide", "08030693872", "20", "Female", 150349 (5 marks)

3. Given (a=3, b= 4, c=2, i=2, j= 3), evaluate the following expressions in the order of precedence

i. (j * (a ^ 2 – 10)) / 2

ii. (a * (b+i) * c) ^ 2 (4 marks)

4. Write the following expressions in appropriate formula in QBASIC language

(i) $Y = (s (x – 2))(s^2 - 1)$

(ii) P = I ^2 – B *A -A / I (3 marks)

INTERIM TEST III (Practical)

INSTRUCTIONS: Answer two questions. Do wither question 1 and 2 or 3 and 4. You are to run the program on QBASIC environment. Remember to save your work.

Time: 40mins

1. Write a program to enter any two numbers and find their sum, product and difference.
2. Write a program to find the value of a box whose dimensions are 8.5cm by 7.7cm by 5.2cm using READ and DATA statements. Assuming the formula for calculating volume is L x B x H.
3. Write a program to find the average of the following numbers (6, 7, 8, 9, 10).
4. An examination consists of two papers. A candidate fails if his or her percentage for either paper is less than 30%. Input a candidate number, and two percentages. Output the information with the word "Pass" or "fail" as appropriate.

Appendix A 9: Final Test

DEPARTMENT OF COMPUTER SCIENCE

FIRST SEMESTER 2015/2016

COURSE TITLE: BASIC PROGRAMMING **COURSE CODE:** CSC 112

UNIT/STATUS: 2C

TIME: 1 HOUR

INSTRUCTION: ANSWER ALL QUESTIONS

1. Match the following QBASIC keywords to the appropriate category.

DIM	LOOP
REPEAT…UNTIL	CONDITIONAL BRANCHING
IF…THEN	LOOP
FOR…NEXT	ARRAY
GOTO	UNCONDITIONAL BRANCHING

(5marks)

2. BASIC means _____ (2 marks)

3. A small error has been introduced in the code below. Debug the program.

```
CLS
10 INPUT “ENTER THE NAME OF THE AUTHOR=”;Name$
20 INPUT “Enter the title of the book=”; Title
30 INPUT “Enter the date of publication=”; Date$
40 INPUT “Number of pages=” N$
50 INPUT “Place of publication=”; Place
60 PRINT Name$, Title, Date$, Place, N$
70 END
```

(5marks)

4. Study the pseudocode below and answer the questions that follow.

```
BEGIN
    Ask “Length of the room?”
    Get Length
    Ask “Width of the room?”
    Get Width
    Cost = Area x 10.50
    Deposit Cost * 0.8
    Balance = Cost –Deposit
    Display “Cost is”; Cost
    Display “Deposit is”; Deposit
    Display “Balance is”; Balance
END
```

(i) Identify the variables in the code (3 marks)

(ii) Explain in plain English what it does. (3 marks)

5. Suppose a= 5, x=4, y= 2, z =2; execute the following expressions in order of precedence.

(i) (x * (y + 1) * z) ^ 2 (ii) (x + z* (x / y – a ^ 2)) + a (3marks)

6a. This program computes the area and circumference of a circle. Study the program well and prepare a flowchart for the program.

```
10      REM Program to computer the area and circumference of a circle
```

```
20      Pi = 3.141592
30      INPUT “Radius of a circle is”;R
40      A = R ^ 2 * Pi
50      C = 2 * Pi * R
60      PRINT “The area is”;A
70      PRINT “The circumference is“;C
```
(3 marks)

6b. Predict the output of the above program supposing R is 15.4 (2 marks)

7. Generate a QBASIC expression for the arithmetic expression below.

(a) $D = \frac{ra\,(I + r)^n}{(1 + r)^{n-1}}$ (2 marks)

Differentiate between READ/DATA statement and INPUT statement. (2 marks)

8. This program assign values to the 12-element numeric array C. Study and complete the missing lines of code.

```
10 DIM C(12)
20 DATA 1,4,7,10….34
30 FOR I = 1 TO 10
40 READ C(I)
50 PRINT
60
70 END
```
(2 marks)

9. The following program calculates the total score in mathematics and English for five students can be written in another form. ***Rewrite*** the program in a better way and ***make a judgment*** why your solution is better.

```
10 C= 0
20 INPUT “MATHS SCORE”; MS
30 INPUT “ENGLISH SCORE”; ES
40 TOTAL = MS + ES
50 PRINT “TOTAL SCORE”; TOTAL
60 C = C+ 1
70 IF C =5 THEN 90
80 IF C < 5 GOTO 20
      90 END
```
Rewrite (5 marks)

Your Judgement (3 marks)

10. This program assigns values to the 2-dimensional numeric array B given by

B	1	6	7	2	3
	2	5	4	0	1
	1	4	3	8	7

Study and complete the program.

```
10 DIM B(      )
20 FOR I = 1 TO
30 FOR J =
40 READ B(     )
50 PRINT
60
70
80 END
```
(5 marks)

11. CLS

```
10 REM This program inputs a friend's personal details
20 READ NAM$, COMPANY$, TELNO
30 DATA "Omoyeni Saheed', "Shell Corporation Ltd", 08026728272
40 END
```

Rewrite the above program using the INPUT statement. (5 marks)

Goodluck!

Appendix A 10: Test of Individual Concept I (Cycle 2)

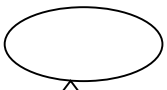
DEPARTMENT OF COMPUTER SCIENCE

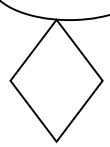
FIRST SEMESTER 2016/2017

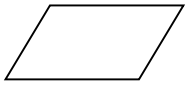
INSTRUCTION: Answer ALL questions **Time: 25mins**

1. What is the full meaning of BASIC? **(1mark)**
2. Mention the 3 major ways in which your program can be represented before you start coding. (1.5 mark)
3. Identify the following flowchart symbols (2.5marks)

a. = _____

b.  **=** _____

c.  = _____

4.  = _____

5.  = _____

4. A step by step method for solving problem is called _____ *(1mark)*

5. A false code or structured method of solving a problem which is not necessarily in a computer programming language is _____. *(1mark)*

6. Write a pseudocode to find the simple interest; if simple interest is P x R x T/100 *(3marks)*

7. Draw a flowchart to *enter any two numbers at the keyboard, and then calculate their Sum, Product and the Difference.* *(4marks)*

8. What is concrete algorithm? *(1mark)*

9. Give two differences between concrete and abstract algorithm *(2marks)*

10. Draw a flowchart that best describe the pseudocode below. *(3marks)*

TASK: Program to enter any number and find out whether it is negative or positive.

```
BEGIN
Ask "What is the number"; N
Get number
    IF   N > 0 THEN
         Display "The number is positive"
    ELSE
         Display "The number is negative"
    ENDIF
END
```

Appendix A 11: Test of individual concept II

DEPARTMENT OF COMPUTER SCIENCE

TEST OF INDIVIDUAL CONCEPTS 2 Time: 15mins

INSTRUCTION: Answer ALL questions

1. "No 54, Iwaya Road, Onipanu Lagos" is a _____ data type.

2. RADIUS is a _____ data type

a. Variable b. String c. Real d. Fixed quantity

3. _____ allows a numeric value or character string to be typed at the variable name or data name specified.

a. READ b. DATA c. INPUT d. LET

4. The process of removing bugs in a program is

a. Correcting b. Debugging c. Removing d. Cleaning

5. The _____ statement provides an alternative method of input to the INPUT statement

a. REMARK b. LET c. IF-THEN d. READ

6. To run a QBASIC program _____ command is used

a. Shift + F5 b. F4 c. F5 d. ALT + F5

7. Which of the following string variables are written incorrectly, Name$, Acct_No$, Address, Name$5

8. ***Write an appropriate statement, or group of statements for each of the situation described below:***

Display the value of A1, A2, A3, A4 and A5 all on one line

a. PRINT A1, A2, A3, A4, A5 b. PRINT A1=; A2=; A3=; A4=; A5=;

c. PRINT A1, A2, A3, A4, A5 d. PRINT A1, A2, A3, A4, A5

9. A non-executable statement used to give comments to the program is _____

Write a LET statement for each of the following situations

10. Assign the value of the expression X/ (A +B-C) to the variable P_STAR

a. LET P_STAR = A\ (A+B-C) b. LET X \ (A+B-C) = P_STAR

c. LET P_STAR = A / (A+B-C) d. LET X / (A+B-C) = P_STAR

11. Write a LET statement for the problem. *Double the value assigned to the variable* Cost

a. LET Cost = 2 * Cost b. LET Cost = Cost ^ 2

c. LET Cost = Cost* Cost d. LET 2 * Cost = Cost

12. Decrease the value assigned to the variable P by 2 and find the square root.

Total marks = 12

Appendix A 12: Interim Test I

DEPARTMENT OF COMPUTER SCIENCE

FIRST SEMESTER 2016/2017

INTERIM TEST I Time: 40mins

INSTRUCTION: Answer ALL questions

1. Study and debug this program. You are to rewrite the correct code.

```
10  REM  Program to computer the area and circumference of a circle
20  Pi = 3.141592
30  INPUT "Radius of circle is" R
40  A = R ^ 2 * Pi
20  C = 2 * Pi * r
60  Print "The areas is"; a
70  PRINT "The circumference is"; C
```

(4marks)

2. Study this code and explain in plain English what it does.

```
CLS
10  INPUT "Enter the length"; L
20  INPUT "enter the breadth"; B
30  INPUT "Enter the height"; H
40  Let Volume of box= L*B*H
50  PRINT "The volume of box ="; Volume
```

(2marks)

3. Write a QBASIC program for the pseudocode below.

```
BEGIN
        Ask 'Length of the room?'
Get Length
        Ask 'Width of the room?'
        Get Width
        Area = Length x Width
        Cost = Area x 10.50
Deposit = Cost x 0.8
        Balance = Cost-Deposit
Display 'Cost is: N', Cost
Display 'Deposit is: N', Cost
Display 'Balance is: N', Cost
END
```

(5marks)

4a. Write a single line of code to perform each of the following tasks

i. Y = X^ 2-2*X-4

ii. Display the value of Y (2marks)

(b) Write the following expressions in appropriate QBASIC formula.

i. $Y = X . \frac{r^2}{c.b} + s.r-b$

ii. $A = (s (x – 2))(S^2 – 1)$ (2marks)

(c) Suppose a = 5, b = 4, c= 2, I = 1, j = 1, evaluate the following in order of precedence.

i. (a * (b + I) * C) ^2

ii. (j * (a ^ 2-10)) / 2 (2marks)

5. Here are some QBASIC code that when arranged in the correct order would make up a program to calculate the roots of equation.

```
START
PRINT "the first root is"; X2
```

```
REM Program to compute roots of equation
X1 = (-b+SQR (b^2-4*a*c))/(2*a)
INPUT A, B, C
X2 = (-b+SQR (b^2-4*a*c))/(2*a)
PRINT “the second root is”; X1
END
```

(5marks)

6. Using an IF…THEN statement, calculate the percentage a learner gets for a test if the learner attained a certain mark and the total marks for the test are 60. Determine whether the learner has a distinction (≥ 80). Display the mark, percentage, and suitable message if the learner has a distinction. *(5marks)*

b. List three QBASIC control structures keywords you know. (3marks)

Note: *Try as much as possible to write neatly.*

Appendix A 13: Interim Test II

DEPARTMENT OF COMPUTER SCIENCE

FIRST SEMESTER 2016/2017 (Interim test II)

(Type 1) Time: 40mins

INSTRUCTION: Answer ALL questions. Run your program on QBASIC and write the solution in your answer sheet.

1. Write a program to enter any two numbers and find their sum, product, and difference.
2. Write a program to find the average of the following numbers: 6, 7, 8, 9 and 10.

(Type 2) Time: 40mins

INSTRUCTION: Answer ALL questions. Run your program on QBASIC and write the solution in your answer sheet.

1. Using the **READ** and **DATA** statements, write a program to find the value of a box whose sides are 8.5cm by 7.7cm by 2.5cm. Assuming the formula for calculating Volume is L x B x H
2. Write a program to find the circumference of a circle.

Appendix A 14: Final Test

DEPARTMENT OF COMPUTER SCIENCE

FIRST SEMESTER 2016/2017 (FINAL TEST)

COURSE TITLE: BASIC PROGRAMMING **COURSE CODE:** CSC 112
UNIT/STATUS: 2C
TIME: 1 HOUR
INSTRUCTION: ANSWER ALL QUESTIONS

1. Match the following QBASIC keywords to the appropriate category.

DIM	LOOPING
REPEAT…UNTIL	CONDITIONAL BRANCHING
IF…THEN	LOOPING
FOR…NEXT	ARRAY
GOTO	UNCONDITIONAL BRANCHING

(5marks)

2. A small error has been introduced in the code below. Debug the program.

```
CLS
10 INPUT “ENTER THE NAME OF THE AUTHOR=”;Name$
20 INPUT “Enter the title of the book=”; Title
30 INPUT “Enter the date of publication=”; Date$
40 INPUT “Number of pages=” N$
50 INPUT “Place of publication=”; Place
60 PRINT Name$, Title, Date$, Place, N$
70 END
```

(5marks)

3a. This program assigns values to the 2-dimensional numeric array B given by

$$B\begin{Bmatrix}1 & 6 & 7 & 2 & 3\\ 2 & 5 & 4 & 0 & 1\\ 1 & 4 & 3 & 8 & 7\end{Bmatrix}$$

Study and complete the program.

```
10 DIM B (      )
20 FOR I = 1 TO
30 FOR J =
40 READ B (     )
50 PRINT
60
70
80 END
```

(7marks)

3b. Suppose a= 5, x=4, y= 2, z =2; execute the following expressions in order of precedence.

(i) (x*(y +1)*z)^2 (ii) (x+z*(x/y-a^2))+a

(3marks)

4. Study the pseudocode below and answer the questions that follow.

```
BEGIN
        Ask “Length of the room?”
        Get Length
        Ask “Width of the room?”
        Get Width
        Cost = Area x 10.50
        Deposit Cost * 0.8
        Balance = Cost –Deposit
```

```
            Display “Cost is”; Cost
            Display “Deposit is”; Deposit
            Display “Balance is”; Balance
        END
```

(i) Identify the variables in the code

(ii) Explain in plain English what it does.

4b Prepare a flowchart for the pseudocode above. (10 marks)

5a. (i) Differentiate between a READ/DATA statement and INPUT statement.

(ii) Generate a QBASIC expression for the arithmetic expression below:

(a) $D = I^2 – BA - A / I$ (b) $A= (D-R)^2 / (D+R)$

5b.

```
CLS
10 REM This program inputs a friend’s personal details
20 READ NAM$, ADDR$, COMPANY$, TELNO
30 DATA “Omoyeni Saheed’, “Shell Coporation Ltd”, 08026728272
40 END
```

Rewrite the above program using the INPUT statement. (10 marks)

6.

```
10 REM
    20 INPUT "THE RADIUS IS"; R
    30  LET P = 3.142
    40  GOSUB
    50  GOSUB 110
    60  END
    70 REM “DECLARING SUBROUTINE”
    80 C = 2 * P * R
    90 PRINT "THE CIRCUMFERENCE IS="; C
    100 REVERSE
    110 A = P * R * R
    130 PRINT "THE AREA IS="; A
    140 RETURN
```

a. What is the goal of the program?

b. Debug the program and rewrite the program in a better way.

C. The following program calculates the total score in mathematics and English for five students can be written in another form. ***Rewrite*** the program in a better way and ***make a judgment*** why your solution is better.

```
10 C= 0
20 INPUT “MATHS SCORE”; MS
30 INPUT “ENGLISH SCORE”; ES
40 TOTAL = MS + ES
50 PRINT “TOTAL SCORE”; TOTAL
60 C = C+ 1
70 IF C =5 THEN 90
80 IF C < 5 GOTO 20
90 END
```

(10 marks)

7a. Write a QBASIC program for the pseudocode in question 4 above.
b. 10 REM PROGRAM TO COMPUTE FACTORIAL OF SOME NUMBERS
20 FOR I = 1 TO 15
30 READ N
40 DATA 5, 4,3,2,1
 50 FACT = NUM
60   FOR J = 1 TO 5
70   FACT = FACT * J
80   NEXT I
 90 NEXT J
100 PRINT "FACTORIAL IS="; FACT
110 DATA 5, 4,3,2,1
120 END
Trace the program and identify the line numbers with errors.

(10marks)

Goodluck!

Appendix A 15: Whole Brain Learning approach survey

Learning Approach Survey: Whole Brain Thinking

This 24-item questionnaire was designed to obtain your preferred thinking styles. The understanding of your thinking style preference will give you the opportunity to understand how you learn, make certain decisions, communicate, and why you do things the way you do it. Ensure that you answer the questions truthfully.

SECTION A: Personal information

Name:

Age: Below 15years ☐ 15-18 ☐ 19-22 ☐ 23-25 ☐
26-29 ☐ 30 and above ☐

Gender: Male ☐ Female ☐

Course combination: _____

Semester: _____ **Year of Study:** _____

SECTION B: Whole brain thinking

Instructions

1 Think about a subject/course of study you are enrolled for.

2. Answer all the following questions. ***Note:*** Your first reaction to a statement is usually the best response.

3. For each of the statements, encircle your best fitting answer:

SD- strongly disagree **(1)**	**D -** disagree **(2)**	**N -** Neutral **(3)**
A - agree **(4)**	**SA -** strongly agree **(5)**	

Please avoid the use of **neutral (3)** unless when it is very necessary.

S/N	QUADRANTS	SD	D	N	A	SA
1.	I learn a lot when the lecturer/teacher gives an informative lecture					
2	I need frequent opportunities for guided practice, feedback, and review during class					
3.	I enjoy hands-on, active instruction where I get to move around					
4.	I like to have the lecturer/teacher use visuals that show patterns, connections, and relationships					
5.	I like brief, clear and to-the-point information that is accurate and tied to expert sources.					
6.	I want instruction to be well-planned, practical, structured, and sequential.					
7.	I dislike competition in the classroom and need to feel comfortable with everyone in order to learn.					
8.	I enjoy collecting and analyzing facts, ideas, and theories.					
9.	I need checkpoints and activities to verify my understanding of material.					
10.	I like the freedom in class to explore ideas, make discoveries, and figure out things on my own.					
11.	I enjoy discussion in class so that I can share my ideas and hear what others are thinking.					
12.	I enjoy variety of classroom activities and change in classroom					
13.	I don't like a lot of open-ended sharing of feelings and opinions					
14.	I am frustrated by a lot of open-ended activities, discussion, and exploration of ideas.					
15.	I like to receive personal attention, care, and acceptance from the lecturer/teacher with smiles and good eye-contact.					
16.	I'm frustrated with a structures lecture with lots of lists and outlines.					
17.	I like it when the lecturer is in charge and clearly explains the topic.					
18.	I like content of teaching organized into lists, outlines, and categories.					
19.	I appreciate hearing personal stories from the lecturer/teacher that connect to the topic I am learning about.					
20.	It is important for me to see the big picture whenever we start a new unit of instruction.					
21.	I enjoy critical analysis and problem solving					
22.	I want the lecturer/teacher to give clear instructions, to keep the classroom orderly, and start and stop on time.					
23.	I like to get to know other students and discuss ideas with them.					
24.	It is important for me to choose how to do things and to have several options available					

Thank you for filling this questionnaire.

Appendix A 16: Whole Brain feedback questionnaire

FEEDBACK QUESTIONNAIRE: WHOLE BRAIN LEARNING

Effective learning is considered a collaborative effort between students, their peers and lecturer. The design of this questionnaire is based on the principles of learning-centeredness. Your thoughtful answers to the following items will provide helpful information to me that can help me enhance students' learning experience, and that of future students.

Name: _____ **Gender:** _____ **Quadrant:** _____

Section A: Lecturer's contribution

Describe the lecturer's contribution to learning in terms of each of the aspects addressed in the items below, using the following scale: **1 - Hardly ever 2 – Occasionally 3 - Frequently 4 - Almost always**				
Category I The lecturer inspires students by:	**1**	**2**	**3**	**4**
a. showing strong interest about the subject matter and learning tasks				
b. expressing herself well (variety in tone of voice)				
c. promoting insight in the importance and significance of the subject matter/constructs and related problems/innovations.				
d. providing learning opportunities (sessions) that are lively and encouraging				
Category II The lecturer initiates learning by:				
a. creating a climate conducive to deep learning				
b. clearly stating the purpose and learning outcomes of the session				
c. linking learning to real-life situations				
Category III The lecturer maintains learning by:				
a. promoting lecturer-student discussions/academic discourse to allow students to develop an enquiring mind				
b. encouraging students to construct own understanding and material (constructivism)				
c. providing for learning style flexibility (other ways of learning, not only according to students' own preference – a challenge beyond comfort zone)				
d. encouraging students to express themselves freely and openly				
e. encourages critical thinking and self-reflection as integral part of elf-regulated learning.				
f. creating opportunities for cooperative learning – learning in group to achieve a purpose.				

General comments

Section B: Student's contribution (based on your perceptions/observations)

Describe your own contribution to your learning in terms of each of the aspects addressed in the items below, using the following scale: **1 - Hardly ever 2 – Occasionally 3 - Frequently 4 - Almost always**				
Category I As student I contribute to my own and others' learning by:	**1**	**2**	**3**	**4**
a. showing strong interest about the subject matter and learning tasks				
b. expressing myself well (variety in tone of voice and with confidence)				
c. gaining insight in the importance and significance of the subject matter/constructs and related problems/new methods and ideas.				
d. participating in such a way that learning opportunity (sessions) become lively and encouraging.				
Category II As student I:				
a. creating co-create a climate conducive to deep learning				
b. continuously attempt linking my learning to real-life situations				
b. attempt to construct a big picture of the multidimensional nature of the session/my teaching practice				
Category III As student I:				
a. take part in lecturer-student discussions/academic discourse to allow me to develop an enquiring mind				
b. construct my own understanding and material (constructivism)				
seek opportunities for developing learning style flexibility (learning in different ways, not only to my preference.				
c. make use of opportunities to express myself freely and openly				
e. reconsider many of my former attitudes and values (concerning learning/facilitating learning, etc.)				
f. gained a better understanding of myself				
g. developed a greater sense of my responsibility				
h. contribute to my peers' learning (helping them find solutions/answers)				
i. actively take part in cooperative learning opportunities and establishing communities of practice.				

General comments

Appendix A 17: Collaborative group checklist

Behaviour/Activities	**GA**	**GB**	**GC**	**GD**	**GE**	**GF**
Brainstorming						
Group stays on task						
Group members respects each other						
Group member actively participates						

G: means group

Appendix A 18: Whole brain observation checklist (non- participant observer)

Feedback questionnaire (1st and 2nd Cycle teaching)

Effective learning is considered a collaborative effort between students, their peers and lecturer. The design of this questionnaire is based on the principles of learning-centerdness. Your thoughtful answers to the following items will provide helpful information to me that can help me enhance students' learning experience, and that of future students.

Key: S means supported; NS means not supported; NO means not observed

Category I **The lecturer inspires students by:**	**S**	**NS**	**NO**
a. showing enthusiasm about the subject matter and learning tasks			
b. expressing himself well (variety in tone of voice)			
c. promoting insight in the importance and significance of the subject matter/constructs and related problems/innovations			
d. providing learning opportunities (sessions) that are lively and encouraging			
Category II The lecturer initiates learning by:			
a. creating a climate conducive to deep learning			
b. clearly stating the purpose and learning outcomes of the session			
c. linking learning to real-life situations			
Category III The lecturer maintains learning by:			
a. promoting lecturer-student discussions/academic discourse to allow students to develop an enquiring mind			
b. encouraging students to construct own understanding and material (constructivism)			

c. providing for learning style flexibility (other ways of learning, not only according to students' own preference – challenge beyond comfort zone)			
d. encouraging students to express themselves freely and openly			
e. inculcating critical thinking and self-reflection as integral part of self-regulated learning			
f. creating opportunities for cooperative learning – establishing communities of practice			

General Comments on my teaching and student learning

Appendix A 19: Behaviour log

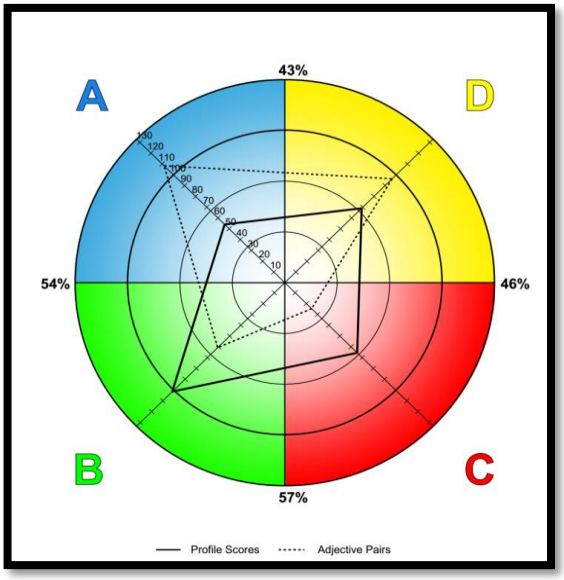
Name of student: _____

Name	Time	Behaviour

Appendix A 20: HBDI profiles of participants in AR cycle 1

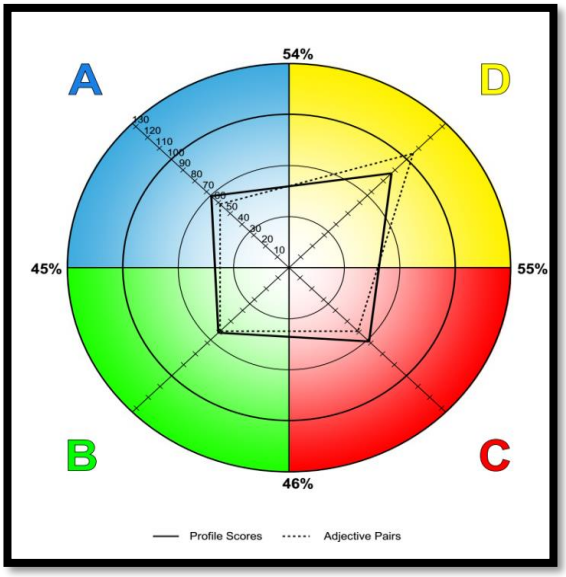
Double dominant profiles

Preference Code 2121



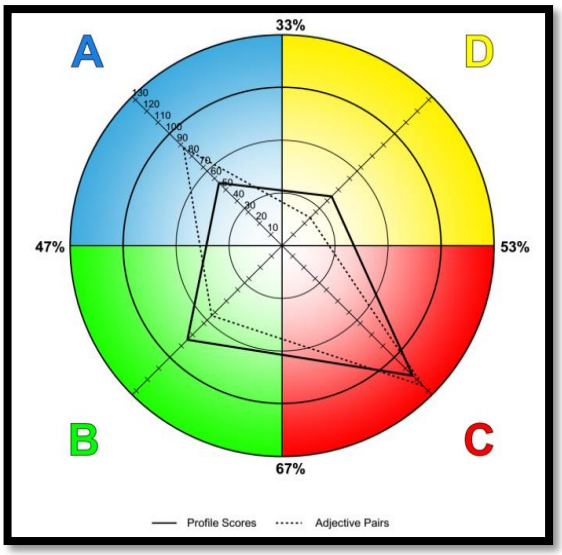
Participant 12

Preference Code 2211



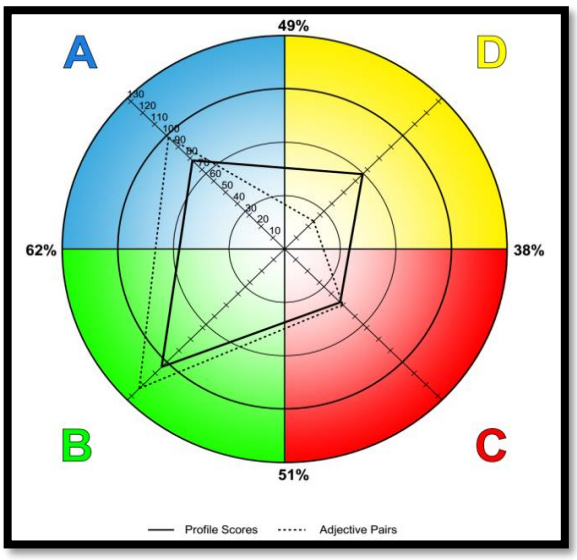
Participant 1

Preference Code: 2112



Participants 6, 9 and 10

Preference Code 1122

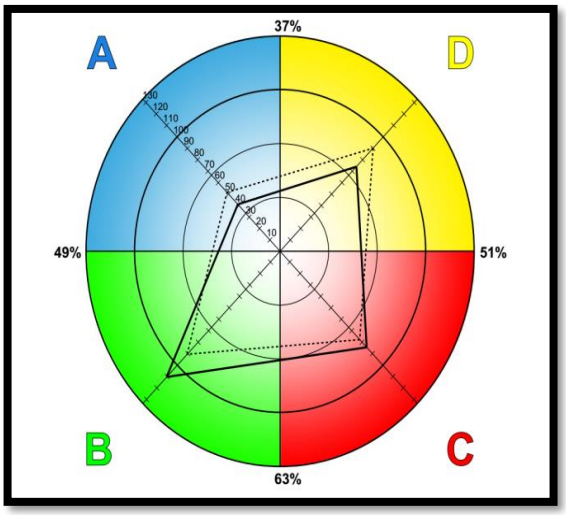


Participants 1 and 7

Source: Herrmann International (2016)

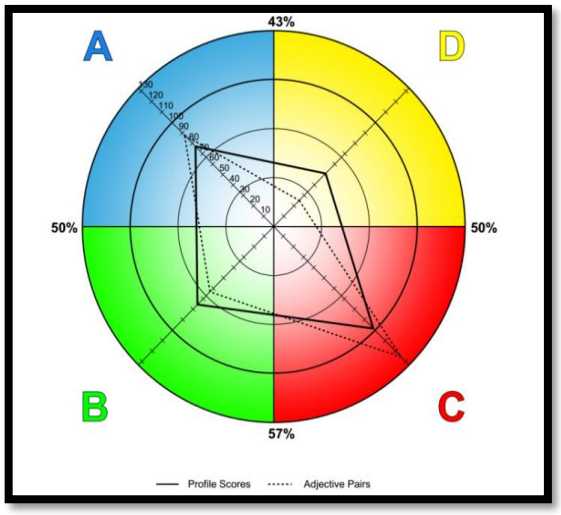
Triple dominant profiles

Preference Code 2111



Participants 2, 3, 8 and 13

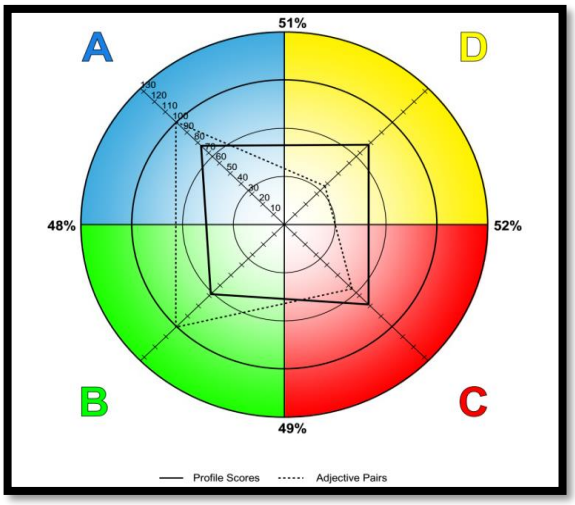
Preference code 1112



Participant 5

Quadruple profile

Preference code 1111



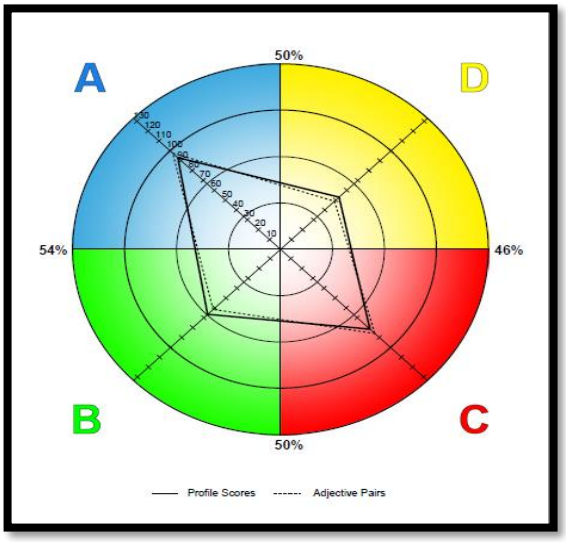
Participant 11

Source: Herrmann International (2016)

Appendix A 21: HBDI profiles of participants in AR cycle 2

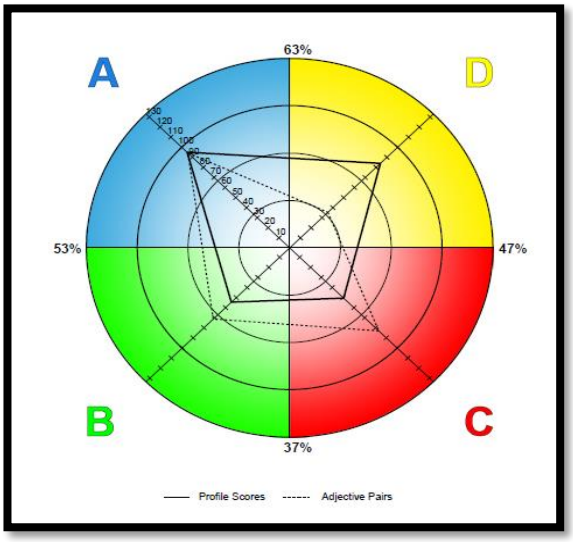
Double dominant profiles

Preference code 2 1 1 2



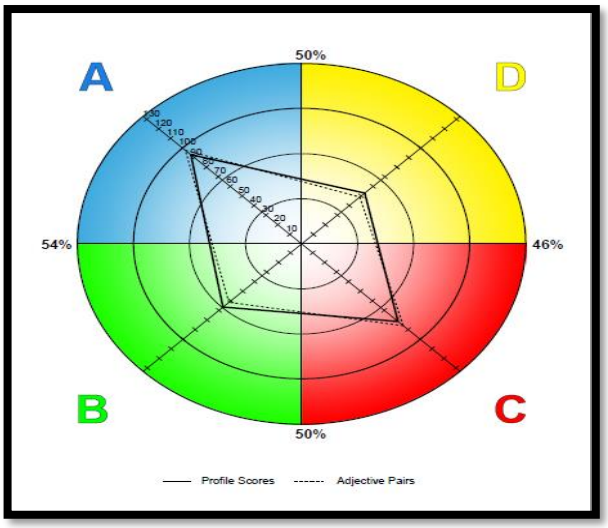
Participant 2

Preference code 1 2 2 1



Participant 5

Preference code 1212

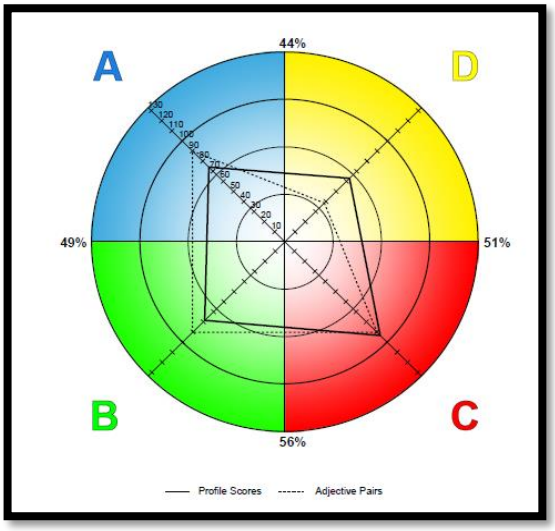


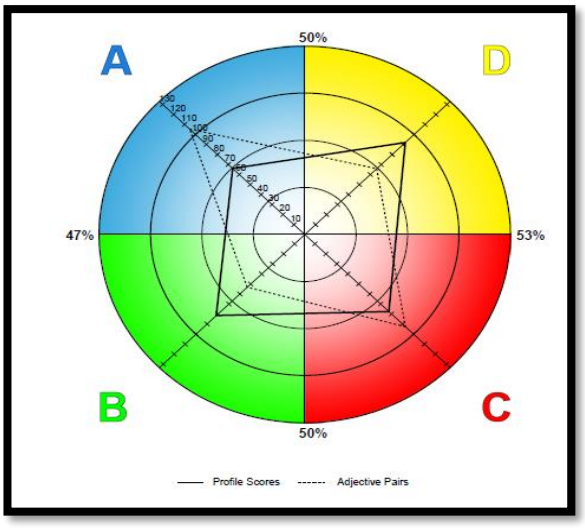
Participant 4

Source: Herrmann International, 2017

Triple dominant profiles

Preference code: 1112

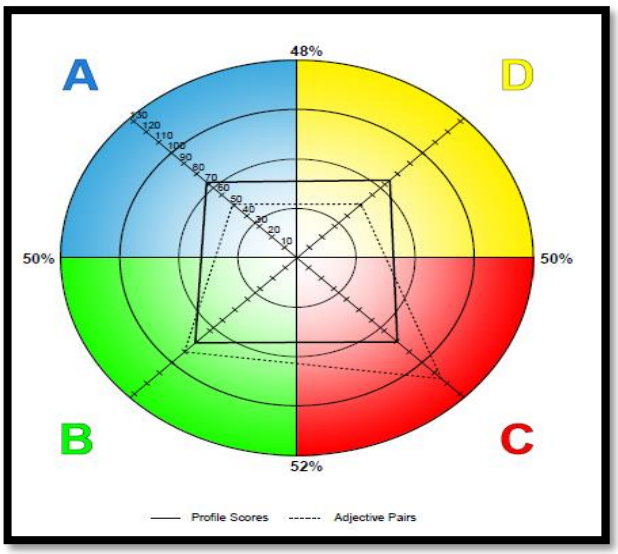




Participant 1, 3 and 7

Quadruple profile

Preference code 1111



Participants 8 and 9

Source: Herrmann International, 2017

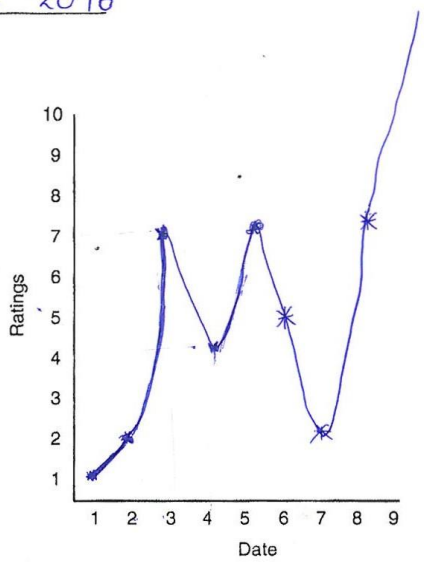
Appendix A 22: Reflective learning journal of participants

Reflective journal

Name:

Date: 13- April- 2016

Scale 1



Explain?

Date

① My first time in class, I enjoyed the class but I couldn't add up what programing had to do with drama presentation

② I was catching up and scratch was introduced as a concept of basic programing

③ Everything fits together. now I know what the drama presentation was for. we had a test which I thought I did well on Will.

④ Test result came out and I didn't do so ~~well~~ so I felt bad through out the class

⑤ A presentation class; I did my best and I enjoyed the class so much

⑥ Basic programing not so very basic for me. More like

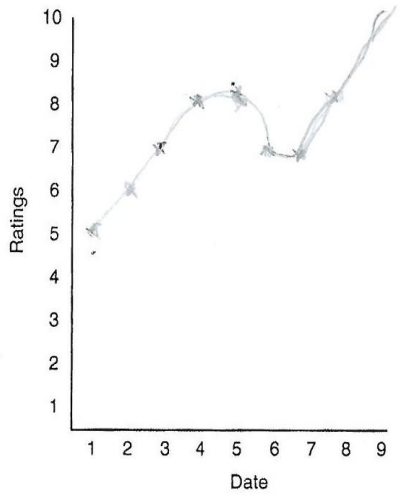
Complex programing

Reflective journal

Name:

Date: _____

Scale 1



Explain?

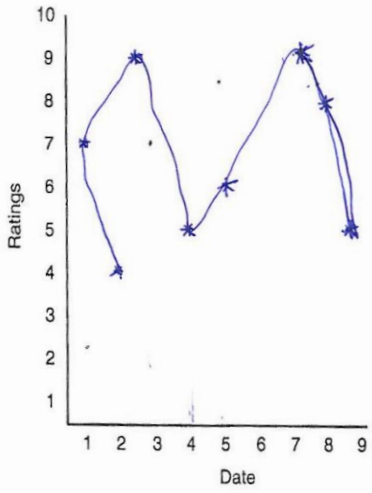
the reason of this rating is because the 27/01/2016 was because I was new to the system ~~and the topic~~ but not new to the ~~sys~~ topic. Todate 03/02/2016 is also new to me ie the topic and on 10/2/2016 it goes out of gradual personal and teaching improvement, on the 17/2/2016 I was able to understand the topic due to the practical aspect of the teaching

Reflective journal

Name:

Date: _____

Scale 1



Explain?

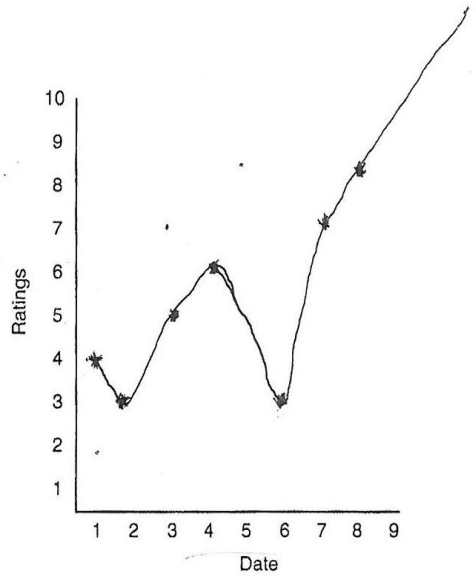
I got 7 the first day because I understood the flowchart and algorithm well but it was not easy for me to do it myself.
I got 4 the second time because I did not understand the quantities at all. ~~But I can~~ just use an example to do it.
I got 9 the third time because it was about scratch programming. It was practical and easy to understand.
The rate went down to 5 again because I do not really understand the repetition structure but I was able to have an idea of it.
The rate went up to 6 because the new topic introduced that is, BASIC was easy to understand ~~and~~ I understood it well but not perfectly.
The rate at which I understand went up to 9 again because the writing in QBASIC language was very easy and straightforward just like the normal maths BODMAS but for QBASIC it is BEDMAS.
I had 8 in the next lesson because the programming was quite easy but not very easy to get and understand. But the knowledge of the scratch help me to get it better.
In the last lesson I had 5 because I understood the control structure but I didn't know how to write and solve some questions under it.

Reflective journal

Name:

Date: 14/04/2016

Scale 1



Explain?

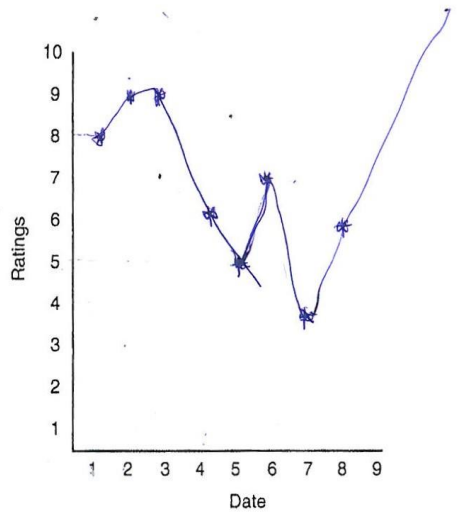
In the first week we where taught about the difference btw algorithms and a flowchart because it was the first class so I couldn't understand better On the 2nd week i understand a bit but not all The 4 & 5th i understand the cause and topic very well on the 6th lesson the basic program was new to me and i did not get much out of it. On the 7th and 8th lesson I understand the lesson very well.

Reflective journal

Name:

Date: 14/4/2016.

Scale 1



Explain?

The main why my reflective lesson log droped is because the first three Lesson not has hard as the other classes. They Other Lesson realy need thinking, so I have to go home and read and understand what I have learnt in school and thats why my reflective lesson log droped. After reading and understanding I can now apply what I have learnt to anything at anytime.

Appendix B 1: Alignment of procedural programming ILOs and assessment

Taxonomy domain	Course objectives (ILOs)	Programming ILOs and affective values	Allocated time	Assessment activities
Cognitive 1	Explain programming, algorithm, flowchart, and pseudocode in programming	**Recognize** – Base knowledge and vocabulary of the domain. **Explain** programming, algorithm and pseudocode. **Identify** flowchart symbols **Do** algorithm. **Convert** an algorithmic solution to flowchart.	4hours	Quizzes, assignments, group class-work and tests
Cognitive 2	Identify QBASIC programming data types	**Define** constants, strings and variables, operators, statements and commands. **Identify** constants, strings and variables in a code. **Convert** arithmetic expressions to QBASIC expressions. **Describe** function of a statement.	4hours	Assignment and class work, Tests
Cognitive 3	Explain the following concepts: Looping, array, table, control structure and subprogram	**Identify** the QBASIC statements of each of the following concepts: Looping, array, table, control structure and subprogram. **Explain** the function of each of the concepts. **Create** a small program using each of the following concepts statements. **Trace** programs and identify goals.	4hours	Quizzes, assignment and class work, Tests
Cognitive 4	Design and write QBASIC code to create and access data file	**Design**- devise a solution structure using knowledge of flowchart and algorithms. **Create** – write the program codes **Execute** programs. **Debug** programs.	6hours	Quizzes, assignment and class work, and Tests

Taxonomy domain	Course objectives (ILOs)	Programming ILOs and affective values	Allocated time	Assessment activities
Cognitive 5	Write error free program.	**Design**- devise a solution structure using knowledge of flowchart and algorithms. **Create** – write the program codes. **Execute** programs. **Debug** programs. **Trace** and **redesign** a program.	6hours	Quizzes, assignment and class work, group project and Tests.
Affective 6	Work productively as part of a pair/team.	Cooperate, communicate ideas, receive, respond, and value.		Assignment, group class-work and group project.
Affective 7	Demonstrate ability for organization and internalization of values.	Reflect, organize.		Class-work and journals.

Appendix B 2: Alignment of visual programming ILOs and assessment

Taxonomy	Course objectives	Programming ILOs and affective values	Allocated time	Assessment activities
Cognitive 1	Solution Development: - Introduction to algorithms and software development principles - create a solution for a problem for simple mathematical problems	**Describe** the basic concepts of an algorithm. **Define** an algorithm and develop a clear understanding **Create** algorithms for everyday problems. **Devise** an algorithm/basic instruction to complete similar tasks. **Describe** a task with a flowchart tool *Interpret* a basic flow chart	6hours	Quizzes, class discussion, assignments, practical group work, group project, tests.
	- Algorithms with decisions - Algorithms with repetition - Input, output, processing - Decision making	**Explore** algorithms such as: **Determine** smallest, largest value of more than two values **Determine** aggregates e.g. sum **Solve** basic calculations such as calculating area, volume, VAT **Determine** whether a number is even **Determine** whether a number is a factor of another number **Produce** an algorithm to solve a problem. **Create** basic flow charts/pseudocode to represent an algorithm		
Cognitive 2	Solution Development: Software Engineering Principles	**Explain** problem solving? **Discuss** problem solving steps *(Polya, G., 1957)* **Understand** the problem **Devise** a plan/algorithm **Carry** out the plan/implement the algorithm (write the code) **Look** back/test (see if it works) **Solve** a problem using the problem-solving steps	6hours	Quizzes, class discussion, assignments, group work, tests.

Taxonomy	Course objectives	Programming ILOs and affective values	Allocated time	Assessment activities
Cognitive 3	Algorithm design and programming: - Introduction to the development of solutions using a graphical programming tool - Principles of software engineering principles.	**Identify** Scratch programming terms and development environment **Identify** and *explain* the function of Scratch controls **Design** animations with costumes and sprites **Explore** interactive user interface (animations) **Explore** and *discuss* iteration constructs: [repeat, repeat until and forever] **Explore** the use of variables **Discuss** variable naming conventions **Assign** values to variables **Explore** data types: integers, strings, floats, Boolean **Explore** operators (+, - , * , /) and order of precedence **Do** basic calculations such as area, volume, VAT and simple formulae and typical calculations done **Do** string operations **Retrieve** remainders: modulus **Compare** operators **Perform** logical comparisons, **Solve** functions – random, round, square root **Solve** conditional constructs [if and if-then-else] **Apply** algorithms such as swapping values, finding aggregates, isolate digits in an integer number, finding the smallest/biggest of two numbers, **Determine** if a number is a factor of another number, **Determine** if a number is even **Extend** the use of variables, logical operators, random numbers and built-in functions **Design** event handling [When clicked, when key pressed, Broadcast and When I receive] **Design** sensing events/actions and responding programmatically **Explore** lists/arrays (storing and accessing a list of numbers and strings) and containers	8hours	Quizzes, class discussion, assignments, practical, Group work, group project, tests.

Taxonomy	Course objectives	Programming ILOs and affective values	Allocated time	Assessment activities
Affective 5	Work productively as part of a pair/team	Cooperate, communicate ideas, receive, respond, and value		Assignment, group class-work and group project
Affective 6	Demonstrate ability for organization and internalization of values	Reflect, organize		Class-work and journals

Appendix B 3: Classification of formative test questions (1st AR cycle)

Assessments	SOLO	Revised Blooms	Type of programming skill for each question				Block model		Total no questions
		Cognitive dimension	Tracing skill	Explaining skill	Reading skill	Writing skill	Level	Comprehension dimension	
Test of individual concepts	Unistructural Unistructural Unistructural Multistuctural Unistructural Unistructural Multistructural	Remembering Understanding Remembering Applying Understanding Understanding Applying	 x x	X (Q1) x x x x x x	 	 x			10
Interim test 1	Relational Relational Relational Relational	Applying Applying Creating Applying	x x 	 	 x	 x 			4
Interim test 2	Unistructural Unistructural Unistructural Unistructural	Understanding Understanding Applying Applying	 x 	x 	 	 x x			
Interim test 3	Relational	Creating				x (4)			4

Assessments	SOLO	Revised Blooms	Type of programming skill for each question				Block model		Total no questions
Final test	Unistructural	Understanding	x						13
	Unistructural	Remembering		x					
	Relational	Applying	x						
	Multistructural	Understanding		x					
	Relational	Applying	x						
	Relational	Applying	x			x			
	Relational	Applying		x		x			
	Relational	Applying			x				
	Relational	Creating			x				
	Multistructural	Applying			x				
		Creating			x				

Appendix C 1: Theme Construction for the AR Cycle one and two

Themes	Subthemes	Categories	Codes	Acronym	AR Cycle 1		AR Cycle 2	
					Source	Number of occurrence	Source	Number of occurrence
Theme 1 Strategies for teaching and learning programming	Teaching strategies	Whole brain teaching strategies	Quadrant grouping aided learning knowledge of self Whole brain boost thinking Whole brain teaching B-aided decision making Knowledge of group characteristics Presentation	KOS QG WT WBT DM KGS P	INT INT INT OBS, WBF, INT - - OBS, INT, REF	6 4 1 10 - - 13	INT INT INT OBS INT INT REF, INT	4 3 2 2 1 1 4
		Constructivist teaching strategies	Student engagement Group learning Scaffolding by teacher Scaffolding by student Pair programming Teacher feedback Project Assignment Real life application Revision Individual learning Cooperative learning “Practical and explanatory terms” Clear explanatory terms Practical Research work Critical thinking	SE GPL SCT SCS PP TF PW AS RLAP REV IL CL PET CEF PRTL RW CT	OBS OBS, WBF OBS, INT OBS - OBS OBS OBS WBF OBS OBS OBS, INT - - REF INT WBF	12 2 11 2 - 2 2 6 1 3 2 5 - - 2 2 1	OBS, WBF OBS, INT, WBF OBS, WBF OBS, WBF OBS OBS OBS INT INT REF, INT OBS OBS WBF REF REF REF INT -	23 12 11 6 6 3 3 3 3 2 2 2 1 1 1 1 -

		Grouping benefits and negative impact on learning	Group cooperation Group negative impact No group cooperation Group members attitude Group behaviour Group struggle Student cooperation Group displeasure Coping strategy	GRC GNI NGRC GMA GB GS STC GD CS	OBS - OBS - OBS - OBS - -	4 - 3 - 1 - 1 - -	OBS INT, REF, WBF OBS, INT, REF INT INT, OBS INT OBS OBS INT	9 9 7 4 3 2 2 2 2
		Students perspectives and preferences for Pair programming	Perspectives about pair programming Prefer pair programming Pair cooperation Ideas not respected in pair	PAPP PPP PC INRP	- - - -	- - - -	INT INT OBS, INT INT	7 4 3 2
		Teacher's personality	Teacher's qualities Expected teacher's qualities	TQ ETQ	WBF, INT INT	6 2	WBF, REF INT	10 4
	Strategies students used for the learning of programming	Cognitive learning strategy	Meaning construction Accommodation Disequilibrium Discovery learning	MC ACM DSE DL	OBS - OBS -	6 - 3 -	OBS OBS OBS OBS	14 3 2 1
		Self-regulation of learning	Positive emotional climate Regulation of cognition Focused goal attainment Seeking help Not valuing learning Knowledge transfer Valuing learning Challenge myself Hardworking Regulation of behaviour Multidimensional view	PEC RGC FGA SH NVL KT VL CM H RGB MTDM	OBS OBS - OBS - - OBS, REF - - - WBF	3 2 - 7 - - 8 - - - 1	OBS OBS OBS, INT OBS, INT OBS OBS OBS WBF WBF OBS -	9 6 6 5 1 1 1 1 1 1 -

		Social interaction during learning	Social interaction Student contribution Expression skill Providing solution Group interaction Sharing ideas	SI SC EPS PRS GI SHI	OBS, INT, WBF, REF - - - - -	12 - - - - -	OBS, INT OBS INT, QWF OBS QWF, INT INT, REF	17 16 5 4 3 2
		Problem-solving	Brainstorming Explaining Problem-solving Group dynamics	BRST EXPL PBS GDM	OBS - - -	5 - - -	OBS INT, OBS OBS, WBF, INT OBS	14 4 4 2
Theme 2 Programming knowledge gained by students	Programming knowledge gained by students	Perspectives about Scratch and QBASIC programming	QBASIC easy with Scratch Scratch is easy Scratch and QBASIC similarity Scratch application in QBASIC Enjoyable Perspectives about QBASIC Fun Perspectives about programming Scratch motivates Scratch as foundation	QBS SIE SQS SEQ E PAQB F PAAP SCM SFDN	INT, REF INT, REF INT INT INT INT, REF INT INT INT INT	7 2 6 7 6 7 3 3 3 3	INT INT, REF INT INT INT INT INT INT, REF - - -	4 4 4 4 3 3 2 - - -

		Knowledge gained in programming	Scratch understanding Knowledge of concepts Knowledge of flowchart algorithm Running of programs QBASIC understanding Knowledge of variables Knowledge of repetition Knowledge of control structure Knowledge of expression	SU KOC KOFA ROP QBU KOV KOR KOCS KOE	INT - OBS, INT REF INT INT, REF OBS REF REF OBS	2 - 9 1 8 2 1 1 2	REF REF OBS, INT INT REF, INT OBS, INT OBS, REF OBS, REF -	19 16 15 10 5 2 2 2 -
		Previous experience aided programming knowledge gained	Previous experience Previous knowledge “I am a math student” Mathematics background	PE PK MS MB	REF - - -	1 - - -	REF, OBS, INT REF REF OBS	5 2 2 1
		Debugging skills	Debugging skills Error detection Lack of debugging skills Scratch encourage debugging In QBASIC	DS ED LDS SED	- OBS - -	- 1 - -	OBS, INT OBS, INT INT INT	12 4 3 2
		Word processing skills	Word processing skill Lack of word processing skills	WPS LWPS	OBS, INT -	5 -	OBS, INT RINT	2 1
	Impact of programming knowledge on students	Programming aids critical thinking	Thinking Programming involves thinking Critical thinking Scratch aided thinking in QB “No serious thinking” Thinking ahead Relax and think	T PRGT CT SAT NST TAH RT	- INT INT - - - -	- 3 1 - - - -	OBS, WBF, INT INT INT INT INT INT INT	5 3 3 1 1 1 1
		Scratch builds program writing skills	Program writing Scratch helped to learn order precedence Program arrangement	PW SHOP PA	- - INT	- - 1	OBS, INT INT INT	7 3 3

		Scratch boosts interests	Scratch boosts interest QBASIC interesting with Scratch	SBI QIS	INT -	1 -	INT INT	4 3
		Scratch encouraged Creativity	Creativity	CRT	-	-	OBS	3
		Programming impact on students	Programming impact Love to be a programmer Programming aids planning	PI LTP	INT, REF, WBF INT INT	7 1 1	INT INT -	11 1 -
		Scratch inclusion to the curriculum	Scratch inclusion Curriculum adjustment “Programming as a major course” Scratch emphasis	SCI CA PMJ SE	INT - INT INT	6 - 1 1	INT INT INT INT	2 1 1 1
	Impact of student factors on the programming knowledge	Student’s strengths in Programming and challenges faced	Learning challenge Student strength Scratch challenges Challenge with new experience Extra classes	LC SS SC CNE EXTC	OBS, INT, REF OBS, INT INT - INT	14 2 8 - 3	OBS REF OBS INT RINT -	8 6 4 2 -
		Student behaviour	Undesired behaviour Student behaviour Lateness Desired behaviour Negligence to assignment Group behaviour	UDB SB LT DB NGA GB	- OBS OBS - OBS OBS	- 23 8 - 1 1	OBS OBS OBS OBS OBS OBS	11 8 4 2 1 1
		Student well-being	Student health Student personal issues Student mood	SHLT SPI SMD	OBS - REF	1 - 4	OBS, REF REF REF	7 2 2
	Contextual problem as impediment to the teaching and learning of programming	Contextual problem	Electricity problem Faulty computers Contextual factor No enough practical Limited computers Projector fault Student enrolment Clashing of courses Whiteboard	EP FC CTF NEP LC PF NERT CLC WB	OBS, INT INT - - - - OBS OBS OBS	6 1 - - - - 3 2 1	OBS, REF OBS OBS INT OBS OBS - - -	4 4 1 1 1 1 - - -

		Suggested solution to contextual problem	Solution regarding practical Provision of computers Possession of a PC by student Improved laboratory environment New programming laboratory Whiteboard Provision of electricity	SRP PRC PPS ILE NPL WB PEL	- INT - - - - INT	- 1 - - - - 1	INT INT INT INT INT INT INT	9 3 3 3 2 2 2
Theme 3 Mental representation and affective states during programming	Mental representation of the Block Model	Atoms	Text surface Functions Program execution	TXTSF FUNC PREXC	INT INT INT	4 1 4	INT INT -	1 1 -
		Macro-structure	Text surface	TXTSF	INT	6	INT	1
		Block	Functions	FUNC	INT	2	-	-
		Relations	Program execution	PEXC	-	-	INT	1
	Student's affective and behavioral states during programming writing	Affective states (Confusion)	Confusion "Did I get it or not" "How do I follow this step" "What did I press" I don't" "Can I put them together" "I don't know" "I don't know where" "How am I" "Should I" "Will I"		INT INT INT INT INT INT INT INT INT INT INT	1 - - - 1 - 3 1 1 1 1	INT INT INT INT INT INT INT INT INT INT INT	6 1 1 1 2 1 1 1 - - -
		Affective states (Figure it out)	Figure it out "thinking of the way" "Looking at it closely "Checking the program" Searching for display "looking for a suitable topic" "I went for help" "I remembered" Trial		INT INT INT INT INT INT INT INT INT	1 - - - - - - 2 3	INT INT INT INT INT INT INT INT INT	3 1 1 1 1 1 1 1 -
		Behavioral states	Wiping of face Moving of slips Changing position Frowned face Stood up Bored Shaking of hands		INT INT INT INT INT INT INT	- - - - - - 1	INT INT INT INT INT INT INT	1 1 1 1 1 1 -

Appendix C 2: Comprehensive observation codes for all lessons in cycle one

Lesson and Dates	Codes	Acronym	Number of occurrence	Categories
Lesson 1: 20/01/2016	Seeking help	SH	1	Self-regulation
	Assignment Cooperative learning	AS CL	1 1	Constructivist learning strategies
	Student behaviour	SB	3	Student behaviour
	Meaning construction	MC	1	Constructivist learning strategy
	Social interaction	SI	2	Social interaction
Lesson 3: 3/2/2016	No learning material	NLM	1	Student behaviour
	Learning challenge	LC	2	Learning strength and challenge
	Meaning construction	MC	1	Constructivist learning strategy
	Electricity problem	EP	1	Contextual problem
	Knowledge of flowchart algorithm	KFA	3	Scratch and QBASIC knowledge gained
	Cooperative learning Assignment Group learning	CL AS GRPL	3 1 1	Constructivist teaching strategy
	Whole brain teaching	WBT	2	Whole brain teaching strategy
	Seeking help Positive emotional climate Regulation of cognition	SH PEC RGC	1 1 1	Self-regulation
	Student behaviour Lateness	SB LT	1 1	Student behaviour
Lesson 4: 10/2/2016	New enrollment	NERT	1	Contextual problem
	Whole brain teaching	WBT	1	Whole brain teaching strategy
	Scaffolding by teacher	SCT	2	Scaffolding
	Meaning construction	MC	1	Constructivist learning strategy
	Student engagement Revision	SE REV	3 1	Constructivist teaching strategy
	Student behaviour	SB	5	Student behaviour
	Brainstorming	BRST	1	Problem solving
	Learning challenge	LC	1	Learning challenge
Lesson 5: 27/1/2016	Electricity problem	EP	1	Contextual problem
	Social interaction	SI	1	Social interaction
	Scaffolding by teacher Scaffolding by student Learning at ZPD	SCT SCS LZPD	4 3 1	Scaffolding
	Student engagement Presentation	SE P	2 2	Constructivist teaching strategy
	Seeking help Positive emotional climate	SH PEC	1 1	Self-regulation
	Meaning construction	MC	1	Constructivist learning strategy
	Learning challenge	LC	1	Learning challenge
	Student health	SHLT	1	Student well-being
	Knowledge of variables	KOV	2	Scratch and QBASIC knowledge gained
	Student behaviour Group behaviour	SB GB	1 1	Student behaviour
	Whole brain teaching	WBT	1	Whole brain teaching
	Scaffolding by teacher	SCT	1	Scaffolding

Lesson 7: 2/3/2016	Presentation Student engagement	P SE	1 1	Constructivist teaching strategy
	Knowledge of variables	KOV	1	Scratch and QBASIC knowledge gained
	Brainstorming	BRST	2	Problem solving
	Student behavior Negligence to assignment Lateness	SB NGTA LT	4 1 1	Student behaviour
	Clashing of courses Electricity problem	CLCS EP	1 1	Contextual problem
	Valuing learning	VL	2	Self-regulation
	Social interaction	SI	2	Social interaction
	Learning challenge Learning strength	LC LS	3 1	Learning strength and challenge
	Meaning construction Disequilibrium	MC DSEQ	1 1	Constructivist learning strategy
Lesson 8: 9/3/2016	Lateness Student behaviour	LT SB	2 1	Student behaviour
	New enrolment	NERT	1	Contextual problem
	Regulation of cognition Valuing learning Seeking help Positive emotional climate	RCG VL SH PEC	1 1 1 1	Self-regulation
	Meaning construction	MC	1	Constructivist learning strategy
	Student engagement Individual learning Revision	SE IL REV	2 1 1	Constructivist teaching strategy
	Group cooperation No group cooperation	GRC NGRC	1 1	Grouping benefits and negative impact
	Scaffolding by teacher	SCT	2	Scaffolding
	Brainstorming	BRST	2	Problem solving
Lesson 9: 23/3/2016	Lateness Student behaviour	LT SB	2 1	Student behaviour
	Learning challenge	LC	2	Learning challenge
	Lesson summary	WBT	1	Whole brain teaching
	Extra classes	EXTCL	1	
	Presentation Assignment	P AS	1 1	Constructivist teaching strategy
	Group cooperation No group cooperation Student cooperation	GRPC NGRC STC	3 1 1	Grouping benefits and negative impact
	Seeking help Valuing learning	SH VL	2 2	Self-regulation
Lesson 10: 30/3/2016	Disequilibrium	DSE	1	Constructivist learning strategy
	Clashing classes Electricity problem	CLC EP	1 1	Contextual problem
	Student engagement Revision	SE REV	1 1	Constructivist teaching strategy
	Whole brain teaching	WBT	1	Whole brain teaching
	No group cooperation	NGRC	1	Grouping benefits and negative impact
	Student behaviour	SB	5	Student behaviour
Lesson 11: 1/4/2016	No learning material Truancy	NLM TRU	1 1	Student behaviour
	Knowledge of expression	KOE	1	Programming knowledge gained

	Assignment	AS	1	Constructivist teaching strategy
Lesson 12: 2/4/2016	Knowledge of flowchart algorithm Knowledge of expression	KOFA KOE	1 1	Programming knowledge gained
	Student behaviour Lateness	SB LT	5 2	Student behaviour
	Student engagement	SE	3	Constructivist teaching strategy
	Whiteboard Electricity problem	WB EP	1 1	Contextual problem
	Scaffolding by teacher	SCT	1	Scaffolding
Lesson 13: 7/4/2016	Individual learning Assignment	IL AS	1 2	Constructivist teaching strategy
	Scaffolding by teacher	SCT	1	Scaffolding
	Disequilibrium	DEQ	1	Constructivist learning strategy
	Word processing skill	WPS	1	Word processing skill
	Error detection	ED		Debugging skills
	Bored Stood up	BRD STDU	1 1	Affective state and behaviour states
	Knowledge of variables	KOV		Programming knowledge gained

Appendix C 3: Coding analysis of all classroom observations in cycle two

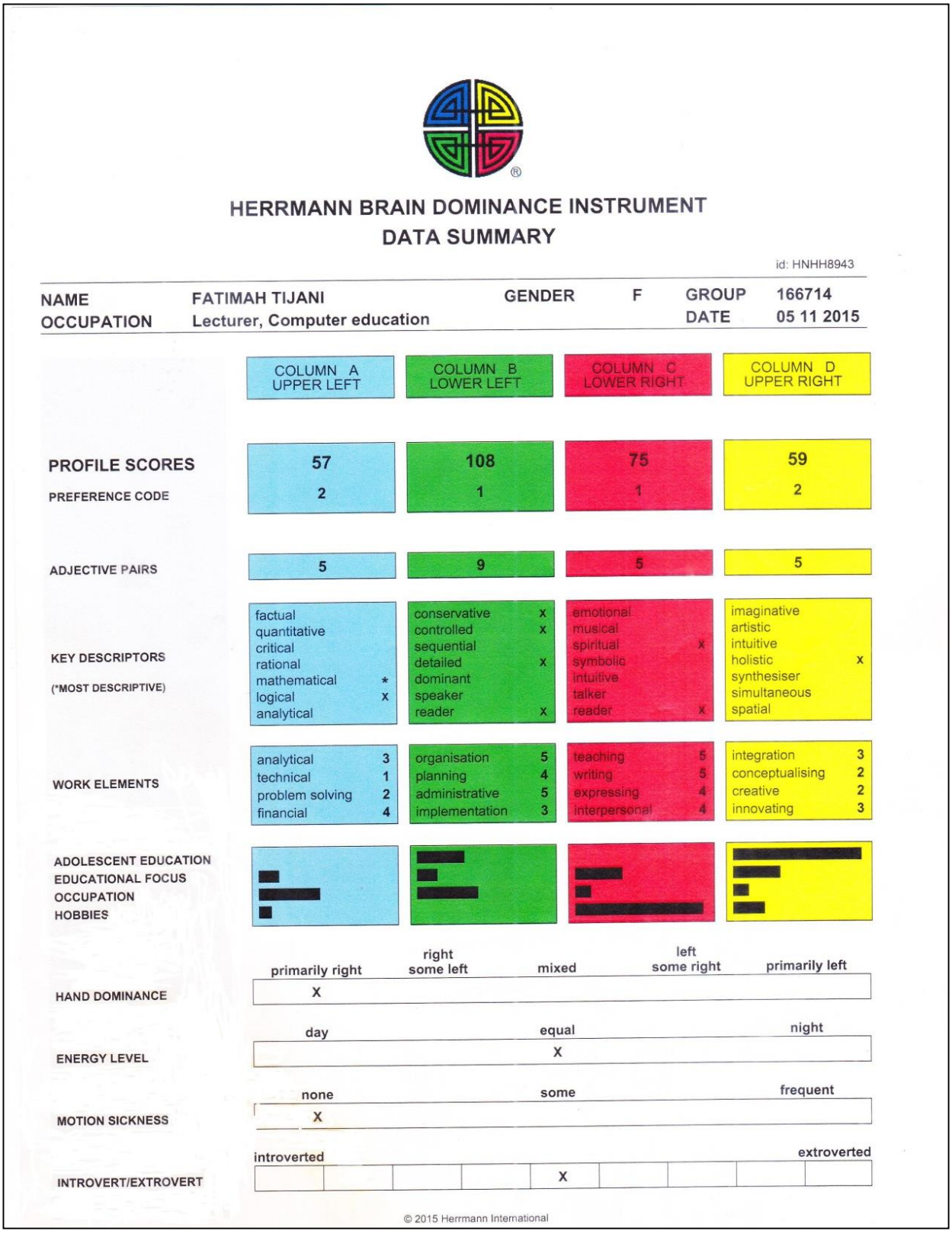
Lesson and Dates	Codes	No of occurrence	Acronym	Categories
Lesson 1: 17/01/2017	Electricity problem	1	EP	Contextual factor
	Teacher feedback Real life examples Student engagement Group learning	1 1 2 1	TF RLE SE GPL	Constructivist teaching strategy
	Brainstorming Problem-solving	1 1	BRST PBS	Problem-solving
	Desired behaviour Undesired behaviour	1 1	DB UDB	Student behaviour
	Knowledge of variables Knowledge of repetition	1 1	KOV KOR	Programming knowledge
	Previous experience	1	PE	Previous experience
Lesson 2: 24/01/2017	Group learning	1	GPL	Constructivist teaching strategy
	Brainstorming	1	BRST	Problem-solving
	Critical thinking	1	CT	Critical thinking
Lesson 3: 25/01/2017	Student engagement	2	SE	Student engagement
	Brainstorming	1	BRST	Problem-solving
Lesson 5: 1/02/2017	Electricity problem	1	EP	Contextual problem
	Group cooperation No group cooperation	3 2	GRC NGRC	Group cooperation
	Scaffolding by teacher	1	SCT	Scaffolding
	Student engagement Individual learning	1 1	SE IL	Constructivist teaching strategy
	Group behaviour	1	GB	Group behaviour
	Meaning construction	1	MC	Meaning construction
	Knowledge of flowchart algorithm	1	KF	Programming knowledge
Lesson 6: 03/02/2017	Regulation of behaviour Focused goal attainment	1 1	RGB FGA	Self-regulation
	Student strength Learning challenge	4 5	SS LC	Learning strength and challenge
	Undesired behaviour Negligence to assignment	3 1	UDB NGA	Student behaviour
	No group cooperation Group cooperation	1 3	NGRC GRC	Group cooperation
	Knowledge of flowchart algorithm	2	KFA	Programming knowledge
Lesson 7: 08/02/2017	Seeking help Positive emotional climate	1 1	SH PEC	Self-regulation
	Pair programming Individual learning	1 1	PP IL	Constructivist teaching strategy
	Scaffolding by teacher	2	SCT	Scaffolding
	Social interaction	4	SI	Social interaction
	Meaning construction	1	MC	Meaning construction
	Limited computers Projector fault Faulty computers	1 1 3	LC PF FC	Contextual factors
	Lesson summary	2	LSM	Whole brain teaching
Lesson 8: 10/02/2017	Focused goal attainment Not valuing learning	1 1	FGC NVL	Self-regulation
	Knowledge of flowchart algorithm Knowledge of variable	9 5	KFA KOV	Programming knowledge

	Scaffolding by teacher	2	SCT	Scaffolding
	Student contribution	2	SC	Student contribution
	Group learning Student engagement	1 1	GPL SE	Constructivist teaching strategy
	Undesired behaviour	1	UDB	Student behaviour
Lesson 9: 15/02/2017	Positive emotional climate	1	PEC	Self-regulation
	Previous experience	1	PE	Previous experience
	Brainstorming	2	BRST	Brainstorming
	Faulty computers	1	FC	Contextual factor
	Pair programming Student engagement Meaning construction	1 1 1	PP SE MC	Constructivist learning strategy
	Scaffolding by teacher Scaffolding by student	1 1	SCT SCS	Scaffolding
Lesson 10: 17/2/2017	Teacher feedback Meaning construction	1 1	TF MC	Constructivist teaching strategy
	Brainstorming Explaining	1 1	BRST EXPL	Problem-solving
	Group displeasure	2	GD	Grouping issues
	Positive emotional climate	1	PEC	Self-regulation
	Learning challenge	1	LC	Learning challenge
Lesson 11: 22/2/2017	Social interaction	1	EXPL	Social interaction
	Lateness Undesired behaviour No learning material	1 1 1	LT UDB NLM	Student behaviour
	Meaning construction Accommodation Discovery learning	1 1 1	MC AC DL	Learning strategy
	Project work Pair programming	1 1	PW PP	Constructivist teaching strategy
	Brainstorming Explaining	1 1	BRST EXPL	Problem-solving
Lesson 12: 24/2/2017	Revision Student engagement	1 1	TS SE	Constructivist teaching strategy
	Knowledge of variable Program writing	1 1	KOV PW	Programming knowledge
Lesson 13: 1/3/2017	Lateness Undesired behaviour	2 1	LT UDB	Student behaviour
	Meaning construction	6	MC	Meaning construction
	Pair programming	1	PP	Pair programming
	Social interaction Providing solution	1 4	SI PS	Social interaction
	Thinking	3	T	Thinking
	Learning challenge	1	LC	Learning challenge
	Pair cooperation	1	PC	Pair cooperation
	Error detection	2	ED	Debugging skills
	Scaffolding by student	1	SCS	Scaffolding
	Unclear handout	1	UCCLH	Unclear handout
	Explaining	1	EXPL	Problem-solving
Lesson 14: 8/3/2017	Student strength Learning challenge	2 1	LS LC	Learning strength and challenge
	Scaffolding by teacher Scaffolding by student	1 1	SCT SCS	Scaffolding
	Creativity	3	CR	Creativity
	No group cooperation Student cooperation	2 1	NGRC GR	Student cooperation

	Knowledge transfer Positive emotional climate	1 2	KT PEC	Self-regulation
	Critical thinking	1	CT	Critical thinking
	Student contribution	3	SC	Social interaction
	Student behaviour Group behaviour	2 1	SB GB	Student behaviour
	Knowledge of repetition	1	KOR	Knowledge of repetition
Lesson 15: 10/3/2017	No group cooperation Student cooperation	1 1	NGRC SC	Group cooperation
	Brainstorming Explaining	3 1	BRST EXPL	Problem-solving
	Disequilibrium Accommodation Meaning construction Group dynamics	1 1 1 1	DSE AC MC GDM	Learning strategy
	Undesired behaviour Lateness	1 1	UDB LT	Student behaviour
	Revision Student engagement	1 2	TS SE	Constructivist learning strategy
	Valuing learning Positive emotional climate	1 1	VL PEC	Self-regulation
Lesson 16: 17/3/2017	Student engagement	1	SE	Constructivist teaching strategies
	Student contribution	1	SC	Social interaction
	Group dynamics Brainstorming	1 1	GDM BRST	Problem-solving
Lesson 17: 18/3/2017	Student engagement	4	SE	Constructivist teaching strategy
	Student contribution Social interaction	1 2	SC SI	Social interaction
	Problem-solving	1	PBS	Problem solving
	Program writing		PW	Programming knowledge
	Debugging skills	1	DS	Debugging skill
	Disequilibrium Accommodation Meaning construction	1 1 1	DSE AC MC	Learning strategy
	Learning challenge	1	LC	Learning challenge
Lesson 18: 24/3/2017	Teacher feedback	1	TF	Constructivist teaching strategy
	Knowledge of control structure Mathematics background Knowledge of flowchart algorithm Program writing	1 1 2 1	KCS MB KFA PW	Programming knowledge
	Regulation of cognition	1	RGC	Regulation of cognition
	Student behaviour Undesired behaviour	1 1	SB UDB	Student behaviour
Lesson 19: 29/3/2017	Regulation of cognition Seeking help Positive emotional climate Focused goal attainment	1 2 1 3	RGC SH PEC FGA	Self-regulation
	Scaffolding by student Scaffolding by teacher	3 1	SCS SCT	Scaffolding
	Debugging skills	5	DS	Debugging skills
	Brainstorming	1	BRST	Problem-solving
	Student engagement	2	SE	Constructivist teaching strategy
	Meaning construction	1	MC	Learning strategy

	Word processing skill	1	WPS	Word processing skill
	Student behaviour	1	SB	Student behaviour
Lesson 20: 31/3/2017	Student engagement	3	SE	Student engagement
	Social interaction Social contribution	1 2	SI SC	Social interaction
	Debugging skills	1	DS	Debugging skill
	Student health	1	SHLT	Student health
	Scaffolding by teacher	1	SCS	Scaffolding
Lesson 21: 5/3/2017	Pair programming	1	PP	Pair programming
	Undesired behaviour Student behaviour	2 1	UDB SB	Student behaviour
	Brainstorming	1	BRST	Brainstorming
Lesson 22: 7/3/2017	Student engagement Pair programming	4 1	SE PP	Constructivist teaching strategy
	Meaning construction	1	MC	Learning strategy
	Scaffolding by student Scaffolding by teacher	1 1	SCS SCT	Scaffolding
	Social interaction Student contribution	1 8	SC SI	Student contribution
	Regulation of cognition Positive emotional climate	3 2	RCG PEC	Self-regulation
	Debugging skills	2	DS	Debugging skill
	Contextual factor Faulty computer	1 1	CF FC	Contextual factor
	Student behaviour	2	SB	Student behaviour
	Group cooperation	1	GC	Group cooperation
	Student health	1	SHLT	Student health

Appendix C 4: My HBDI data summary sheet



HERRMANN BRAIN DOMINANCE INSTRUMENT
DATA SUMMARY

id: HNHH8943

NAME	FATIMAH TIJANI	GENDER	F	GROUP	166714
OCCUPATION	Lecturer, Computer education			DATE	05 11 2015

	COLUMN A UPPER LEFT	COLUMN B LOWER LEFT	COLUMN C LOWER RIGHT	COLUMN D UPPER RIGHT
PROFILE SCORES	57	108	75	59
PREFERENCE CODE	2	1	1	2
ADJECTIVE PAIRS	5	9	5	5
KEY DESCRIPTORS (*MOST DESCRIPTIVE)	factual quantitative critical rational mathematical * logical x analytical	conservative x controlled x sequential detailed x dominant speaker reader x	emotional musical spiritual x symbolic intuitive talker reader x	imaginative artistic intuitive holistic x synthesiser simultaneous spatial
WORK ELEMENTS	analytical 3 technical 1 problem solving 2 financial 4	organisation 5 planning 4 administrative 5 implementation 3	teaching 5 writing 5 expressing 4 interpersonal 4	integration 3 conceptualising 2 creative 2 innovating 3
ADOLESCENT EDUCATION EDUCATIONAL FOCUS OCCUPATION HOBBIES				

	primarily right	right some left	mixed	left some right	primarily left
HAND DOMINANCE	X				

	day		equal		night
ENERGY LEVEL			X		

	none		some		frequent
MOTION SICKNESS	X				

	introverted								extroverted
INTROVERT/EXTROVERT					X				

Source: Herrmann International (2015)

Appendix C 5: The TPLF Form

Peer evaluation of the teaching and learning process framework (TLPF) for programming

1. Thank you for accepting to evaluate the TLPF for programming.
2. Please supply your views on this form with extensive description.
3. Your constructive criticisms and/or support for the TLPF will be appreciated (Please do not supply a Yes or No answer).
4. After filling the form, click submit.

*Required

The TLPF for programming teaching.

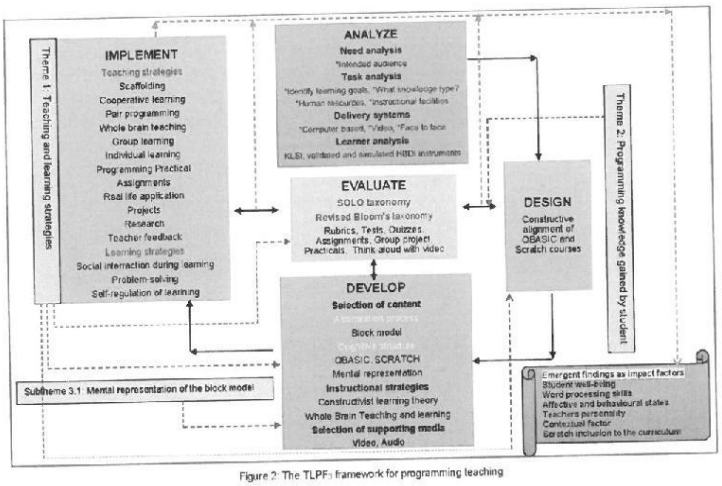


Figure 2: The TLPF framework for programming teaching

1. What do you think about the phases of the TLPF? *

2. **What are your views about the teaching and learning strategies in the new TLPF for programming?** *

3. **What value do you think theTLPF will contribute to the teaching of programming in Nigerian colleges of Education?** *