

UNIVERSITY OF KWAZULU-NATAL

**An exploration of the teaching and learning of Information Technology (IT)  
programming in a higher education institution in KwaZulu-Natal (KZN)**

by

Thamotharan Prinavin Govender

Student no. 891288849

A thesis submitted for the degree of Doctor of Philosophy in the School of Science,  
Mathematics and Technology Education (SSMTE), University of KwaZulu-Natal,  
Edgewood Campus.

Supervisor

Professor V. Mudaly

December 2019

## **Preface**

Three conference papers emanated from this study:

Paper 1 entitled *Teaching and Learning Computer Programming to first-year Information Technology (IT) students at a University of Technology in KZN: a grounded theory of its academic* was presented at the Global Trends and Challenges in Research and Innovation Conference, hosted by DUT from 17 to 20 September 2019 at the Hilton, Durban, South Africa.

Paper 2 entitled *Software Tools used by an IT educator to collaborate & stay organized in an Introductory Programming Course at a University of Technology* was presented at the 4<sup>th</sup> Annual International Conference on Scholarship of Teaching and Learning (SoTL) in Higher Education, hosted the Central University of Technology, Bloemfontein, Free State, South Africa from the 24 to 25 October 2018.

Paper 3 was presented at the 4th Teaching and Learning Conference, which was hosted by the Teaching and Learning Centre University of Zululand at the Tusk Umfolozi Casino Hotel from 9 to 11 October 2019. The paper was entitled: *Teaching and Learning Computer Programming to First-year Information Technology (IT) Students at a University of Technology in KZN: The role of visual programming in introducing programming concepts to first-year students.*

## Declaration

I, Thamocharan Prinavin Govender, declare that the research reported in this thesis, except where otherwise indicated, is my original work, and has not been submitted for any degree or examination at any other university.

---

T.P Govender

---

Date

---

Supervisor

---

Date

## Ethical Clearance – UKZN



15 August 2017

Mr Thamocharan Prinavin Govender (891288849)  
School of Education  
Edgewood Campus

Dear Mr Govender,

**Protocol reference number: HSS/1058/017D**

**Project title:** An exploration of the teaching and learning of Information Technology (IT) Programming in two higher education institutions in KwaZulu-Natal (KZN)

### Approval Notification – Expedited Application

In response to your application received on 11 July 2017, the Humanities & Social Sciences Research Ethics Committee has considered the abovementioned application and the protocol has been granted **FULL APPROVAL**.

Any alteration/s to the approved research protocol i.e. Questionnaire/Interview Schedule, Informed Consent Form, Title of the Project, Location of the Study, Research Approach and Methods must be reviewed and approved through the amendment/modification prior to its implementation. In case you have further queries, please quote the above reference number.

**PLEASE NOTE:** Research data should be securely stored in the discipline/department for a period of 5 years.

The ethical clearance certificate is only valid for a period of 3 years from the date of issue. Thereafter Recertification must be applied for on an annual basis.

I take this opportunity of wishing you everything of the best with your study.

Yours faithfully

Dr Shamila Naidoo (Deputy Chair)

/ms

Cc Supervisor: Professor V Mudaly  
Cc Academic Leader Research: Dr SB Khoza  
Cc School Administrator: Ms Tyzer Khumalo

---

Humanities & Social Sciences Research Ethics Committee

Dr Shenuka Singh (Chair)

Westville Campus, Govan Mbeki Building

Postal Address: Private Bag X54001, Durban 4000

Telephone: +27 (0) 31 260 3587/6350/4557 Facsimile: +27 (0) 31 260 4609 Email: [ximbap@ukzn.ac.za](mailto:ximbap@ukzn.ac.za) / [snymanm@ukzn.ac.za](mailto:snymanm@ukzn.ac.za) / [mohunp@ukzn.ac.za](mailto:mohunp@ukzn.ac.za)

Website: [www.ukzn.ac.za](http://www.ukzn.ac.za)



Founding Campuses: Edgewood Howard College Medical School Pietermaritzburg Westville

## **Dedication**

To the Supreme Lord for His guidance and blessing

To my family  
for their love and support

My IT students  
my inspiration to improve my IT Teaching practice

&

My Supervisor  
Prof Vimolan Mudaly

## **Acknowledgements**

I am deeply indebted to the following people who inspired and helped me throughout this research:

- This thesis could not have been written without the teaching experiences and ideas of the hundreds of students in the IT programming courses which I have taught at the Durban University of Technology and secondary schools for over 20 years.
- The participants, for their valuable contribution to this study.
- Prof Vimolan Mudaly, my supervisor, for his guidance and support throughout this study.
- I wish to thank the University of KwaZulu-Natal (Faculty of Education-Edgewood campus).
- I am extremely grateful to my statistician, Dr Sachin Suknunan, for all his assistance.
- My editors, Ms Sury Bisetty, Mr Trevor French and Dr Nereshnee Govender who has been most supportive of my work and has provided useful insights and encouraged me throughout the process.
- I also wish to thank the DUT Research Department for funding which enabled me to do the necessary fieldwork for this study and to fund my conference presentations.
- My wife, Vaneshree Govender – thank you for your love, patience and understanding.
- Klareesa, and Shivar Govender, for being my inspiration.
- My sister, mother and late dad, for their unconditional love and support.
- Finally, I thank God for granting me the strength, courage and perseverance to complete this thesis.

## Turnitin Report

An exploration of the teaching and learning of Information Technology (IT) programming in two higher education institutions in KwaZulu-Natal (KZN)

### ORIGINALITY REPORT

6%

SIMILARITY INDEX

%

INTERNET SOURCES

6%

PUBLICATIONS

%

STUDENT PAPERS

### PRIMARY SOURCES

1

"Perceptions of Learners of a Learning Management System to Support Teaching and Learning Using the Diffusion of Innovation Theory", Mediterranean Journal of Social Sciences, 2014.

Publication

1%

2

David F Birks, Walter Fernandez, Natalia Levina, Syed Nasirin. "Grounded theory method in information systems research: its nature, diversity and opportunities", European Journal of Information Systems, 2012

Publication

<1%

3

Péter Szlávi, László Zsakó. "Methods of teaching programming", Teaching Mathematics and Computer Science, 2003

Publication

<1%

4

Sayed Hadi Sadeghi. "E-Learning Practice in Higher Education: A Mixed-Method Comparative Analysis", Springer Nature, 2018

<1%

## **Abstract**

In this study, classic grounded theory, threshold concepts, self-study and practitioner research capture the processes of Information Technology (IT) academics who teach computer programming to first-year IT students at a university of technology in Kwa-Zulu Natal. The qualitative data analysis revealed the basic pedagogy of teaching and learning computer programming, and described how IT academics perceived their vocation and their decisions to take action to ultimately improve the quality of teaching and learning of IT programming. From the data, the following four themes emerged in the process of teaching and learning computer programming: 1) Teaching IT; 2) Learning IT and its impact; 3) Challenges in teaching IT; 4) Recommendations for teaching IT programming. This study will assist first-year IT programming academics to understand their pedagogical impact at an institution of higher learning. This study will further potentially serve as a path for future research and aid in understanding the pedagogical impact of the teaching and learning of IT on first-year IT students.

**KEYWORDS:** Teaching and Learning, Information Technology, Programming, Pedagogy, Practitioner research



## **Glossary of Terms**

In the context of this study, I attached broad meanings to the core concepts as follows:

<b>Educator</b>	Includes a teacher at a high school and a lecturer and/or academic at a tertiary institution.
<b>Grounded Theory</b>	Using guidelines for the organisation of data, theory is developed that provides relevant interpretations, applications, predictions and explanations.
<b>Higher Education</b>	A post-secondary school education that occurs at a college or university.
<b>Information technology</b>	Includes programming/coding, networking, human-computer interaction, databases and web systems.
<b>Learners</b>	Includes learners from high school and first-year tertiary students.
<b>Learning Environment</b>	Includes both a high school class room and a university lecture theatre.
<b>Object</b>	Contains both data and operations in the same entity.
<b>Object-oriented programming</b>	A computer programming paradigm based on the object-oriented approach, whereby objects have the responsibility of carrying out specific operations to solve a problem.
<b>Peer-to-peer</b>	Learners studying programming in the same classroom/grade/programming course.
<b>Programming</b>	The practice of writing computer programs in a specific computer programming language

## List of Abbreviations

<b>API</b>	Application Program Interface
<b>AppDev</b>	Applications Development
<b>BRICS</b>	Brazil, Russia, India, China and South Africa
<b>CBL</b>	Competency Based Learning
<b>CS</b>	Computer Science
<b>DHET</b>	Department of Higher Education and Training
<b>FET</b>	Further Education and Training
<b>GTM</b>	Grounded Theory Methods
<b>IDE</b>	Integrated Development Environment
<b>IS</b>	Information Systems
<b>IT</b>	Information Technology
<b>ICT</b>	Information and Communication Technology
<b>IoT</b>	Internet of Things
<b>IS</b>	Information Systems
<b>LCT</b>	Learner Centred Teaching
<b>MCU</b>	Microcontroller Unit

<b>MIT</b>	Massachusetts Institute of Technology
<b>MOOCS</b>	Massive Open Online Course
<b>MSDN</b>	Microsoft Developer Network
<b>MVC</b>	Model-View-Controller
<b>ND</b>	National Diploma
<b>NSFAS</b>	National Student Financial Aid Scheme
<b>OO</b>	Object Oriented
<b>OOP</b>	Object-Oriented Programming (concepts of classes, objects, events, inheritance, encapsulation)
<b>PBL</b>	Problem Based Learning
<b>PHP</b>	Hypertext Preprocessor
<b>POP</b>	Procedural Oriented Programming includes PPA
<b>PP</b>	Pair Programming
<b>PPA</b>	Procedural Programming Approach (Concepts of sequence, selection and iteration)
<b>SCL</b>	Student Centred Learning
<b>SoTL</b>	Scholarship of Teaching and Learning
<b>SQL</b>	Structured Query Language

<b>TCT</b>	Teacher-Centred Teaching
<b>TCT</b>	Teacher-Centred Teaching
<b>TLA</b>	Teaching Learning and Assessment
<b>ToT</b>	Technology of Turings
<b>UoT</b>	University of Technology

## Table of Contents

Preface .....	i
Declaration .....	ii
Ethical Clearance – UKZN .....	iii
Dedication .....	iv
Acknowledgements.....	v
Turnitin Report .....	vi
Abstract.....	vii
Glossary of Terms .....	viii
List of Abbreviations .....	ix
<b>CHAPTER ONE: INTRODUCTION .....</b>	<b>1</b>
<b>1.1 Introduction – Setting the scene .....</b>	<b>1</b>
<b>1.2 Rationale for the study .....</b>	<b>3</b>
1.2.1 Personal imperative .....	3
1.2.2 Research imperative .....	3
<b>1.3 Problem statement.....</b>	<b>4</b>
<b>1.4 Bounding the study .....</b>	<b>5</b>
1.4.1 The institution .....	5
1.4.2 The students.....	6
1.4.3 The department of ICT .....	7
1.4.4 The IT syllabus .....	8
1.4.5 The researcher’s role.....	11
<b>1.5 Aim of the study.....</b>	<b>12</b>
<b>1.6 Theoretical and conceptual framework.....</b>	<b>12</b>
<b>1.7 Key research questions.....</b>	<b>13</b>
<b>1.8 Structure of the study.....</b>	<b>13</b>
<b>1.9 Conclusion .....</b>	<b>14</b>
<b>CHAPTER TWO: LITERATURE REVIEW.....</b>	<b>15</b>

<b>2.1. Introduction .....</b>	<b>15</b>
<b>2.2 Chapter overview .....</b>	<b>16</b>
2.2.1 Benefits of teaching and learning Programming.....	18
2.2.2 Coding versus Computer Programming, coding literacy, Computer science and Information Systems .....	21
2.2.3 Defining Computer Science, IT, Theory of Computer Programming and Information Systems (IS) ..	23
2.2.3.1 Computer Science .....	23
2.2.3.2 Information Technology (IT).....	23
2.2.3.3 Theory of computer programming /Information System.....	25
2.2.4 Teaching and Learning Theories.....	26
2.2.4.1 Instructivism .....	27
2.2.4.2 Constructionism .....	27
2.2.4.3 Social Constructivism.....	29
2.2.4.4 Project based learning (PBL).....	29
2.2.4.5 Scaffolding and the gradual release of responsibility model .....	30
2.2.4.6 The Gradual Release of Responsibility model .....	30
2.2.4.7 Discovery learning .....	32
2.2.4.8 Learner-centered teaching (LCT) versus teacher-centered teaching (TCT).....	32
2.2.4.9 Meaningful Learning.....	33
2.2.4.10 Mastery learning .....	34
2.2.4.11 Situated learning .....	35
2.2.4.12 Competency based learning .....	36
2.2.4.13 Problem-based learning .....	37
2.2.5 Theories and methodologies of teaching and learning IT programming (coding) .....	37
2.2.5.1 Language oriented and semiotic ladder .....	38
2.2.5.2 Instruction-oriented .....	39
2.2.5.3 Problem type oriented/Mathematics oriented method .....	40
2.2.5.4 Based on a model .....	41
2.2.6 Approaches adopted by IT academics at the ToT when teaching programming .....	41
2.2.6.1 Portions of Code.....	42
2.2.6.2 Deep learning .....	44
2.2.6.3 Procedures and Functions.....	46
2.2.6.4 Microsoft MakeCode: inclusion of a physical computing device .....	47
2.2.6.5 Scratch programming .....	49
<b>2.3 Conclusion .....</b>	<b>51</b>
<b>Chapter Three: Theoretical, Conceptual and Philosophical framework.....</b>	<b>53</b>

<b>3.1 Introduction .....</b>	<b>53</b>
<b>3.2 Chapter overview .....</b>	<b>54</b>
3.2.1 Constructing knowledge within an interpretive paradigm .....	54
3.2.2 Frames of enquiry: Qualitative research.....	55
3.2.3 Self-study and practitioner research.....	56
3.2.4 Personal reflexivity and prior experience .....	57
3.2.5 Threshold concepts .....	59
3.2.5.1 Nine considerations for threshold concepts .....	61
3.2.6 Grounded theory.....	68
3.2.6.1 Application of grounded theory methods (GTM) to this study.....	70
<b>3.3 Conclusion .....</b>	<b>74</b>
<b><i>Chapter Four: Research methodology .....</i></b>	<b><i>75</i></b>
<b>4.1 Introduction .....</b>	<b>75</b>
<b>4.2 Chapter overview .....</b>	<b>76</b>
4.2.1 Research context and sampling .....	76
4.2.2 Research questions and design.....	78
4.2.3 Timeframes for data collection .....	80
4.2.4 Data collection methods .....	80
4.2.4.1 Observation.....	81
4.2.4.2 Interviewing .....	82
4.2.5 Design of the interview and observation .....	83
4.2.6 Interview questions/observation criteria.....	83
4.2.7 Data analysis and value of the question set.....	83
4.2.8 Validity/reliability/trustworthiness.....	85
<b>4.3 Ethical issues of the study.....</b>	<b>87</b>
<b>4.4 Limitations of the study .....</b>	<b>89</b>
<b>4.5 Conclusion .....</b>	<b>90</b>
<b><i>Chapter Five: Qualitative analysis and discussion.....</i></b>	<b><i>91</i></b>
<b>5.1 Introduction .....</b>	<b>91</b>
<b>5.2 The Participants .....</b>	<b>91</b>
<b>5.3 Biographical data .....</b>	<b>91</b>
5.3.1 Participants qualifications.....	91
5.3.2 Length of time teaching programming .....	92

<b>5.4 Thematic Analysis of Qualitative Data – Identification of Themes .....</b>	<b>93</b>
5.4.1 Word cloud.....	94
5.4.2 Treemap .....	95
5.4.3 Cluster analysis.....	97
<b>5.5 Formulation of themes .....</b>	<b>98</b>
5.5.1 Teaching .....	98
5.5.1.1 Concepts and topics .....	98
5.5.1.2 Assessments .....	125
5.5.1.3 Pair programming.....	133
5.5.1.4 Support and Engagement.....	135
5.5.1.5 Teaching support.....	140
5.5.2 Learning and Impact.....	143
5.5.2.1 Background and knowledge .....	143
5.5.2.2 Learning methods.....	151
5.5.2.3 Group learning.....	156
5.5.2.4 Performance .....	162
5.5.3 Challenges .....	168
5.5.3.1 Background and diversity .....	168
5.5.3.2 Student ability and readiness .....	169
5.6.3.3 Lecture and classroom .....	171
5.6.3.4 Overcoming challenges .....	173
5.5.4 Recommendations to lecturers teaching programming .....	176
5.5.4.1 Teaching .....	177
5.5.4.2 Students .....	180
5.5.4.3 Resources and support.....	183
<b>5.6 Conclusion .....</b>	<b>184</b>
<b>Chapter Six: Summary, Recommendations and Conclusion .....</b>	<b>185</b>
<b>6.1 Introduction .....</b>	<b>185</b>
<b>6.2 Research question one and research question four .....</b>	<b>186</b>
6.2.1 Lecturer style.....	187
6.2.2 Exhibitor .....	188
6.2.3 Helper .....	188
6.2.4 The Deputy .....	188
6.2.5 Mixed Methods Strategy.....	189
<b>6.3 Research Question Two and Research Question Three .....</b>	<b>190</b>



<b>6.4 Summary: Knowledge to Practice.....</b>	<b>195</b>
<b>6.5 Concluding remarks &amp; final implications to the findings from the study .....</b>	<b>197</b>
<b>6.6 Concluding remarks &amp; a key message of the study.....</b>	<b>199</b>
<b>References.....</b>	<b>201</b>
<b>Appendix A: Interview Schedule for the academics .....</b>	<b>211</b>
<b>Appendix B: Observation Schedule .....</b>	<b>215</b>
<b>Appendix C: Interview Schedule with Academic participant 3.....</b>	<b>217</b>
<b>Appendix D: Request for Gatekeepers Permission.....</b>	<b>247</b>
<b>Appendix E: Letter of Permission .....</b>	<b>248</b>
<b>Appendix F: Ethical Clearance – MUT .....</b>	<b>249</b>
<b>Appendix G: Informed Consent to Student.....</b>	<b>250</b>
<b>Appendix H: Informed Consent to Academic .....</b>	<b>254</b>
<b>Appendix I: CS50 online course @Harvard extension school .....</b>	<b>258</b>
<b>Appendix J: Language Editing Certificate.....</b>	<b>260</b>

## List of Tables and Figures

Table 1-1: Entrance requirements IT qualification at ToT (DUT, 2018).....	7
Table 1-2: Entrance requirement four-year diploma (DUT, 2018).....	8
Table 2-3 Review of Instructivist Learning Theory (adapted from Sadeghi, 2015).....	27
Table 2-4 Review of Constructionism Learning Theory (adapted from Sadeghi,2015) .....	28
Table 5-1 Biographical data of sample population .....	78
Table 6-1 Treemap Data derived from the data analysis .....	96
Figure 2-1 The Information Technology Discipline.....	24
Figure 2-2 The transition to student independence (Pearson & Gallagher, 1983).....	31
Figure 2-3 Compiled in MatLab (MathWorks, 2019a).....	42
Figure 2-4 C++ code .....	43
Figure 2-5 PHP code highlighting the IF-elseIF statements (w3schools.com, 2019) .....	44
Figure 2-6 Classify Webcam Images Using Deep Learning .....	46
Figure 2-7 Create Procedure (w3schools.com, 2019).....	47
Figure 2-8 Programme in Microsoft MakeCode.....	48
Figure 2-9 Prinavin’s Scratch Project 15 Feb (Govender, 2018).....	50
Figure 3-1 A threshold concepts view of learning (Goebel & Maistry,2019).....	59
Figure 4-1 Excel spreadsheet denoting nodes from the data set.....	85
Figure 5-1 Length of teaching time .....	92
Figure 5-2 Word cloud generated from qualitative data .....	94
Figure 5-3 Treemap generated from qualitative data.....	95
Figure 5-4 Cluster analysis generated from qualitative data .....	97
Figure 5-5 Most popular and most viewed video on my YouTube video channel.....	101

## CHAPTER ONE: INTRODUCTION

### 1.1 Introduction – Setting the scene

“Learning to write programs stretches your mind, and helps you think better, creates a way of thinking about things that I think is helpful in all domains.”

– Bill Gates Co-Chairman, and Co-Founder, Microsoft (Code.org,2019b)

In 1964 the programming language BASIC (Beginners All-purpose Symbolic Instruction Code) was developed at Dartmouth College by John Kemeny and Thomas Kurtz (Scaruffi, 2016). Computer scientists and programmers were introduced to print “Hello World!” and whilst there have been mammoth advancements in computing processing power and a significant reduction in the size and complexity of computer hardware, one of the consistent greetings and introductory programming lessons, even after over fifty-five years, still refers to “Hello World!” From the humble beginnings of complicated machine code of binary digits and assembly language coding, twenty-first century millennial programming students who design and develop applications (Apps) now revel in the appreciation of simplified high-level English-like source code that would be the envy of programming pioneers of the early 1960s.

Can this high-level code be implemented for all learners and not just millennial programming students? The former President of the United States of America, Barak Obama, thought so. He called it “Computer Science for all Initiative”. In 2016 President Obama wanted to ensure that every K-12 student had access to the computer science curriculum (Dickey, 2016). South African President, Cyril Ramaphosa, followed suit in February 2019 when he stated in his State of the Nation Address 2019 (SONA 2019):

In line with our Framework for Skills for a Changing World, we are expanding the training of both educators and learners to respond to emerging technologies including the Internet of Things, robotics and artificial intelligence (Ramaphosa 2019).

President Ramaphosa’s announcement sparked a subsequent announcement by the Minister of Basic Education, Angie Motshekga on the 8 March 2019 regarding the implementation of a coding (programming) curriculum at schools. Motshekga described the initiative as:

... an ambitious project that will use the University of South Africa's twenty-four Information and Communication Technology (ICT) laboratories throughout the country for the training of 72 000 teachers in coding. This implementation is going to commence in 2020 in 1 000 schools in five provinces for Grade 7 to Grade 9 (Motshekga, 2019).

Since the dawn of South African democracy, this is the first time that the computer science subject is gaining such prominence as an important subject at school level. As seen by the President's and the Minister of Basic Education, Angie Motshekga's subsequent announcement, much of the rhetoric concerning computer science has focused on broadening the outreach of computer science to school learners. Broadening school participation is an important strategy given the poor levels of participation of school pupils in computer science subjects, which will ultimately influence the tertiary institutions' enrolment of students who want to pursue studies in Information Technology (IT) programming-related subjects. Most importantly, stemming from the recent announcements, computer science and the related IT subjects are finally being awarded importance at a national level. The announcements also support this study as the findings from academics and students studying introductory programming courses at higher education institutions can be used as a catalyst to improve the teaching strategies of IT teachers at both primary and secondary school levels. ToT computer programming lecturers and students were interviewed on their programming curriculum, programming language taught, assessment techniques, availability of resources and a host of other challenges. It is important to note that apart from the active and participatory role adopted, to enhance the depth of this study, I also enrolled as a computer programming student via distance/correspondence learning, hence, mixing student, academic and researcher roles. This process allowed me to analyse my efficacy as a computer programming lecturer and evaluate my teaching and learning practice.

Hence, I considered both my teaching and learning strategies and those of fellow academics and students with a view to enhancing my practice as an IT programming academic as well as the effectiveness of such strategies on firstly, my own IT programming students and that of Durban University of Technology (DUT) students. In this chapter, I discuss the rationale for engaging in this study, present the context, outline the key research questions, discuss the significance, and provide an overview of the dissertation.

## **1.2 Rationale for the study**

The rationale for this study is driven by two imperatives:

1.2.1 Personal

1.2.2 Research related

### **1.2.1 Personal imperative**

My imperative arises from the fact that I have been a computer programming academic for the past twenty-five years and have observed poor pass rates and high dropout rates, especially at the first-year level. Furthermore, my journey in computer programming commenced with the BASIC programming language: 10 print "Hello World!" in 1986. The journey continued as a learner in secondary school, as a student of computer science, a secondary school academic in computer studies and now a computer programming lecturer at DUT. Personally, there is no greater feeling of excitement, enjoyment and accomplishment than to see the fruition of a successful compilation of a computer program, having struggled with it for days, weeks and even months; and the passion and desire to program seemingly illegible text on a computer either on a stand-alone machine or cloud-based and to produce an output, still both fascinates and excites me.

### **1.2.2 Research imperative**

The research imperative for this study is that students perceived programming to be difficult, coupled with the increased dropout rate amongst programming students and decreased throughput rate of programming qualifications (Mendes, Paquete, Cardoso & Gomes, 2012; Bati, Gelderblom, & van Biljon, 2014; Watson & Li, 2014; Carter & Dewan, 2018; Yeomans, Zschaler & Coate, 2019). Students do not just consider programming difficult; they also perceive programming as highly demanding and stressful (Javidi, 2014). At the ToT, the initial dropout rate was between 50 to 60 percent of the intake at the first year (van Zyl, 2015). The first-year IT curriculum includes a course in programming using the programming languages Java, Microsoft C# and Python. Earlier programming courses were consistent with command-driven and procedural programming languages. The inclusion of object-oriented programming languages introduced greater flexibility in teaching and learning; however, the pass rate in introductory programming remains low. The introductory programming courses succumb to the above-mentioned dropout rates. In 2019, supported by anecdotal evidence, the failure rate was between 50 to 60 percent.

Furthermore, this study noted that introductory IT students considered passing a theoretical programming course more difficult than a practical programming course. This study then further explored the experiences of academics and students engaged in computer programming at ToTs in Kwa-Zulu Natal to determine the factors contributing to the high failure rate and students' difficulty in passing practical programming courses, as well as factors contributing to student success in IT programming. The outcomes and recommendations of the research will offer relevant stakeholders in the realm of IT academia strategies to adopt in their secondary school classrooms and tertiary lecture and practical lecture theatres.

The association between my personal and research imperatives is that there is a definite need for an intervention strategy by the ToT's IT department to improve students' understanding of computer programming concepts and ultimately to improve the throughput rate of students. In the next sub-section, I provide detailed information on the area to be critically examined in this study, the objectives of the study, the theoretical and conceptual frameworks, and the research questions which presided over the study.

### **1.3 Problem statement**

At the ToT, Microsoft C# and Java were adopted as the programming languages in introductory programming courses. Computer problems were solved with a programming language and within a programming paradigm. The lecturers assign problems to students in a programming language course, and students then independently solve or attempt to solve the problem and code the solution in the programming language.

Academics at the tertiary institution lectured programming subjects according to their experiences as students, or they implement "quick fix" teaching strategies acquired at institutional workshops and conferences. Their role models were their former lecturers or workshop and conference presenters; hence, the pedagogy of teaching programming was a continuous cycle of repetition from former lecturers or workshop and conference presenters. This disjuncture between how academics were taught or lectured to and how they are currently teaching is problematic and requires further investigation.

## **1.4 Bounding the study**

This study is bounded by the institution, the students, the staff, the syllabus and the researcher.

### **1.4.1 The institution**

This study would be conducted on the IT campuses of ToTs in KwaZulu Natal. The ToTs are situated in urban areas and emerged from the political transformation in South Africa, wherein Technikons were merged and transformed into Universities of Technology. This research study would be confined to the Information Technology departments within the ToTs, and due to the sensitivity and confidentiality of the data from this point onwards, the ToTs would be referred to by pseudonyms. The ToTs would be collectively referred to as Technology of Turings (ToTs) in honour of Alan Turing, who is considered the founder of computer science (Dawson, 1985).

In order to bound the study within the research setting, I have provided descriptions of the student population, the institution and the IT department. This is crucial because the themes and categories that emerged during data analysis had a direct bearing on the student and academic population in the study. The descriptions will also provide an enhanced understanding of the context of this research. The department of ICT, where this study was conducted is situated within the Faculty of ToTs. At the time of this study, student enrolment at the TOT was in excess of 40 000 students. Most of the qualifications are at the Diploma level with a few departments offering postgraduate qualifications and award Diploma, Bachelor, Master's and PhD qualifications.

ToTs were once considered part of institutions of higher learning from the former disadvantaged sector of the South African population. However, since 1994 with the advent of democracy in SA; the South African government has engaged in improvements in all former disadvantaged areas, and facilities such as roads were developed (Reitzes, 2009). These former disadvantaged areas, however, have retained much of their demographic composition and are still reeling from the vestiges of underdevelopment from the apartheid days, particularly in education and schooling (Mavunga, 2014). This vestige of underdevelopment is evident in the student population which originates from secondary schools, especially rural schools that still experience traces of apartheid. ToTs attract students from the aforementioned secondary

schools. The aim of this study was not to highlight and discuss the background of the students, but rather to create awareness of their socio-economic background, which influences their success as IT students. During the study, it became evident that many students were dependent on study grants to fund their studies. Problems with the allocation of funding through the National Student Financial Aid Scheme (NSFAS) provided fertile ground for student protests, and disruptions were prevalent during data collection phases of this study. This was one of the contributing factors which hindered students' learning and it had a ripple effect on the rest of the student population. Another factor that was seen as a hindrance to student success was the housing and accommodation situation of many students. This aggravated the problems with the NSFAS and had a negative effect on student attendance at lectures. High absenteeism at lectures was attributed to students' poor living conditions, inclusive of unreliable travel arrangements. These socio-economic conditions contribute to the way students learn and to their success at university.

#### **1.4.2 The students**

Even though the number of first-year students who applied to ToTs was almost quadruple the number that the departments could admit; the selection committee at ToTs did indicate that the prospective students completed their secondary school in rural township schools. Jantjies and Joy (2015) note that in rural areas there are inadequate resources, limited books and teacher shortages. The students who originated from the surrounding urban areas came from secondary schools that did not offer computer studies or IT in the Further Education and Training (FET) phase. This does not auger well for first-year IT students from rural or township secondary schools. A pertinent observational measure of the financial well-being of students was noted in the number of students who had personal laptops. Most of the first-year students did not have personal laptops. The students who were in possession of a laptop had basic specifications and were generally allocated to be the leader of programming project groups. While this may not necessarily be a clear indication of the financial wellbeing of students, it is the lack of resources that hampers student success. A laptop is just as essential as a prescribed textbook or study guide when registering in a programming course at a tertiary institution.

Another inhibiting factor that surfaced during the observation phase was student culture. First-year IT students are passive and quiet. They arrive from secondary school having had twelve years of a teacher-pupil relationship, sitting quietly behind a classroom desk and



expecting a teacher to be the knowledgeable sage. They receive a culture shock when faced with sitting in a lecture room with three hundred students from diverse cultures and a lecturer who does not even learn their name. Whilst student culture is beyond the scope of this research, it is a contributing factor to student success, and it contextualises the student sitting in a first-year programming class who was taught English as a secondary language and now has to learn and study a brand-new programming language. An important observation was the level of computer illiteracy as contextualised by the frequency of requests on how to type the “@” symbol in an email address or the keystrokes to type ALL capital letters on the keyboard. This was disconcerting during orientation of first year students.

### 1.4.3 The department of ICT

The departments of ICT at ToTs are composed of lecturing, technical and administrative staff. At the time of the data collection phase, the departments offered a National Diploma (ND) in Information Technology (IT) and provided End User Computing support services to other faculties within the institutions. For students to gain entry into the ICT department and study for the three-year Diploma in Applications Development or a three-year Diploma in ICT in Business Analysis, they were required to meet the following requirements:

The minimum entrance requirement is a National Senior Certificate (NSC) or Senior Certificate (SC) or a National Certificate Vocational (NCV) for entry into a Diploma and students must meet the following requirements:

Compulsory Subjects	NSC Rating	Senior Certificate		NCV
		HG	SG	
English (Home Language) OR English (1 <sup>st</sup> Additional Language)	3 (40-49%) 4 (50-59%)	E n/a	C n/a	50% n/a
Mathematics OR Mathematical Literacy	3 (40-49%) 6 (70- 79%)	E n/a	C n/a	50% n/a
Two 20 credit subjects (Life Orientation or more than one additional language is excluded)	3 (40-49%)	n/a	n/a	(a) At least 50% in one fundamental subject, in addition to English & Mathematics. (b) At least 60% in three compulsory vocational subjects

Table 1-1: Entrance requirements IT qualification at ToT (DUT, 2018)

**Note:** This requirement represents the minimum requirements and students would be ranked according to a points system based on the rating code. For students to gain entry into the department and study the four-year Diploma in ICT in Applications Development, they were required to meet the following requirements.

Compulsory Subjects	NSC Rating	Senior Certificate		NCV
		HG	SG	
English (Home Language) OR English (1 <sup>st</sup> Additional Language)	3 (40-49%) 3 (40-49%)	E n/a	C n/a	50% n/a
Mathematics OR Mathematical Literacy	3 (40-49%) 5 (60- 69%)	E n/a	C n/a	50% n/a
Two 20 credit subjects (Life Orientation or more than one additional language is excluded)	3 (40-49%)	n/a	n/a	(a) At least 50% in one fundamental subject, in addition to English & Mathematics. (b) At least 60% in three compulsory vocational subjects

Table 1-2: Entrance requirement four-year diploma (DUT, 2018)

Table 1-2 represents the minimum requirements and students would be ranked according to a points system based on the rating code.

#### 1.4.4 The IT syllabus

The first year of study requires all students to register for the following compulsory subjects:

##### Semester 1

- Cornerstone101
- ICT Literacy and Skills
- Business Fundamentals 1
- Applications Development 1A
- Fundamentals of Computer Security
- Information Systems 1

## Semester 2

- Me, My World, My Universe
- Operating Systems
- Applications Development Project 1
- Applications Development 1B
- Communications Networks 1

In previous years it was found that students who failed a prerequisite subject in the first semester were disadvantaged from progressing into the second semester. Hence, in order to improve throughput rates, students who had failed Applications Development (AppDev) 1A were granted an exposure status to AppDev 1B in the second semester. Applications Development 1B does not have a prerequisite. There were instances of a few students who repeated the subject.

The computer programming subject, AppDev 1, at first year level was the focus of this study; hence, the subject content and abridged syllabi are included below.

### Applications Development 1A (APDA101)

- Introduction .Net Platform
- Introducing the C# Programming Language
- Getting started with .Net developing using C#
- Language Essentials
- Expressions and Operators
- Primer on Types and Objects
- Simple Flow Control
- Basics of Exception and Resource Management
- Introduction Types Methods
- Introduction to Unit Testing

### Applications Development 1B (APDB101)

- Fields, Properties and Indexers
- Constructors and Finalizes

- Operators, Overloading and Conversions
- Object Oriented Programming
- Generic Types and Methods
- Collection
- Types Delegates
- Events
- Language Integrated Query Essentials
- Exceptions
- Working with IO

Another subject closely related to App Dev 1A is Applications Development Projects.

Applications Development Project 1 (APDP101) has the following outcomes:

- Fundamental knowledge of how to design, develop and implement an application;
- Ability to test the application in a live environment;
- Ability to incorporate limited processing capabilities into the application;
- Create and submit documentation for the web application in the form of a report;
- Ability to apply logic and problem-solving skills;
- Abilities to synthesise knowledge from other learning areas into the capstone project;
- Demonstrate and present the application.

Unfortunately, it was noted during the data collection phases that students did not see the link between the AppDev modules and were unable to apply the knowledge acquired in AppDev 1A and AppDev 1B. As a result, concepts taught in AppDev 1A were repeated whilst lecturing AppDev 1B. It is evident that students did not acquire the threshold concepts in AppDev1A. Threshold concepts are discussed in greater detail in Chapter Three.

In contextualising and setting the scene for this study, it is imperative that I define my role as a lecturer. Hence, in this final section, I discuss my role as an academic, a researcher, a student and a lecturer.

#### **1.4.5 The researcher's role**

My personal experiences as a student, a secondary school teacher and a lecturer shaped my perceptions of IT programming. I played an active role in this qualitative study, particularly in the data collection and analysis of this research; hence, I would like to identify my personal biases and values. Wood, Mento & Locke (1987) state that the investigator's research contribution can be useful and positive rather than detrimental. I consider my contribution to be useful and positive.

From 1986 to 1988 I was a pupil of computer studies at a secondary school. Post-secondary school, I studied for a diploma in higher education, further specialising in mathematics and computer science, and then taught computer studies for another five years. Whilst I was teaching, I continued my studies in undergraduate and postgraduate computer science studies which facilitated my transition into higher education. I believe this understanding of my role as a pupil, a student, a teacher and eventually a lecturer in IT programming, enhanced my awareness, knowledge and sensitivity to many of the challenges, decisions and issues encountered by first year students and academics at higher institutions.

I bring knowledge of both the structure of secondary school education and the role of a student and a lecturer in higher education. I inadvertently bring certain biases to this study due to my role as a lecturer, academic and researcher. Although every effort was made to ensure objectivity, these biases may have shaped the way I viewed and understood the data I collected and the way I interpreted my experiences. I commenced this study with the perspective that teaching is one of the most respected and noble professions. It is also the most undervalued and most gratifying professions. Though successful student outcomes and increased throughput rates are expected, I question the gravity of the first-year lecturer and first-year student at ToTs. I view the first-year student experience as critical; filled with adjustments, frustrations, unanticipated surprises and challenges. Furthermore, successful student experiences during the first semester may contribute to increased graduation throughput rates, and well-adjusted, satisfied tertiary students could graduate in minimum time and become model South African citizens.

The previous sections contextualised the study by describing the students, the institution, the syllabus, staff and my role. All of the above impinged upon student success in IT programming and were my motivation for exploring the teaching and learning of IT programming at ToTs.

### **1.5 Aim of the study**

The aim of the study was to examine the teaching and learning of an introductory programming course at ToT and to provide academics at tertiary institutions with a strategy they could adopt when teaching and learning introductory programming courses.

The aim of this study is separated into the following goals:

1. To determine the type of teaching strategies used in IT classes at a ToT in KZN;
2. To examine the teaching and learning of IT at a ToT in KZN; and
3. To determine how these teaching strategies used by ToT academics could influence the learning of IT programming.

### **1.6 Theoretical and conceptual framework**

The theoretical and conceptual framework for this study comprised threshold concepts in computer science, and grounded theory, self-study and practitioner research. Threshold concepts might be used to organise the educational process, and are likely to be transformative, integrative, irreversible, and potentially troublesome for students, and are often boundary markers (Meyer & Land, 2005). These attributes are discussed in detail in Chapter Three. Grounded theory was used to develop the theory that provided relevant interpretations, applications, predictions and explanations (Glaser & Strauss, 1967). I draw distinctions in this study between being an academic practitioner, a researcher, and my student and learning practice. Data were analysed and discussed according to these criteria in Chapter Three. The interpretive paradigm was selected for this study as it is concerned with descriptions that produce deep understanding and emphasises the interpretation of data from lecturers and students. It is within the boundaries of these theoretical frameworks that the research questions were answered.

## **1.7 Key research questions**

In this study, I was interested in one central phenomenon, namely, the teaching strategies of IT academics at a ToT and to achieve this, specific research questions were formulated. The research questions narrowed the aim of the study, and became a major focal point for this study.

- 1) What are the teaching strategies used by ToT IT academics when teaching IT programming?
- 2) What are the perceptions of students regarding the teaching methodologies used for ToT IT programming?
- 3) What are the perceptions of ToT academics regarding factors impeding student IT programming?
- 4) How do the teaching strategies employed by IT academics at a ToT influence the learning of programming?

The above questions addressed a description of the teaching and learning of IT programming at a ToT, and from the above questions, the following main themes emerged in the process of teaching and learning computer programming: 1) Teaching IT; 2) Learning IT and its impact; 3) Challenges in teaching IT; and 4) Recommendations for teaching IT programming.

## **1.8 Structure of the study**

This thesis is made up of six chapters. This section will provide a summary of each chapter.

Chapter One introduces the study and presents the research topic, research questions, problem statement and rationale for the study.

Chapter Two provides a literature review of the research topic, including the difference between IT programming, the theory of programming and information systems. It further presents the results of previously conducted studies on teaching programming in academia.

Chapter Three discusses the theoretical and conceptual framework for the investigation, including threshold concepts, grounded theory and self-study and practitioner research.

Chapter Four sets out the research methods used to explore the research questions and offers an overview of the research design; it also details the interpretative paradigm, data collection and data analysis methods.

Chapter Five presents an analysis of the results of the research questions. Tables and graphs are used to show the findings from the questionnaire, and thematic analysis is applied to data from the interviews. This is followed by a discussion of the research findings according to the principles of grounded theory and threshold concepts, and with reference to the literature review.

Chapter Six provides an argument for the findings which relate to the research questions. Recommendations are provided for further study and limitations are discussed.

## **1.9 Conclusion**

This chapter explored the purpose for conducting this study and provided a background to the context of the study. The research questions were introduced, and the structure of the thesis was outlined. In the next chapter, the relevant literature that supports the theories and methodologies of teaching and learning computer programming is reviewed; computer programming terms are defined, and the benefits of teaching and learning IT programming are discussed.



## CHAPTER TWO: LITERATURE REVIEW

“I think everybody in this country should learn how to program a computer, because it teaches you how to think.”

– Steve Jobs on Computer Science – Chairman, chief executive officer, and co-founder of Apple Inc. (Jobs,2013)

### 2.1. Introduction

This chapter consolidates ideas emerging from developments of teaching computer programming in institutions of higher learning from a theoretical perspective. Drawing from a variety of studies conducted by numerous researchers, the purpose of this chapter is to establish the nature of teaching and learning IT programming and the variables that influence both academics and students. This chapter further seeks to unpack theoretical assumptions developed around teaching programming from the perspective of researchers who have conducted in-depth studies which explore the workings of the teaching and learning phenomenon as practised in diverse contexts at institutions of higher learning. Literature reviewed in this chapter provided a view of the key arguments and findings of several studies that were used to generate assumptions necessary to inform the outcomes of this study. The literature reviewed applied to the themes and categories which emerged from the analysis of the data, as per the theoretical criteria of grounded theory. The literature informed the structure adopted by this study as it draws on how other researchers whose work was used to develop this literature conducted their studies from both the theoretical and methodological perspectives.

One such perspective was that of Boyer (1990), whose theoretical and methodological discussion of the Scholarship of Teaching and Learning (SoTL) gave rise to a worldwide discussion about academic engagement in teaching practice and increased emphasis on a scholarly approach to teaching. Ernest Boyer was an educator who first used the term scholarship of teaching in which he advocated that “The time has come to move beyond the tired old teaching versus research debate and give the familiar and honourable term scholarship a broader and more capacious meaning, one that brings legitimacy to the full scope of academic work” (Boyer, 1990, p.16).

The time has come again in 2019 and beyond, when we as IT academics must move beyond the tired old teaching versus researching, debating and critiquing the teaching and learning of introductory programming at ToTs. This view of SoTL is the underpinning premise upon which this research was based and is a further conduit through which the data were analysed and through which the teaching practice of first-year IT programming was reviewed and critiqued.

The researcher and the sample population are academics from institutions of higher learning, and they are actively engaged in teaching computer programming to introductory programming students. Boyer's approach required instructors to be aware of two distinct categories of research literature: theories of teaching; and knowledge acquisition relevant to the discipline. In this instance, the discipline is teaching computer science (IT) programming to introductory higher institution students and knowledge acquisition refers to a particular programming language construct and syntax. Whilst the participants may not have been acutely aware of particular theories of teaching, evidence based on their teaching styles indicated an alignment to a particular theory of teaching. Chapter Five and Chapter Six discuss the teaching styles of the participants in greater detail.

All the participants from the sample population are ToT academics, qualified in the computer science discipline and are schooled in the theories of teaching and knowledge acquisition. During the data analysis phase, it became apparent that key themes that surfaced were teaching; learning and its impact; and challenges and recommendations for teaching. The key themes would be discussed in greater detail in Chapter Five. The ensuing learning theories which emerged are in direct association to Boyer's theory of teaching and are discussed with reference to the data from participants. The learning theories were selected based on the coding and observational data from the participants. The theories are to be considered a thematic analysis of the literature review. With reference to grounded theory terminology the themes and theories are coded categories of teaching and learning IT programming which emerged from the data.

## **2.2 Chapter overview**

The thematic categories of the literature review in this chapter examine and emphasise:

- Benefits of teaching and learning IT programming (coding);
- Coding versus Computer Programming,

- coding literacy Computer science,
- Information Systems;
- Defining IT programming,
  - Theory of Programming and Information systems (IS)
  - Computer Science (CS) and
  - Coding;
- Theories and methodologies of Teaching and learning IT programming (coding) with specific reference to observational and interview data from participants.

The reference to the theories correlates to the major themes/categories of the research as discussed in Chapter Five and Chapter Six along with the framework for enhancing academics as university teachers Version: November 2018 (Training, 2018; DHET, 2018). The framework is underpinned by the following principles:

- Good teaching is a vital contributor to student learning and success.
  - The outcome of this study has contributed to the “good teaching contributions” for the IT academic teaching IT programming for first-year ToT students.
- Universities that place students at the centre of their work are characterised by a pedagogy of care and are underpinned by a strong social justice agenda.
  - This study highlights the importance of student-centeredness.
- The work of an academic involves being a teacher and a researcher.
  - IT academics at ToTs are both teachers and researchers. This is a self-study in IT teaching and learning of first-year ToT students.
- Good teaching is grounded in a deep understanding of a discipline.
  - This study reflects a deep understanding of the IT discipline at a ToT.
- Improving university teaching and learning must take into account the needs of academics at all levels, of teacher leaders and of professional support staff.
- Professional development activities for university teachers and the professionals who support teaching development need to be available across the career continuum.
- Whilst adequate resources are essential, the greatest barrier to good teaching is ideology.

- The teaching role is often more fully embodied in a team than in an individual.
- Teaching can be advanced when the discipline and the people involved identify and address their own teaching development needs.
- Professional development cannot be imposed but must be undertaken by the person concerned.
- A recognition and reward system can contribute positively to teaching development.
- Time needs to be allocated to professional development activities.
- Teaching development professionals need be able to develop their own capacity and careers.

The above principles are all discussed (in Chapter Five) to varying measures of extent under the themes and sub-themes of this study.

### **2.2.1 Benefits of teaching and learning Programming**

Feurzeig (1981) provides an extensive set of cognitive outcomes from learning to program. Teaching the set of concepts related to programming can be used to provide a natural foundation for the teaching of mathematics, the art of logical and rigorous thinking in general (Feurzeig, Papert & Lawler, 2011). Learning to program is expected to bring about seven fundamental changes in thought (Feurzeig, 1981). The seven changes are:

- 1) Rigorous thinking, precise expression, recognise the need to make assumptions explicit.
- 2) Understanding of general concepts such as formal procedures, variables, and function and transformation.
- 3) Greater facility when the art of “heuristics”, explicit approaches to problem-solving in any domain, such as problem solving by decomposing it into parts.
- 4) The general idea that debugging of errors is a constructive and plannable activity applicable to any kind of problem-solving since it is an integral part of getting programs to run as intended.
- 5) The general idea that one can invent many small procedures as building blocks for gradually constructing solutions to larger problems since programs are composed of procedures.

- 6) Generally enhanced self-consciousness and literacy about the process of solving problems. This is due to the practice of discussing the process of problem-solving programming by means of the language of programming concepts.
- 7) Enhanced recognition for domains beyond programming. Different ways have comparative costs and benefits associated with specific programming goals.

During the data analysis phase (observation and interview sessions) the participants indicated programming tasks had to be designed to generate precise output. Students had to make assumptions that were explicit, for instance it was explicitly indicated that the input to the program had to be numeric. This aligns to point 1.

In the data collection phase, the use of procedures, variables, functions and the transformation of input into an output according to a set of procedures and rules on mathematical expressions aligns to point 2 above. Whilst the scope of this study includes threshold concepts, it excludes categorising and assigning threshold concepts; however, procedures, variables and functions are defined and considered threshold concepts to first-year programming students (Kallia & Sentance, 2017). Chapter Three discusses threshold concepts with specific reference to the teaching of programming concepts observed in this study.

With respect to point 3; the fundamental change occurred in each and every practical lecture. The instructor or tutor would decompose a large problem into smaller sub-problems. Generally, the problem-solving task would answer the following questions:

- What is the input to the program?
- What mathematical expressions must be used to generate the input?
- What are the processing tasks involved in the program?
- What is the expected output and the format of the display?

Furthermore, error-checking and debugging were initiated as part of problem-solving. The instructor would deliberately input an error variable then execute the programme to generate programme output; for instance, if the input required a numeric value, the instructor would include alphanumeric input values. With reference to “debugging” (an integral part of programming), it was observed during the data collection phases that it was very rare that a

student's programming task would compile and execute on the first attempt. More often than not, a student had to constantly debug the program code prior to successful compilation.

Similarly, with regard to point 5, initially the instructor would write program code without procedures or functions. However, once procedures and functions were taught to students they realised that a number of small functions and procedures would be combined together to solve the larger problem. This study considers procedures and functions to align to the characteristics of threshold concepts.

Towards the middle of the semester, students displayed a sense of maturity in the self-consciousness of problem-solving. This was observed in specific and targeted problem-solving strategies, for instance, online searches of Internet bulletin boards, or using the concept of heuristics to determine an optimal solution to a problem or simply working with a fellow student.

With regard to point 7, students realised that a single problem could have numerous solutions depending on the assumptions and interpretation of the problem provided by the instructor. The numerous solutions have numerous costs and benefits associated and this is of importance not only to academia but to the software industry. Jørgensen (2014) finds that inaccurate estimation is the root factor of failure in most of the software projects that fail. Galorath and Evans (2006) find, amongst other reasons for software project failures, are insufficient requirements engineering, poor planning, sudden decisions and inaccurate estimations. ToT lecturers neglected to consider cost benefit factors of software solutions. However, teaching IT programming problem-solving skills will benefit academia and the students' future success in the software industry. Industry, and more especially the software industry is fraught with extra costs and project failures. Most projects (60-80 percent) encounter effort and/or schedule overruns (Molokken & Jorgensen, 2003). This adds to the final cost of the project. Cockburn and Williams (2000) note that managers may view programmers as a scarce and a costly resource.

Furthermore, with the advent of the Fourth Industrial revolution, it has been stated, "As a result of a massive development of information technology and in the Internet, one will only need to turn on the computer and go online" (Slyozko & Zahorodnya, 2016). This massive development in IT and in the Internet will ensure that software developers and IT programmers

were not only a scarce and costly resource in 2000, as noted by Cockburn and Williams (2000), but in 2020 and beyond.

In summary and to conclude this section, the seven fundamental changes apply to the data collection phase of this study. The aforementioned seven fundamental changes and the cost benefits to programme development, including the demand for IT professionals in the Fourth Industrial Revolution is motivation to pursue the teaching and learning of computer programming. It also enhances the rationale of the study. Further evidence is presented in Chapter Five on data analysis.

Section 2.2.2 defines commonly used terms during the course of the data collection phases in this study. The rationale in reviewing the literature regarding specific terminology emanated from interactions with participants in the study. When questioned, first-year IT students stated (amongst other replies) that they were studying programming, information technology, coding or simply how to “fix a computer”. Hence, the following terms are reviewed and defined.

### **2.2.2 Coding versus Computer Programming, coding literacy, Computer science and Information Systems**

The terms Coding and Computer Programming are used interchangeably (Maxwell, 2016). Maxwell states:

Computer programming, or coding, is a skill that will be in increasing demand in the next few years.

This year and the years to follow are the “next few years” that Maxwell predicted in 2016. The term coding has been popularised by the ever-popular Hour of Code. The word code has been incorporated in the URL <https://code.org/> (Code.org, 2019a). This website has the following facts listed: 30 percent of American students have accounts on code.org and 49 states in USA include Computer Science in the school curriculum. Code.org is an organisation which introduced the Hour of Code in 2013. This is an introduction to computer science; anyone can participate in or organise a one-hour coding experience during Computer Science Education Week (e.g. December, 2018). During the Hour of Code, students participated in self-guided tutorials that allow them to work at their own pace and skill level. In 2018, South Africa

registered 203 events and globally there were 220 545 events. This includes the institution from the data sample (Code.org, 2019). In South Africa, the term coding and not programming was included in the State of the Nation Address by President Cyril Ramaphosa (Ramaphosa, 2019).

In this study, the term IT Programming will be synonymous with the term Coding and would be used interchangeably. During the analysis phase of this study, participants interchangeably used the words coding and programming. This ambiguity is noted by Duncan, Bell and Tanimoto (2014): “It is not always clear what is meant by teaching coding (which is often used as a synonym for programming)”.

In this study, programming was observed to be the “hands on practice of writing computer programs” in a specific computer programming language. This is similar to research by Choy et al. (2005), as well as Guzdial (2014). They state that students in introductory programming classes are required to do a lot of hands-on practice in writing programs in order to comprehend the key concepts and develop the relevant skills.

According to Hutchison, Nadolny and Estapa (2016), the term coding originally applied to the act of creating in complex programming language; however, it is now also used to describe the creation of a sequence of instructions with tools basic enough for young children. An alternate and artistic definition of coding is:

Computer code is the language we use to communicate with technology and machines to convert our words into their commands. Coding is just the practise of arranging text and numbers in different permutations to tell a computer to perform a function (Fach, 2017).

This is the basis of the study, how to successfully teach and make the young novice first-year ToT programming students learn to arrange text and numbers in different permutations to ultimately tell the computer to perform a function. To conclude this section, the terms coding literacy or computational literacy are included. Clement (2018, p.126) argues:

Coding literacy or computational literacy gives us access to the rhetorical force of the term and concept of literacy, ways of looking at the social aspects of coding practices,



historical lessons of mass literacy efforts, barriers to access, and also the well-developed tools and theories of literacy education.

This study embraced the social aspects of teaching and learning, historical lessons, barriers to access and theories of education, confining the aforementioned aspects specifically to the teaching and learning of IT. Section 2.2.3 defines these aspects with very specific reference to this study.

### **2.2.3 Defining Computer Science, IT, Theory of Computer Programming and Information Systems (IS)**

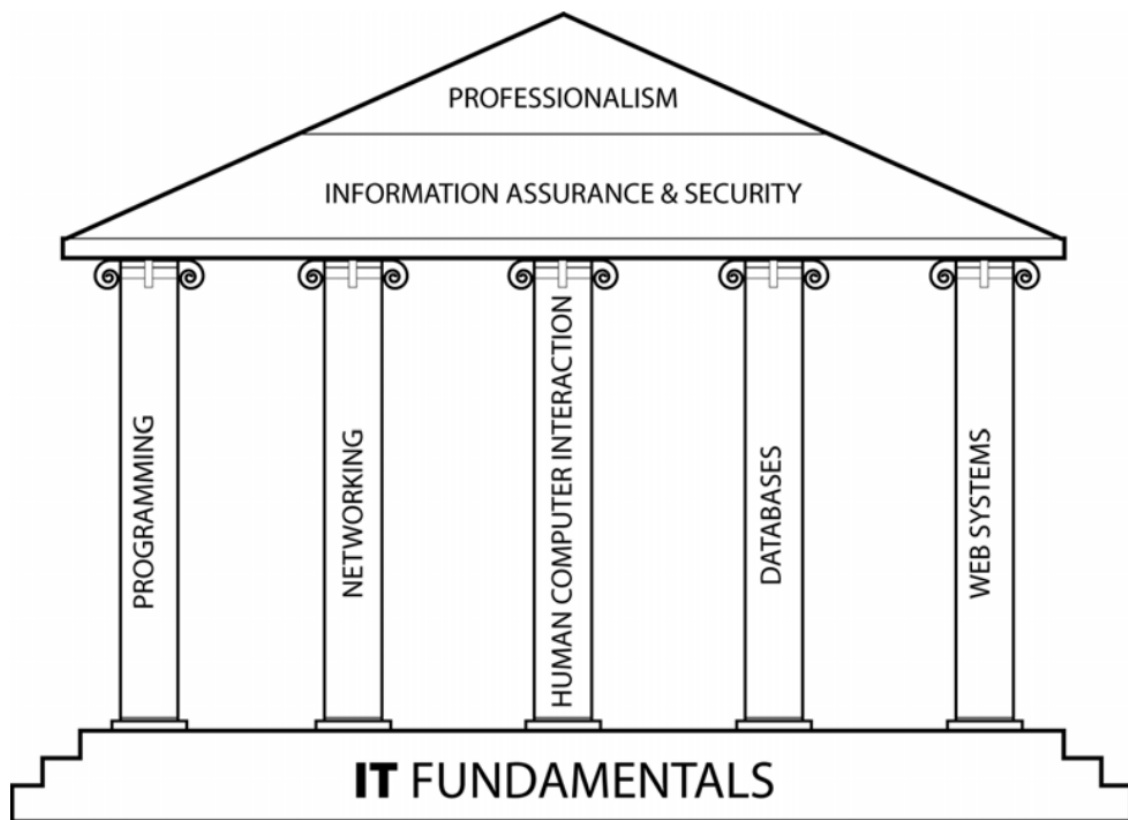
#### ***2.2.3.1 Computer Science***

Lunt et al. (2008) assert the following definitions of computer science as the study of the design and properties of algorithms, and their linguistic and mechanical realisation. In the field of computer science, programmers write code to give a computer step-by-step instructions on how to complete a task (Hutchison, Nadolny & Estapa, 2016). This definition aligns with the definition of coding above and asserts the interchangeability of the terms programming and coding.

#### ***2.2.3.2 Information Technology (IT)***

As an academic discipline, Information Technology focuses on meeting the needs of users in an organisational and societal context through the selection, creation, application, integration and administration of computing technologies (Lunt et al., 2008). Information Technology (IT), in its broadest sense, encompasses all aspects of computing technology. IT, as an academic discipline, is concerned with issues related to advocating for users and meeting their needs within an organisational and societal context through the selection, creation, application, integration and administration of computing technologies.

The pillars of IT include programming, networking, human-computer interaction, databases, and web systems, built on a foundation of knowledge of the fundamentals of IT.



*Figure 2-1 The Information Technology Discipline*

As depicted in Figure 2-1 (Lunt et al., 2008), the academic discipline of Information Technology (IT) includes programming, networking, human-computer interaction, databases and web systems. The rationale for including this figure 2-1 stemmed from discussions with the sample population. Many participants were quick to highlight the difficulties with IT programming. However, students appeared to succeed in the networking, website design and databases modules, as evident from course marks and throughput rates in the afore-mentioned modules from the ToT. This is also the basis for the inclusion of the words IT programming in the title of this study.

Based on anecdotal evidence, there must be a clear distinction amongst the difficulties, successes and satisfactory performances of IT modules. It is the perspective of this study that only then will IT academics and researchers reflect a decline in the following research outputs where it is stated that there exists an overwhelming and immediate difficulty in learning programming. This difficulty in learning programming has been differentiated by researchers as “known” difficulty and “perceived” difficulty.

Computer programming is known to be difficult (Kelleher & Pausch, 2005; Lahtinen et al., 2005; Govender, 2006; Macgregor, 2007; Havenga & Mentz, 2009; Guzdial, 2014; Kátai, 2015). Computer programming is perceived as a difficult subject (Govender, 2006; Macgregor, 2007; Havenga & Mentz, 2009). The data from the sample population also supported the claim that programming is considered difficult to teach and learn. This difficulty was observed to be both “know” and “perceived”. This difficulty is, however, not expressed with theoretical concepts; hence, the distinction between IT programming, theory of programming and information systems. This delineation, although not clearly demarcated during the data collection phase, was a commonality amongst the participants.

Participants claimed students perceived that programming was difficult and that the theory of programming or information systems was easy. This was evidenced in the assessment of computer courses. Students rote-learn theoretical programming concepts and regurgitate them in either formative or summative assessments. An important point to note was the difficulty expressed in the practical components of computer science programming versus the theoretical component of computer science.

Whilst, the scope of this study is predominantly IT programming, during the data gathering phases it became apparent that the research could not separate the theory of computer science from the practical programming components of computer science. Hence, the ensuing section discusses the theory of computer programming.

### ***2.2.3.3 Theory of computer programming /Information System***

Theoretical concepts of computer science embrace the definition of an Information System (IS). An IS is defined as:

An integrated set of components for collecting, storing, and processing data and for providing information, knowledge, and digital products. The main components of information systems are computer hardware and software (Zwass, 2018).

An IS is a set of interrelated components which collect, manipulate, store and disseminate data and information and provide a feedback mechanism to meet an objective (Reynolds & Stair, 2016).

IS as a field of academic study encompassing the concepts, principles and processes for two broad areas of activity within organisations: 1) acquisition, deployment and management of IT resources and services (the information systems function); and 2) development, operation and evaluation of infrastructure and systems for use in organisation processes (system development, system operation, and system maintenance).

The aforementioned definitions of IS includes IT programming concepts, for instance, computer programs collect/acquire (input) manipulate/deploy (process) and output/display data. Hence, if one has to broaden the definition of IT to include CS, coding literacy, programming and IS (as reviewed in the literature above), the notion of the perceived perception of programming as difficult would be minimised, ultimately resulting in improved throughput rates of first-year IT programming students.

In conclusion of this section, a discussion is presented regarding a distinction between computer programming/coding and the theory of computer programming and information systems. The discussion of the definitions in the literature emanated from the data collection phases which are associated with themes which emerged from the data analysis. Further discussion regarding data analysis is included in Chapter Five. Section 2.2.4 discusses teaching and learning theories. This section is directly associated to the concepts of SoTL as defined in the introductory paragraphs of this chapter.

#### **2.2.4 Teaching and Learning Theories**

An overwhelmingly common thread which permeated the entire research is that of student learning and more specifically, first-year ToT IT students learning to program a computer and the IT lecturer teaching and/or lecturing to the students. In this section, a review of the relevant literature concerning the main teaching and learning theories is presented. Whilst there is an arsenal of teaching and learning theories in the literature, only the most influential characteristics associated with teaching and learning theories which surfaced during the data gathering and analysis phases of the study are discussed. The theory is defined and evidence that lead to the application of the theory is outlined. Chapter Six critiques the teaching and learning theories with respect to answering the research questions which guided the study.

### 2.2.4.1 Instructivism

This learning theory is aptly applied in most programming lectures. The lecturer takes on the central role in the learning process (Sadeghi, 2015). Table 2-3 highlights a review of Instructivist learning theory based on data analysis of participants.

Table 2-3 Review of Instructivist Learning Theory (adapted from Sadeghi, 2015)

Dimensions	Instructivism	Evidence Based on
Educational paradigm	Behavioural approach Predetermined goals based on knowledge acquisition	The ToT participants had definite predetermined goals. This was highlighted in their study guides and lecture learning objectives
Role of instructor	Lecturer-centred teaching	The ToT participants were lecturer-centred
Value of errors	Fulfil an instruction course without making mistakes	This was evident during the “debugging” stages of problem solving
Origin of motivation	External interest and needs	The external interest was the throughput rate of students
Accommodation of individual differences	Single -faceted consideration on learners affective and physiological differences accommodated in learning environments	Individual differences were excluded
Learner control	Students learning programme is predetermined and fully controlled	Teaching learning and assessments were fully under the control of the lecturer
User activity	Students access various representations of content limited in predetermined paths	Content was limited to problem solving activities of the lecturer
Collaborative learning	limited support and no facilities for setting up collaborating learning	No collaborative learning evidenced

The challenge to implementing the Instructivism learning theory, in the context of this study, is the concept of the millennial student, who may have a superficial knowledge of IT programming and walks into the classroom carrying a smartphone or tablet with Internet connectivity and challenges the lecturer’s/instructor’s expert knowledge.

### 2.2.4.2 Constructionism

With respect to the learning theory of constructionism: Learners are not passive vessels but actively participate in their own learning. Table 2-4 highlights a review of constructionism based on data analysis of participants (Von Glasersfeld, 2012).

Table 2-4 Review of Constructionism Learning Theory (adapted from Sadeghi, 2015)

<b>Dimensions</b>	<b>Constructivism</b>	<b>Data analysis evidence based</b>
Educational paradigm	Constructivist approaches unfocused Goals based on knowledge transfer	Knowledge transfer was a key factor in the lectures; however, goals were highly focused
Experimental values	Learning practices based on real world	Students were afforded minimal opportunity to practice on real-world examples
Role of instructor	Student-centred teaching	No evidence of student-centred teaching
Value of errors	Mistakes as part of the learning process	Mistakes in programming were part of the learning process
Origin of motivation	Internal motivation and true desire	Little or no evidence of internal motivation and desire from students to excel in programming
Accommodation of Individual Differences	Multi-faceted consideration on learners' needs and preferences based on affective and physiological differences	No consideration of individual differences in students. The large number of students in lecture venues made it almost impossible to accommodate individual differences
Learner control	Students have power to choose what section, and/or what paths to follow.	Students did not have power to choose sections
User activity	Students engage in the learning process for creating and managing knowledge as main user	The lecturer was the main user of creating knowledge
Collaborative learning	Variety of different facilities and support are provided for setting up collaborative learning	Little or no collaboration

The challenges to implement the Constructivist learning theory was the lecturer's perception that first-year introductory IT students lacked the "tertiary student maturity" to co-construct and actively participate in their learning.

### **2.2.4.3 Social Constructivism**

Stemming from Constructivism is Social Constructivism, which refers to a learning theory which is built and socially negotiated through interactions with each other (Hodson & Hodson, 1998). Sir Ken Robinson (Robinson & Aronica, 2015) campaigns for changing education through talks, writing, advising, and teaching. He believes education must change because it is a stale environment in which most students do not really learn what they should or want to learn. How this happens makes all the difference, from the ground up. He termed it Ground Up Diversity, where students, and teachers create the change, not administrators or executives.

The teaching and learning of IT programming must incorporate the concepts of collaboration amongst stakeholders. Collaborative learning and pair programming (PP) are such concepts which emerged during the data collection phase. The sub-theme of PP is investigated in greater detail in Chapter Five. It was observed that ToT academics did not cater for characteristics associated with the Social Constructivist learning theory. This was understandable due to their lectures, the nature of their class sizes, number of students and the time period constraints for syllabus completion.

The classroom design and setup of the computers did not allow for Social Constructivism in the observed practical IT classroom. This is a challenge to future architects of learning spaces to design classrooms which cater for collaborative learning in pairs or groups of students.

### **2.2.4.4 Project based learning (PBL)**

Gary (2015, p.98) describes project-based learning (PBL) as engaging “students in sustained, collaborative focus on a specific project, often organised around a “driving question”. PBL is an approach particularly well-suited to achieve more durable, contextual outcomes for computing students. In PBL, students learn through the design, completion (and often ongoing iteration) of “projects”. PBL is in contrast to traditional “units” of “instruction”. The participants in the study indicated that first-year ToT students lacked the maturity to successfully engage in PBL. Furthermore, the 15-week semester duration posed a challenge to the successful completion of the syllabus. Participants did, however, indicate that there were successful attempts when implementing PBL in the third or final year of study for an IT diploma where the duration of the course extended across two semesters or the entire year of study.

#### ***2.2.4.5 Scaffolding and the gradual release of responsibility model***

Scaffolding provides students with a basic structure that can be used to solve tasks (Caspersen & Bennedsen, 2007). Cognitive scaffolding is what a teacher does when working with a student to solve a problem, carry out a task or achieve a goal which would be beyond their unassisted efforts (Flick, 1998). The teacher's support decreases as the students' competency increases (Pressley, 1996).

Instructional scaffolding is rooted in Vygotsky's (1978) concept of an expert assisting a novice or an apprentice (Eun, 2008). Scaffolding represents the helpful interactions between teacher and student which enable the student to achieve beyond independent effort alone. A scaffold is a temporary framework designed to support learning, as a temporary tool that can be removed as needed when the student progresses. Vygotsky states, "what the student is able to do in collaboration today he will be able to do independently tomorrow" (Vygotsky, 1987, p.211).

Through the use of scaffolds, students are able to learn the basic concepts and techniques and then apply them to more advanced tasks. In the field of education, the term scaffolding refers to a process in which teachers model or demonstrate how to solve a problem, and then step back, offering support as needed. Scaffolding was originally described as temporary adaptive support provided by a teacher or more knowledgeable other to assist a student to solve a problem that they would not normally be able to solve on their own (Smit & van Eerde, 2013; Van de Pol, Volman & Beishuizen, 2010).

Scaffolding is often described as "show me, help me, let me" (Heick, 2015). The stages "show me and help me" were clearly observed during data collection phases; however, the "let me" stage was at times absent or non-existent. To quote Alan Turing (considered to be the father of modern computing), "Programming is a skill best acquired by practice and example rather than from books" (Pişkin, 2019). Chapter six highlights this important teaching strategy.

#### ***2.2.4.6 The Gradual Release of Responsibility model***

This approach to teaching is similar to the scaffolding model, and was aptly applied in IT practical venues during a double-period lecture. The lecturer would demonstrate a programming concept in the first period of the double lecture and then support the students in their attempts to solve the task during the latter part of the lecture.



The gradual release of responsibility model is described as follows:

At the beginning of instruction, teachers model the learning task to be mastered. In doing so, they assume full responsibility for selecting materials for instruction, for explicitly demonstrating and modelling the task in such a way that students can observe the important aspects of task performance, and for completing the task so that students can see a finished product. As teachers continue to model and receive student feedback that indicates a beginning mastery of the task being presented, they gradually move from the central position of providing all the instruction and work toward guided practice, in which students assume more of the responsibility for performing the task.

During the guided practice phase, teachers and students have joint responsibility for learning the material. As the students become more independent through guided practice, teachers bring them to the stage where they are completely responsible for task completion, and they achieve independence in task performance. Of course, the objective in this model is to make students independent learners with careful teacher-centred instruction based on explicit modelling, guided practice with teacher-student joint responsibility, and independent practice and/or application by students.

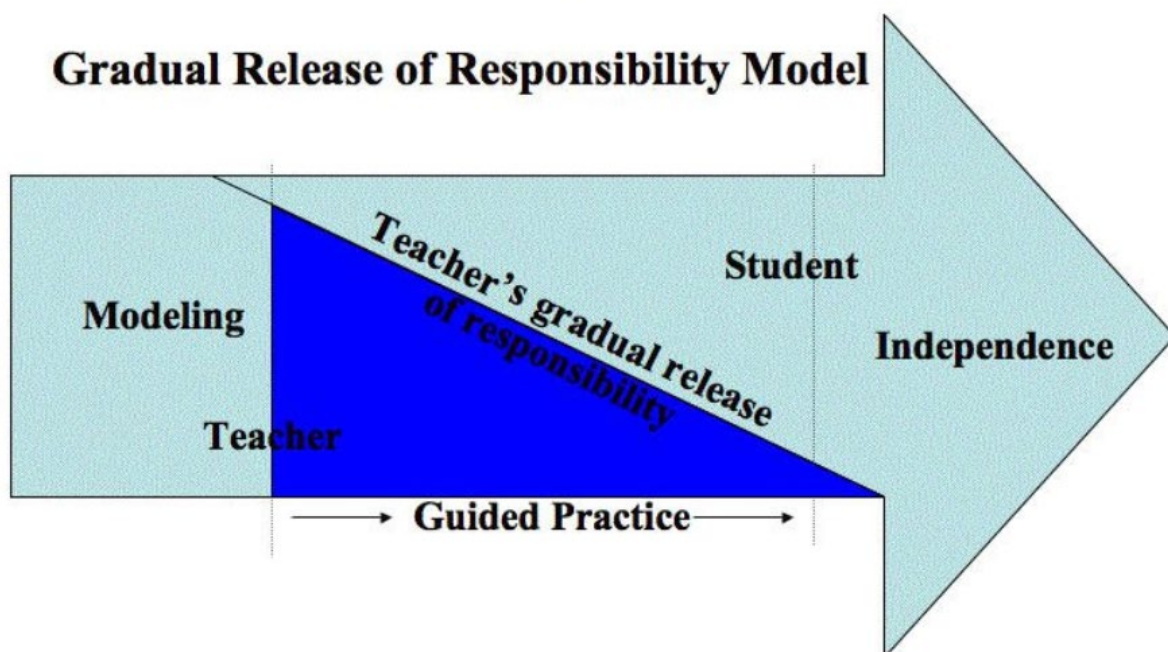


Figure 2-2 The transition to student independence (Pearson & Gallagher, 1983)

The rationale to include this mode of instruction arose from participants' roles as teachers versus lecturers versus facilitators. Participants were of the view that their role was as a lecturer and not a teacher. First-year IT students were expecting to be taught, similar to secondary school. This disjuncture in attitudes and perceptions of the academic and the learner may be a contributing factor to the low student throughput rate in first-year first semester.

#### ***2.2.4.7 Discovery learning***

Discovery learning is an inquiry-based, constructivist learning theory (Ültanır, 2012). The learner draws upon their own past experience and existing knowledge to discover new facts and relationships. Students manipulate and explore objects and questions, and perform experiments. In the context of the IT classroom, students who were repeating the course or who had prior knowledge from secondary school would display speed and accuracy in programming task completion (the experiments). This posed a challenge to the lecturer who now had to modify their task to cater for learning by discovery.

With respect to programming lessons and tutorials, it was noted that students were not given ample opportunity to allow for the "self-discovery" of programming concepts. Students were observed to be passive and complacent and expected the academic to provide the "discovery of programming concepts and syntax." Furthermore, it was noted that the majority of first-year IT students lacked intrinsic motivation. They were motivated by competition in obtaining the highest mark in assessment. Bruner (1966) states that the will to learn is an intrinsic motive that finds both its source and its reward in its own exercise. ToT academics must re-ignite the intrinsic flame in first-year IT students and not be constrained by syllabus completion, time constraints and student culture.

#### ***2.2.4.8 Learner-centered teaching (LCT) versus teacher-centered teaching (TCT)***

When instruction is learner-centred, the action focuses on what students are doing and not the teacher (Weimer, 2002). In this context teachers are guides, facilitators and designers of learning experiences. In teacher-centred teaching (TCT), the focus is on the teacher.

The data analysed from this study supported the notion of TCT. Onurkan, Aliusta and Özer (2017) also conclude that teacher and student roles are still very much teacher-centred. Their reasons quoted vary, including system-wide barriers hindering the adoption of LCT and teacher training offered and their findings draw attention to an urgent need for alternative teacher

training programmes which focus on changing teachers' traditional beliefs, enabling them to put theory into practice and adopt student-centred roles.

Ismail, Ngah and Umar (2010) state that the main cause of difficulty in understanding programming and coding is the "inactive involvement" of students during programming tutorials. Hence, if an academic increases the activity levels of involvement of students during programming tutorials the academic could theoretically reduce difficulty levels in understanding programming and coding. This study reached a similar conclusion with respect to student involvement: IT programming students must be actively involved during programming tutorials.

#### ***2.2.4.9 Meaningful Learning***

According to the meaningful learning theory, the most important factor influencing the learning process is the students' existing knowledge (Ausubel, Novak & Hansian, 1968). Meaningful learning is most commonly described as the intentional connecting of new information to anchored ideas or prior knowledge, particularly if the new knowledge is personally relevant and experiential (Ausubel et al., 1968; Driscoll, 2005; Greeno, Collins & Resnick, 1996; Ramsden, 2003). New knowledge to be acquired is thus related to previous knowledge.

This theory has direct consequences for this study, considering the sample population are first-year introductory students at a ToT. An important theme that emerged from the study was the challenge faced by the participants who indicated that many of the first-year students came from varying secondary school backgrounds, having varied computer literacy levels. Many students who gained admission to the programming course displayed high levels of computer illiteracy and this compounded the challenges of learning to program. The majority of the first-year IT students were faced with a lack of existing computer literacy knowledge and had to simultaneously learn computer literacy skills and programming syntax, logic and problem solving. This proved problematic.

Furthermore, this theory has direct implications for the IT academic when preparing programming tasks and assessments. The following programming tasks were directly extracted from the data: the programming tasks that dealt with motor vehicle and bond repayments or pension pay-outs. This task was totally irrelevant, impersonal and beyond the scope of first-year ToT students' existing knowledge. Tasks were not meaningful to the students.

Programming tasks that dealt with student issues, for instance student fees or student grades would be more meaningful to students. Meaningful learning is coupled to the theory of mastery learning. This would be discussed in the next sub-section.

#### ***2.2.4.10 Mastery learning***

In mastery learning, “the students are helped to master each learning unit before proceeding to a more advanced learning task” (McGaghie, 2015, p.558). McGaghie (2015) is quoted as stating that mastery learning has at least seven complementary features. Each feature would be stated together with the alignment and correspondence of the sample population to the mastery learning feature.

##### *2.2.4.10.1 Baseline (i.e. diagnostic) testing*

In this study, first-year IT students were not subjected to a baseline (diagnostic) testing procedure. The students were randomly assigned to groups. A suggestion from a participant indicated that an aptitude test be administered at the beginning of the semester to gauge students’ preparedness and experience with programming concepts. This group would be continuously monitored during the semester.

##### *2.2.4.10.2 Clear learning objectives, sequenced as units ordered by increasing difficulty*

There appeared to be clear objectives and sequencing of the curriculum in the faculty handbooks and course study guides. Formative assessments were implemented to gauge the concept of increased difficulty and measure the mastery of programming concepts.

##### *2.2.4.10.3 Engagement in educational activities*

There were deliberate IT programming skills practice that focused on reaching the curriculum objectives.

##### *2.2.4.10.4 The establishment of a minimum passing standard*

Test and examination scoring was implemented, including minimum checklist scoring. Furthermore, the concept of a Duly Performed (DP) mark ensured minimum passing standards in the programming course.

#### *2.2.4.10.5 Formative testing to gauge unit completion at a pre-set minimum passing mastery standard*

The pre-set minimum passing standards and course weightings varied amongst programming academics and programming courses; however, they were all consistent with the overall 50 percent pass mark.

#### *2.2.4.10.6 Advancement to the next educational unit given measured achievement at or above the mastery standard*

Students advanced to the next programming module based on achieving a minimum of 50 percent in a prerequisite module or having had exposure to the prerequisite module. Refer to Chapter One regarding the programming curriculum.

#### *2.2.4.10.7 Continued practice or study on an educational unit until the mastery standard is reached*

This was omitted in the sample population at the first-year programming course level. However, it was inferred from the IT academics/participants that students did gain mastery of programming concepts upon entry and at exit of their final year of study.

In concluding this pedagogy, mastery learning is to ensure that all learners accomplish all educational objectives with little or no outcome variation. However, the amount of time needed to reach mastery standards differs amongst learners. This posed a challenge to the sample population in this study. Unfortunately, the short teaching semester did not allow for mastery of basic programming concepts and this was viewed as a challenge to student learning and successful teaching practice. IT academics indicated that time constraints, large class sizes and syllabus completion were factors which hindered mastery of a unit before proceeding to a more advanced learning task. This did not auger well for student success or an IT academics teaching practice.

#### **2.2.4.11 Situated learning**

Situated Learning is an instructional approach developed by Jean Lave and Etienne Wenger in the early 1990s (Lave & Wenger, 1991), and follows the work of Ardichvili (2003) and others who claim that students are more inclined to learn by actively participating in the learning experience (Clancey, 1995). Situated learning is essentially a matter of creating meaning from

the real activities of daily living where learning occurs relative to the teaching environment (Stein, 1998).

The following situated learning example is applicable to learning IT programming employing cooperative education and internship techniques in which students are immersed and physically active in an actual work environment. Cooperative learning has opened up new possibilities for learning programming. This learning must occur at first-year level, where students are immersed in the software engineering industry. At the ToT in this study, only final year students were exposed to minimal internship and cooperative education experiences and not first-year IT students. The recommendation here is similar to student-teacher practice teaching. First-year IT programming students must be placed in “situated/simulated” learning environments as soon as possible.

#### ***2.2.4.12 Competency based learning***

This approach to learning focuses on actual, observable skills rather than concepts measured by traditional academic assessments. Henri, Johnson and Nepal (2017) state that competency based learning (CBL) is an outcome-based, student-centred form of instruction where students progress to more advanced work upon mastering the necessary prerequisite content and skills. CBL is similar to mastery learning. Other terms that have been used to describe CBL include self-paced, student-centred, self-directed, individualised or personalised instruction, outcome based, performance-based, standard-based, and proficiency-based education (Roe & Bartelt, 2015). Hence, there may be an overlap in terms of the aforementioned learning techniques.

The relevance to this study is based on the oral presentation of IT programming students. Students are generally required to present their software project in the presence of the IT academic and fellow group members and/or classmates. The oral presentation is an observable skill; however, the weighting for assessment purposes is minimal or non-existent in the first-year IT programming assessment criteria. Formative and summative assessment were not defined, nor part of the scope of this study. However, it was very much a part of the observable and self-reflective processes of the study, hence, the inclusion of the CBL section.

An equally important goal for CBL is to accommodate individual differences in pace or style of learning (Henri et al., 2017). Unfortunately, due to the large class sizes in the sample population, this was almost impossible to implement. Individual differences and implementing

varying teaching and learning styles were over shadowed by having 60 students in a practical lecture and approximately 150 students in a theory lecture venue.

#### ***2.2.4.13 Problem-based learning***

An approach to learning where the solving of important “problems” occurs, often through inquiry. Project-based learning catalyses the learning experience. Problem-based learning (PBL) uses problems to foster collaborative, self-directed learning (Savery, 2006). Six core principles of PBL include:

- 1) A learner-centred approach;
- 2) Small group work;
- 3) Teachers as facilitators;
- 4) Authentic problems to stimulate learning;
- 5) Problem-solving skill development;
- 6) Self-directed learning.

Lee and Blanchard (2018) conclude the following with respect to PBL: teachers who have taught with PBL have significantly higher levels of perceived competence and value PBL, perceive more support from peers, and perceive lower costs than did the non-PBL using teachers. In this study, the core principles of PBL as per Barrows (1996) were not implemented to maximum benefit. The teaching approach was still teacher-centred, there were large class size groups, academics were largely instructors and problems were spurious and supervised and dependent upon instructors. The above section represents established learning theories that were evidenced in the sample population during the data collection phases.

Section 2.2.5 identifies programming teaching methods as discussed by Szlávi, Zsakó and Törley (2016) and will continue with the review of the teaching methodologies as observed during the data analysis phase.

#### **2.2.5 Theories and methodologies of teaching and learning IT programming (coding)**

Although the following ten methods in teaching programming were identified and discussed, this section in the literature review identified methods of teaching programming that were observed during the data collection phases and in accordance to the criteria and principles of

grounded theory. The reference to the teaching theories and methodologies correlates to the major themes of the research as discussed in Chapter Five and Chapter Six.

The most widespread methods in teaching programming according to Szlávi et al. (2016) are:

- 1) Methodical;
- 2) Algorithmic oriented;
- 3) Data oriented;
- 4) Specification oriented;
- 5) Problem type-oriented;
- 6) Language oriented;
- 7) Instruction oriented;
- 8) Mathematics oriented;
- 9) Hardware oriented; and
- 10) Model oriented.

The teaching methodologies which emerged from the data collection phases based on the above methodologies are:

#### ***2.2.5.1 Language oriented and semiotic ladder***

As far back as 1978 there was a conference on programming languages. The Conference on History of Programming Languages described the 13 computer programming languages present at the time (Wexelblat, 1978). This study does not focus on programming languages, but rather on a strategy that a teacher can adopt to teach a particular programming language.

This method of teaching programming is associated with a particular programming language and the associated syntax. During the interview session, question 15 specifically sought the programming language associated with teaching introductory programming at the ToT. The question was:

Which programming language do you normally use when teaching programming and why?

The basis of the method is that it teaches one particular programming language and it introduces programming knowledge with the help of this language. Programming was taught primarily



with the particular programming language's concepts and syntax, for instance, the programming language C++ was taught using only C++ syntax and in a command driven console-based paradigm. The disadvantage of learning to program in one language as stated by Szlávi et al. (2016) is the difficulty in having to change over to another programming language.

This model is similar in nature to the model proposed by Carbone and Kaasbøll (1998). They define a pedagogic model for programming in the object-oriented style. It was defined as the semiotic ladder. It was based on the language-like features of computer tools: programming languages, modelling languages, formatting, formula and search instructions. The teaching and learning sequence start from syntax and progresses to the semantics and pragmatics of the language-like tools. This is based on the principle that syntactical knowledge is needed to express everything and, therefore, should precede learning of the meaning of the language constructs. When the meaning is acquired, the students can start to learn how to use the language for specific purposes, referred to as "pragmatics".

Papert (1980) states that programming a computer means nothing more or less than communicating to it in a language that it and the human user can both "understand". Learning languages is one of the things children do best. Every normal child learns to talk. Why then should a child not learn to "talk" to a computer?

Stemming from the above, the assumption is that students who are taught more than one programming language at an introductory level tend to become confused and "talk" to the computer in more than one syntax and programming structure. The sample population were taught more than one programming language in different course modules.

#### ***2.2.5.2 Instruction-oriented***

This method is based on a general language type instead of one particular language and defines general language elements, according to the Neumann principle (Szlávi et al., 2016):

- Assignment, expressions;
- Reading, writing;
- Branching (IF-statement, Case-statement);
- Iterations (counting, conditional pre- and post-testing);
- Procedures;

- Functions, operators; and
- Modules.

This methodology of teaching, coupled with language-oriented methodology, was adopted by the academics at the ToT and an analysis of study guides and tutorial material indicated a close resemblance to the aforementioned sequencing of language elements. A critique of the adoption of the aforementioned methodology is based on the theories of learning. Students did not appear to have mastered a particular concept, for instance pre-testing iterations, and were introduced to post-testing iterations. Students were not given sufficient time to engage in the theories of learning as discussed in Section 2.2 above, for instance: self-discovery learning, competency learning, scaffolding, and meaningful learning.

ToT academics, lecturers and professors need to slow down. Berg and Seeber (2016) state that slow professors act with purpose, taking the time for deliberation, reflection and dialogue, and cultivating emotional and intellectual resilience. ToT staff must ensure that students are presented with an opportunity to deliberate, reflect and cultivate an attitude of resilience in dealing with programming syntax, logic and run time errors.

#### ***2.2.5.3 Problem type oriented/Mathematics oriented method***

In problem type oriented methodology a series of problems based on one another have to be solved. The programmer deals with the whole program. The instructor would start from a definite problem which originated from classical mathematics (number theory: divisibility; prime numbers; prime factor expansion; Fibonacci series; etc.).

Typically, during the observational data collection phase, the introductory programming courses dealt with problems from the secondary school mathematics curriculum. Many students struggled with not only programming constructs and syntax but also recalling the mathematical concepts and constructs from secondary school. Furthermore, the entry requirement to study IT at the ToT was a minimum pass in Grade 12 mathematics, and with many students gaining entry with a minimum pass in Grade 12 mathematics, the problems associated with learning to programme mathematical functions and equations is compounded. It is a basic tenet of teaching pedagogy that an academic first establishes the relevant previous knowledge of his learner. Therefore, if the methodology of the problem type/mathematics

oriented methodology is to be adopted, the relevant mathematical functions must be taught to the programming students.

This study did not concentrate on admission requirements at ToT. However, suffice to state Van der Westhuizen and Barlow-Jones (2015) conclude that the mark achieved for school mathematics cannot be considered as a valid admission criterion for programming courses in the South African context. If, however, the ToT had to exclude grade 12 mathematics as an admission requirement, anecdotal evidence would indicate and surmise a further decline in first-year programming student throughput and performance rates.

#### ***2.2.5.4 Based on a model***

In this method models are introduced to the students (e.g. algorithms, program codes) and they obtain information about programming by studying them. Students then produce new programs by modifying the existing code. Experimenting plays a very important role here, pushing programming knowledge into the background. Students modify the code that they have been given by the instructor/academic. They evaluate the program output. Unsatisfactory results would indicate that the student will have to continue experimenting with the program code. This approach is similar to Nicholson and Fraser's (1997) alternative “code-provided” approach. In this approach students are provided with existing code, and the basic syntax and semantics of the language during the programming lesson.

The student would be able to follow the execution of pieces of code. Students then modify or write additional code. East et al. (1996) believe that students may not develop into good programmers unless they see “good programs”, and that it is the responsibility of teachers to ensure that this occurs. The instructors in the sample population adopted this approach during their programming lessons; however, the assessments required students to program without existing code and assessment tasks were more complex. Furthermore, providing students with code created a culture of dependency and rote learning was evident in assessments.

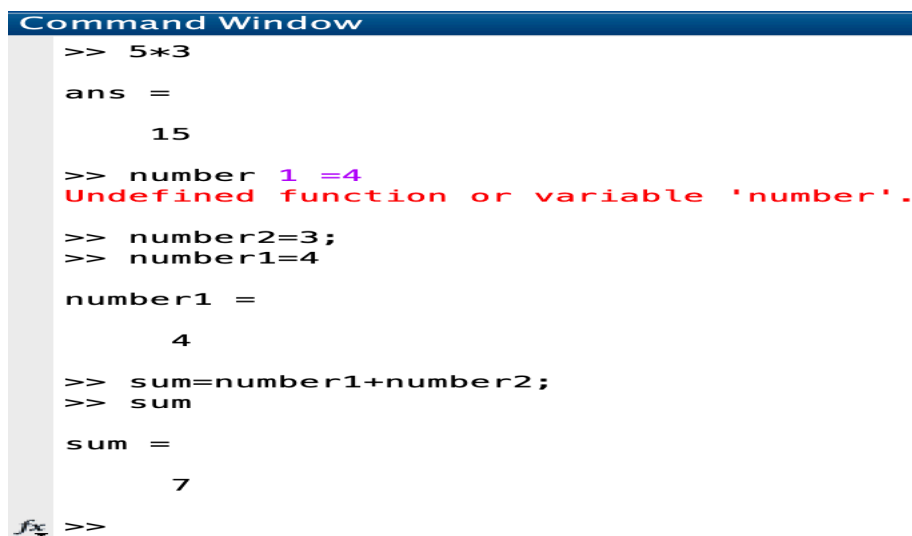
#### **2.2.6 Approaches adopted by IT academics at the ToT when teaching programming**

To conclude the section on teaching methods, I will discuss the one approach adopted by all academics in the sample population. I will review portions of programming code from first-year introductory programming classes. Nicholson and Fraser (1997) define it as the traditional/start small approach. Students were taught to write small, simple programs and they

gradually progressed to more complex, structured programs. This method was aligned to the instruction oriented methodology and the model based method of teaching programming. Instructors and students wrote portions of code, for instance, based on assignments and expressions.

### 2.2.6.1 Portions of Code

A student would be required to write portions of code which displayed a single line of an expression on the console, thereafter, increasing the number of console output lines. For instance, refer to the figure 2-3.



```
Command Window
>> 5*3
ans =
    15
>> number 1 =4
Undefined function or variable 'number'.
>> number2=3;
>> number1=4
number1 =
     4
>> sum=number1+number2;
>> sum
sum =
     7
fx >>
```

Figure 2-3 Compiled in MatLab (MathWorks, 2019a)

In Figure 2-3, students are introduced to maths concepts, variable declaration, language syntax and assignment statements. Students would now be expected to declare new variables, assign values to the variables, perform computations on the variables and display the output. Note the deliberate error, undefined function or variable, emphasises the rules regarding the naming of variables (variable names cannot have a space between characters).

Figure 2-4 refers to C++ code which requests from the user an integer value, displays the value and computes the output by doubling the value of the number (cplusplus.com, 2019).

```
// i/o example

#include <iostream>
using namespace std;
```

```

int main ()
{
int i;
cout << "Please enter an integer value: ";
cin >> i;
cout << "The value you entered is " << i;
cout << " and its double is " << i*2 << ".\n";
return 0;
}

```

*Figure 2-4 C++ code*

In Figure 2-4 students were introduced to the following concepts:

Comment statements // *i/o example*

iostream uses the objects cin, cout and clog for sending data to and from the standard streams input, output.

int main() indicates main takes an unspecified number of arguments.

int i refers to a variable declaration. This variable declaration is not required in Hypertext

Preprocessor(PHP) code, (refer to the example below)

cout refers to the display of a string of characters and i\*2 refers to the doubling of the number.

This example is language specific to the C++ programming language. It uses mathematically oriented/problem solving methodology. Note the comparison with respect to the MatLab example above. The instructor had to incorporate far more language specific syntax to achieve similar results.

Branching (IF-statement, Case-statement); The figure 2-5 is PHP code highlighting the IF-elseIF statements (w3schools.com, 2019).

<pre> &lt;!DOCTYPE html&gt; &lt;html&gt; &lt;body&gt;  &lt;?php \$t = date("H"); echo "&lt;p&gt;The hour (of the server) is " . \$t; echo ", and will give the following message:&lt;/p&gt;";  if (\$t &lt; "10") {     echo "Have a good morning!"; } elseif (\$t &lt; "20") {     echo "Have a good day!"; } else {     echo "Have a good night!"; } ?&gt;  &lt;/body&gt; &lt;/html&gt; </pre>	<p>The hour (of the server) is 03, and will give the following message: Have a good morning!</p>
--	--

Figure 2-5 PHP code highlighting the IF-elseif statements (w3schools.com, 2019)

In the Figure 2-5 students are introduced to the following concepts:

<!DOCTYPE html> refers to comment statements, similar to the C++ statement doctype html

Note the use of < > and the non-declaration of variables as compared to the programming language C++ presented in the previous example

$t = \text{date}("H")$  is an assignment statement, similar to  $\text{number1} = 4$  from the matlab example.

Due to time constraints, instructors and academics alike tend to skim through selection statements and the issue of selection statements is compounded when students are faced with a problem-type example which employs all of the programming constructs of the programming language. This lead to confusion and frustration on the part of the ToT programming students and they concluded that programming is difficult.

### 2.2.6.2 Deep learning

In describing the portion of code to highlight iterations, the following example is described from Matlab. It is interesting to note that the simple While... Loop as depicted in figure 2-6 (lines 61..68) is used to continuously classify images from the camera. This example was chosen for its application to deep learning concepts.

Deep learning is defined as,

A form of machine learning that enables computers to learn from experience and understand the world in terms of a hierarchy of concepts. The computer gathers knowledge from experience, there is no need for a human computer operator to formally

specify all the knowledge that the computer needs. The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones; a graph of these hierarchies would be many layers deep (Goodfellow, Bengio and Courville, 2016).

The rationale to include “deep learning” at a first-year introductory programming course is to attract and grasp the interests of young, impressionable first-year programming students and to embrace the Fourth Industrial Revolution. This example directly aligns to the model based methodology of teaching programming. This portion of code could be included in a young programmer’s IT project, encompassing deep learning concepts of programming. The concepts are novel, intriguing, and based on Science, Technology, Engineering and Mathematics (STEM) concepts.

And now, more than ever before, *educators* are making extensive use of *physical computing* devices as a direct means to teach and inspire a generation of students - and to prepare them for a society where Internet of Things (IoT) will be the norm (Devine et al., 2019).

The physical device used in the example below is a webcam and the neural network GoogLeNet (MathsWorks, 2019b).

```
%% Classify Webcam Images Using Deep Learning
% This example shows how to classify images from a webcam in real time
% using the pretrained deep convolutional neural network GoogLeNet.
%
% Use MATLAB(R), a simple webcam, and a deep neural network to identify
% objects in your surroundings. This example uses GoogLeNet, a pretrained
% deep convolutional neural network (CNN or ConvNet) that has been trained
% on over a million images and can classify images into 1000 object
% categories (such as keyboard, coffee mug, pencil, and many animals). You
% can download GoogLeNet and use MATLAB to continuously process the camera
% images in real time.
% GoogLeNet has learned rich feature representations for a wide range of
% images. It takes the image as input and provides a label for the object
```

% in the image and the probabilities for each of the object categories. You  
% can experiment with objects in your surroundings to see how accurately  
% GoogLeNet classifies images. To learn more about the network's object  
% classification, you can show the scores for the top five classes in real  
% time, instead of just the final class decision.

```
54 %% Continuously Classify Images from Camera
55 % To classify images from the camera continuously, include the previous
56 % steps inside a loop. Run the loop while the figure is open. To stop the
57 % live prediction, simply close the figure. Use |drawnow| at the end of
58 % each iteration to update the figure.
59 - h = figure;
60
61 - while ishandle(h)
62 -     im = snapshot(camera);
63 -     image(im)
64 -     im = imresize(im,inputSize);
65 -     [label,score] = classify(net,im);
66 -     title({char(label), num2str(max(score),2)});
67 -     drawnow
68 - end
```

Figure 2-6 Classify Webcam Images Using Deep Learning

Note the programming construct “While Loop” in lines 61 to 68. Line 61 contains the pretest `ishandle`, a Boolean condition, and lines 62 to 67 contains the body of the loop. Line 68 terminates the loop. This programming construct combines the concepts of deep learning with the simple loop structure of a while loop. The portion of code classifies the images from the camera. Students are immediately taken from simple properties of a while construct loop to concepts of deep learning image classification. This also encompasses attributes from situated learning. Students create meaning from captured images from their daily life experiences.

### 2.2.6.3 Procedures and Functions

The following portion of code depicts the concepts of a procedure in IT programming.



## CREATE PROCEDURE

The `CREATE PROCEDURE` command is used to create a stored procedure.

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

The following SQL creates a stored procedure named "SelectAllCustomers" that selects all records from the "Customers" table:

### Example

```
CREATE PROCEDURE SelectAllCustomers
AS
SELECT * FROM Customers
GO;
```

Figure 2-7 Create Procedure (w3schools.com, 2019)

Procedures and functions are key concepts in programming and align, for the purposes of this study, to threshold concepts from Chapter Three and Chapter Four. At the ToT, this threshold concept was taught towards the end of the semester. However, the key concept is entrenched in the first introductory “hello world programming” lecture. To avoid complexity, and sometimes confusion that arose when students attempted to combine the procedures and function taught at the end of the semester, it is my opinion that students should be introduced to procedures and functions from the first week of lectures. Students also felt overwhelmed and unable to combine assignment statements, looping and selection structures. Portions of code depicting working procedures are mimicked by IT programming students. Students gain competency by emulating the procedures from the instructor and ultimately gaining concepts of mastery learning whereby they can create their own procedures and functions.

### ***2.2.6.4 Microsoft MakeCode: inclusion of a physical computing device***

To conclude the review of the introductory methodologies adopted by programming instructors and academics, I include Microsoft MakeCode, an open-source platform for embedded devices such as the micro:bit that enables the development of embedded applications by non-expert programmers. This platform consists of two major components: (1) Microsoft MakeCode (www.makecode.com), a web app providing a beginner Integrated Development Environment (IDE) for embedded systems; and (2) CODAL (Component-Oriented Device Abstraction Layer), an efficient C++ runtime with high-level programming abstractions. In the sample population, the first-year students were non-expert programmers, with the exception of a few first-year repeating students or students who originated from private secondary schools. Micro:bit is one visual programming tool that uses a physical computing device that an IT

academic may use to introduce programming concepts to first-year introductory students to programming concepts. A similar visual programming tool is Scratch, discussed in the next section.

In the sample population, none of the introductory first-year academics used a physical computing device to introduce programming content to their students. Neil Gershenfeld states that if you give ordinary people the right tools, they will design and build the most extraordinary things (Euchner, 2018). It is the view of this study that first-year academics must incorporate “physical computing devices” in their programming lectures. For instance, Microsoft MakeCode.

Figure 2-8 is a typical programme in Microsoft MakeCode and has been selected and included to complement the php if elseif statements from the section above.

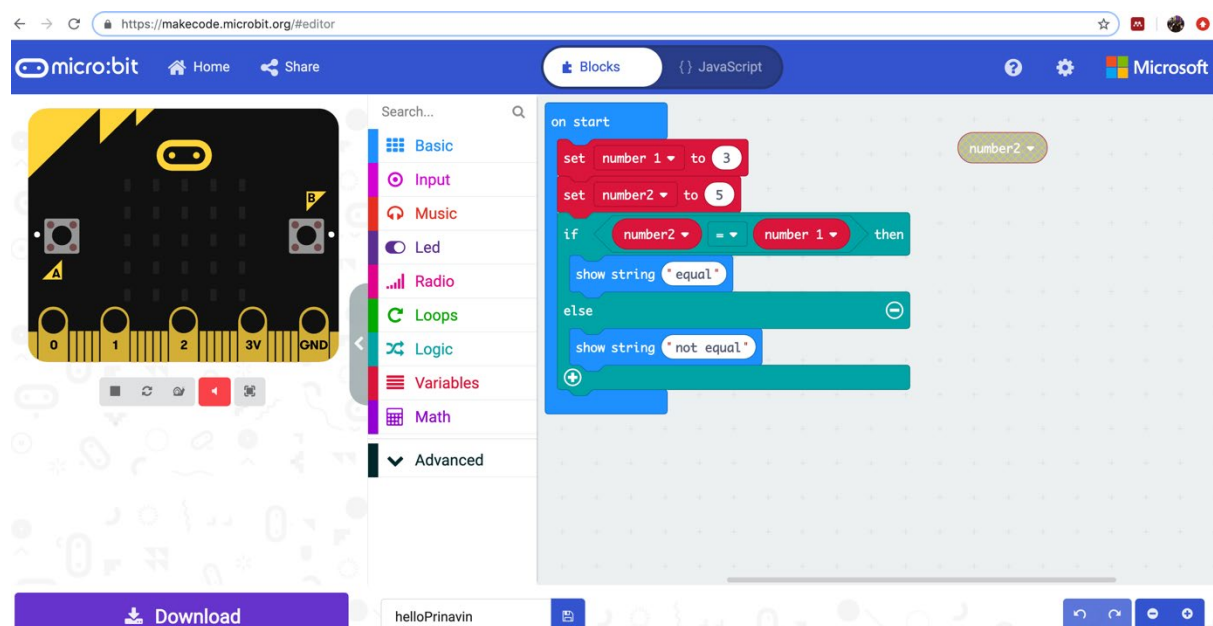


Figure 2-8 Programme in Microsoft MakeCode

In Figure 2-8, the process of coding was a simple select a programming construct from the web-based drag-and-drop visual programming with in-browser compilation, device simulation and driverless USB flash drive microcontroller programming. The output was almost immediate. The web app has five sections: the menu bar allows switching between the two editors; the simulator shows the micro:bit board and provides feedback on user code executed in the browser; the toolbox provides access to device-specific Application Program Interfaces

(API)s and programming elements; the programming canvas is where editing takes place; the download button invokes the compiler, producing a binary executable.

The web application encapsulates all the components needed to deliver a programming experience for microcontroller units (MCUs), free of the need for a C++ compiler for the compilation of user code. This is of importance to first-year introductory IT programming students and their lecturers. Valuable teaching time is lost due to academics having to teach students the intricacies of a compiler and the debugging of programming errors. The micro:bit web application is written in TypeScript and incorporates the TypeScript compiler and language service as well.

#### ***2.2.6.5 Scratch programming***

The final review of a visual programming tool is Scratch. This programming tool was incorporated in the CS50 Harvard extension school online course (Harvard,2019). I was fortunate to have registered and studied the course curriculum (refer to Appendix I). More specifically, in 2016 the Harvard online CS50 course introduced students to programming using the Scratch visual programming tool. The screen shot below is a project submitted in fulfilment of course requirements. This project incorporated the concepts of project based learning as discussed in section 2.2.3.4 and problem-based learning as discussed in section 2.2.12. Furthermore, it incorporated all the fundamental programming concepts (assignment, loops, decision making). The first-year ToT academics lacked a visual programming tool, and hence missed an important teaching strategy to introduce programming concepts to first-year IT ToT students.

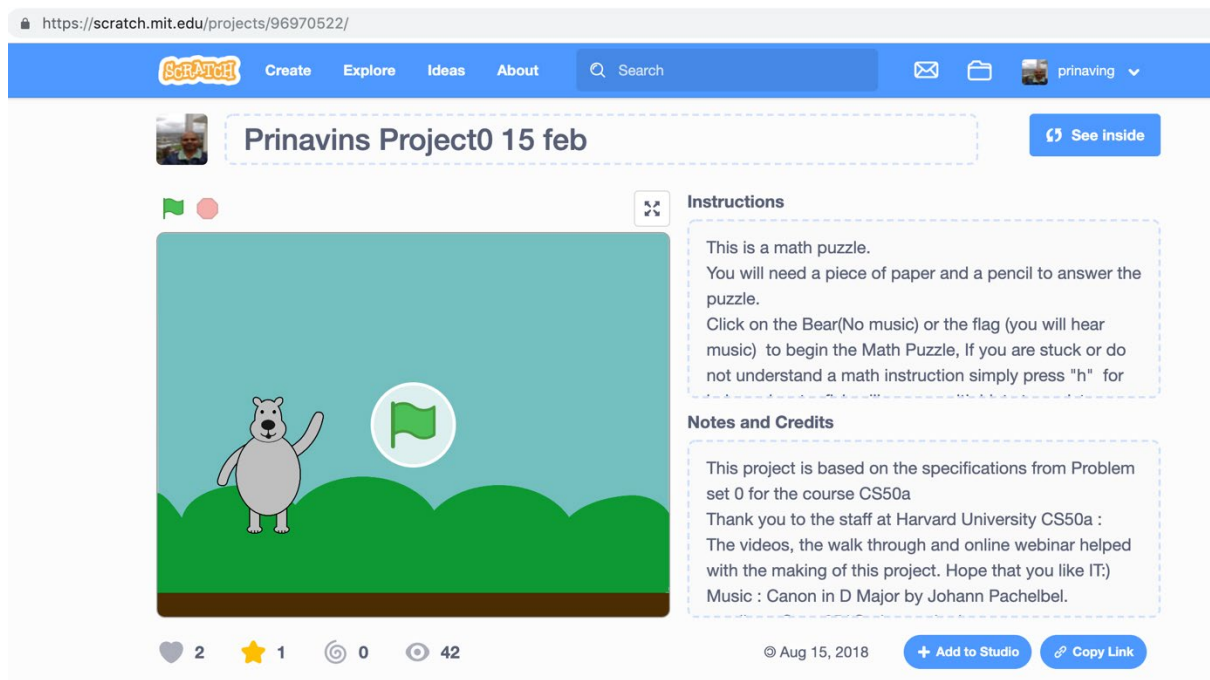


Figure 2-9 Prinavin's Scratch Project 15 Feb (Govender, 2018)

All of the programming portions of code were specifically included in this review because of one common characteristic. They were chosen because they were from the realm of the real world of ToT IT students. According to Gatto (2003), teachers should choose the real world over the classroom. Students do not learn to live or survive in a classroom. They learn to survive in the real world, so the concept of underground education challenges academics in any walk of life to give students the tools with which to live and breathe in the world around them. If the lesson must be taught, then teach it thinking of who students might become.

This has implications for IT programming at first-year level as follows: IT academics and learners teach and learn in a classroom devoid of physical and real-world issues facing the citizens around them. If the practical lesson requires an application to develop and design a reservation system for a client, then the lesson must be taught with the client requirements and specifications. This academic teaching of programming must focus on the software industry. Researchers at Microsoft, Begel and Nagappan (2008), found that most research focuses on academic environments, and there are limited studies about industry. This study has focused on academia – tertiary institutions of learning; however, academics must always bear in mind that the ultimate client of programming is the software industry.

### **2.3 Conclusion**

This chapter was structured such that the literature surveyed offered consideration to relevant aspects relating to the teaching and learning of IT programming, collaboration, students' experiences of learning programming in higher education, as well as strategies for teaching IT programming. This broad strategy has been outlined, describing the elements and dimensions that influence student learning and academics teaching IT programming at the ToT.

This broad strategy and the effective use of pedagogy in support of IT programming teaching by integrating methodologies with an appropriately considered pedagogical approach was central to teaching and learning programming in this study. Literature reviewed in this chapter sought to explore how academics interpreted the concepts of programming, coding, IT and the pedagogy of teaching programming. Improving the quality of education is a broad policy objective in South Africa (Badat, 2004).

The shortcomings and problems associated with teaching and learning IT programming suggest that further research is required to explore strategies to provide solutions and hence negate these. The academics' responses are further elaborated on in Chapter Five and Chapter Six, where the study formalises and creates greater awareness amongst academics with respect to the pedagogy of IT programming.

The aforementioned studies have shown that integration of IT fundamentals (information systems, networking, human computer interaction, databases, web systems) makes a positive contribution to teaching and learning of the IT discipline. By incorporating the full utilisation of disciplines by IT academics, the IT learners' difficulty in programming is minimised. Apart from knowledge of the IT discipline; knowledge of educational theories, strategies and methodologies is crucial in the IT discipline.

The review of the literature in conjunction with the data collection from ToT academics revealed that the pedagogy associated with the ToT's IT curriculum has been used to either generate new software projects and/or to support teaching and learning of programming concepts. Relatively few ToT academics were keen on mixing programming languages and programming teaching strategies (mathematical model, project base, student-centred, model based, etc.) in an introductory programming course as a teaching strategy, and even fewer were

comfortable with combining teaching strategies that incorporated the basics of programming concepts (assignment, iteration and selection) from different programming languages.

While the adoption rate of deep learning, artificial learning and machine learning gaining momentum, the ToT curriculum has been increasing rapidly. Little is known about how this teaching strategy benefits teaching and learning or how it brings about change to current classroom/lecture room practice.

In this study the researcher investigated participants' (learners' and academics') perceptions and experiences of teaching and learning IT programming. The study intends to add to previous research in education, since to date students' and academics' experiences and the efficacy of teaching and learning IT programming have been subjected to limited research, especially at the ToT level in the KwaZulu-Natal region of South Africa.

## **Chapter Three: Theoretical, Conceptual and Philosophical framework**

### **3.1 Introduction**

The purpose of this chapter is to describe the research design and the research paradigm for this study. The research design refers to the method(s) and technique(s) used to collect, analyse and interpret data and a paradigm refers to how a researcher would view that which they study (Du Plooy, 2009). Different paradigms can be used to explain or predict phenomena, and, in this study, paradigms refer to the way in which the research was conducted. In this study the interpretive paradigm was used to explain the behaviour of the research participants.

The research design encompassed the strategies of grounded theory (Charmaz, 1996), threshold concepts (Meyer & Land, 2005) and practitioner research (Anderson & Herr, 1999; Moore et al., 2001) The key features of threshold concepts are that they are: transformative, integrative, irreversible, bounded and troublesome. The importance of threshold concepts to this study is described with specific examples from the participants' data. These concepts are used, with specific examples, to demonstrate how teaching and learning of IT programming concepts are related to the nine considerations of threshold concepts.

The research design also encompassed the concepts of practitioner research in that I reflected on my role as an IT lecturer at a ToT, IT student and a computer studies teacher at a secondary school. I describe myself, the researcher, within the context of observing, analysing and narrating fellow IT colleagues and having to constantly introspect on my own philosophies of teaching practice and learning of IT programming. This introspection is a reflexive account that conforms to the characteristics of a living theory research project, which is described by Whitehead and McNiff (2006) as a form of self-study practitioner research. Hence, I draw distinctions as a student, a teacher, lecturer and ultimately as a researcher.

This necessity to describe the following: the research design; the research paradigm; threshold concepts; grounded theory; and self-study/practitioner research, permeates the entire research study and the concepts and design are embedded in the themes and categories which emerged during the data analysis phase.

Furthermore, my research of my academic practice developed out of my involvement in teaching and studying IT programming, and even though I report on the results and processes of the participants in this study, it is inclusive of the perspectives gained from practitioner research. Therefore, this study can also be seen as self-study research (Bullough & Pinnegar, 2001; Kosnik, Lasonde & Galman) in that I studied myself in relation to the other participants in the study. This is evident in Chapter Six when analysing the teaching styles of the participants and in answering the research questions; I had to first self-introspect and determine whether I conformed to or contradicted the observed characteristics of the participants. The specific relationship between self-study research and the research study as a form of practitioner research is clarified in the following section.

### **3.2 Chapter overview**

The theoretical, conceptual and philosophical framework categories of this chapter examine and emphasise:

3.2.1 Constructing knowledge within an interpretive paradigm;

3.2.2 Frames of enquiry: Qualitative research;

3.2.3 Self-study and practitioner research;

3.2.4 Personal reflexivity and prior experience;

3.2.5 Threshold concepts;

3.2.6 Grounded theory

An explanation of the interpretive paradigm is provided in section 3.2.1.

#### **3.2.1 Constructing knowledge within an interpretive paradigm**

This study was constructivist-based within an interpretive paradigm. The ontology associated with the interpretive paradigm refers to the “internal reality of subjective experience” (Terre Blanche & Durrheim, 2002). Furthermore, this paradigm focuses on the construction of knowledge in situations where problem-solving is required, in this case programming tasks with specific reference to the teaching and learning of IT programming of first-year ToT students.

Qualitative research methods are suited to this paradigm. Babbie (2015) states that “Qualitative research is the collection, analysis, and interpretation of data by observing what people do and



say and it's a generic approach in social research according to which research takes as its departure point the insider perspective on social action". This approach is concerned with seeing the world from the perspective of the research participants. Individuals are at the centre of the research process and people are actively constructing their social world, giving personal meaning to their situations and events, and making informed decisions to act in particular ways.

The interpretive approach was the choice for this study because it was concerned with descriptions that produce deep understanding and emphasised interpretation. Furthermore, it correlates with Terre Blanche and Durrheim (2002), who state that "Interpretive research methods try to describe and interpret people's feelings and experiences in human terms rather than by quantification and measurement".

The research participants were academics who taught a programming subject to first-year introductory ToT students. The researcher found the qualitative, interpretive approach suitable for the study because the aim of the study was to understand the teaching methods and learning experiences of academics and students in tertiary institutions and how they constructed their relationships while experiencing the teaching and learning of computer programming. This meant studying academics in their 'natural educational environment' (lecture venue/computer laboratory) rather than under artificially created conditions.

### **3.2.2 Frames of enquiry: Qualitative research**

Qualitative research was adopted for this study. The purpose of qualitative research is to understand and explain participant meaning (Morrow & Smith, 2000). More specifically, Creswell (1998) defines qualitative research as an inquiry process of understanding based on distinct methodological traditions of inquiry which explore a social or human problem. The researcher builds a complex, holistic picture, analyses words, reports detailed views of participants, and conducts the study in a natural setting (Creswell, 2014).

The themes of qualitative enquiry are naturalistic, holistic and inductive.

The characteristics of the naturalistic theme of qualitative inquiry are listed below, together with their relevance in this study:

1. Studies real-world situations as they unfold naturally.

The study dealt with a real-world situation. A ToT lecturer teaching first-year IT programming students. The study was conducted in the natural settings of a lecture theatre.

2. Non-manipulative, unobtrusive and non-controlling openness to whatever emerges. In the natural setting, I had unobtrusive face-to-face interaction with participants. I did not influence the teaching strategy adopted by the participants nor dictate the implementation of one strategy over another.
3. Avoids predetermined constraints on outcomes.
4. The researcher did not put in place any notable constraints that would have affected the outcome of teaching programming.

### **3.2.3 Self-study and practitioner research**

This research can be regarded as practitioner research (Anderson & Herr, 1999; McAllister & Stockhausen, 2001; Zeichner & Noffke, 2001) in that I inquire into the transformations of my academic practice as a ToT lecturer and researcher. I draw distinctions in this study with respect to my researcher practice, my academic practice and my student/learning practice. I did a self-study of my teaching of IT programming to ToT students, and ultimately there is a personal transformation which has occurred. This research transformation is presented in Chapter Six where I focus and report on the teaching styles of the participants and the personal transformations that took place in my identity as researcher and academic to IT students. The challenges faced by the participants and my ultimate recommendations have transformed my beliefs in the teaching and learning of IT programming to first-year ToT students.

My research of my academic practice (practitioner research) developed out of my involvement, and experience in teaching and learning IT programming rather than the research study growing out of my attempts to research a particular method (discourse analysis or case study or ethnography discourse analysis or experimental) and then look for a problem to apply the method to. McWilliam (2004) cautions against efforts to reduce practitioner research to the level of a method and further states, “Thus the overwhelming tendency remains that of working backwards from a method (‘I’m going to do a case study on something’) to a do-able problem as defined by that method” (McWilliam, 2004, p.120). The afore-mentioned processes conform to the genre of self-study research. There is therefore a constant movement between self-study and self-improvement as an academic and the research study; both from the personal imperative and the research imperative. This requires a framework to set out the research

imperatives and how knowledge can be gained thereof. To discuss this personal imperative requires a focus on the ontological definition of self vis-à-vis this study.

Feldman, Paugh and Mills (2004) suggest that the self as a focus of a study is a “distinguishing characteristic of self-study as a variety of practitioner researcher” (p. 953). Feldman et al. (2004) offer three identifying criteria for self-study research: (1) the importance of the self of the researcher; (2) the experience of the researcher as resource for the research; and (3) a critical stance by the researcher towards their role. This study is aligned to the aforementioned criteria.

In the following section, I discuss a personal reflexive process as a means to inquire into the influence that my ontological and epistemological assumptions had on my teaching and learning of IT programming and on this research study. I also outline how I defined the self when inquiring into my practice as researcher and reflective practitioner (Schon, 1983), both in relation to the other participants and in relation to the context in which this study took place.

### **3.2.4 Personal reflexivity and prior experience**

The research I conducted for my Masters dissertation (Govender & Govender, 2014) was a research study in collaborative pair programming. This experience predisposed me to broaden my research and delve deeper into the teaching and learning of IT programming at a ToT. Furthermore, I received my training as a computer studies teacher at the Transvaal College of Education. As such, for four years I was immersed in a teacher training programme based on a procedural programming paradigm. This had and still does have a significant influence on my view of lecturing and teaching. Prior to my enrolment as a student teacher, as a secondary school pupil I had the unique opportunity to study computer studies at the senior secondary phase. Hence, this research is merely an extension and continuation of my passion for IT and computer programming. Reinharz and Davidman (1992) viewed the aforementioned statement as a functional aspect and as an expression of the researcher’s personal interests. Steier (1991) states that the researcher is part of the process of reflexive and self-reflexive conversations, including the choice of topic and methodology. Self-reflection and introspection form an integral component in Chapter Six with respect to the section on teaching styles.

This ever-increasing understanding of continually reflecting on my teaching styles, learning of IT programming, and IT educational influences as an IT lecturer to first-year ToT students has led me to ultimately mobilise educational debate in the teaching and learning of IT

programming to introductory first-year ToT students, and hence this research study. One of the social goals is not an increased understanding or resolution of a practical problem, but a “raised awareness in people of their own abilities and resources to mobilise for social action” (Bhana, 1999, p. 235).

This process of reflecting and reflexivity is valued in qualitative research, as indicated by Sandelowski and Barroso (2002, p. 216):

Reflexivity is a hallmark of excellent qualitative research and it entails the ability and willingness of researchers to acknowledge and take account of the many ways they themselves influence research findings and thus what comes to be accepted as knowledge. Reflexivity implies the ability to reflect inward toward oneself as an inquirer; outward to the cultural, historical, linguistic, political, and other forces that shape everything about inquiry; and, in between researcher and participant to the social interaction they share.

This research is a qualitative study and I had to acknowledge the participants’ data and the observational data, even when faced with data that was in contradiction to my beliefs as an academic. I had to reflect inwards taking into cognizance the cultural, historical and linguistic factors of the data sample; however, this was beyond the scope of the research.

From the reflective perspective of a ToT comes the notion that universities have an obligation to serve their local communities and that this calls for a mutually beneficial and reciprocal relationship. DUT states in its strategic plan (2015-2019): “University cannot be distanced from the community and context within which it is situated if it wishes to be sustainable”.

Universities have an obligation to serve their local research needs and this calls for continued support to staff and students. Brulin (2001) advocates similarly that universities serve their communities and encourage academics to form networks that will enable resources to be turned into energising assets. These networks could act as a form of social glue that “facilitates access to important resources and it turns such resources into energizing assets” (p. 441). This study is situated in the wider ToT community and in my view the study together with the recommendations and scope for further study are “energising assets” that will ensure its

sustainability in the future, and network not just between ToTs in KZN, but nationally amongst universities of higher learning.

The above section concludes the discussion regarding self-study, practitioner research and personal reflexivity. The theories that frame this study are threshold concepts and grounded theory and are discussed in the ensuing sections.

### 3.2.5 Threshold concepts

Meyer and Land (2005) define threshold concepts “as a way of describing particular concepts that might be used to organise the educational process”. Boustedt et al. (2007) claim that in CS education there is “also a developing context for threshold concepts” and during the academics’ and learners’ teaching and learning experience they will have encountered threshold concepts in the subject. The researcher was also keen to determine to what extent these threshold concepts may have influenced the sample population’s teaching and learning. The purpose of this section is therefore to discuss threshold concepts in CS and reflect on their usefulness for this study.

J. Goebel, S. Maistry

International Review of Economics Education 30 (2019) 100145

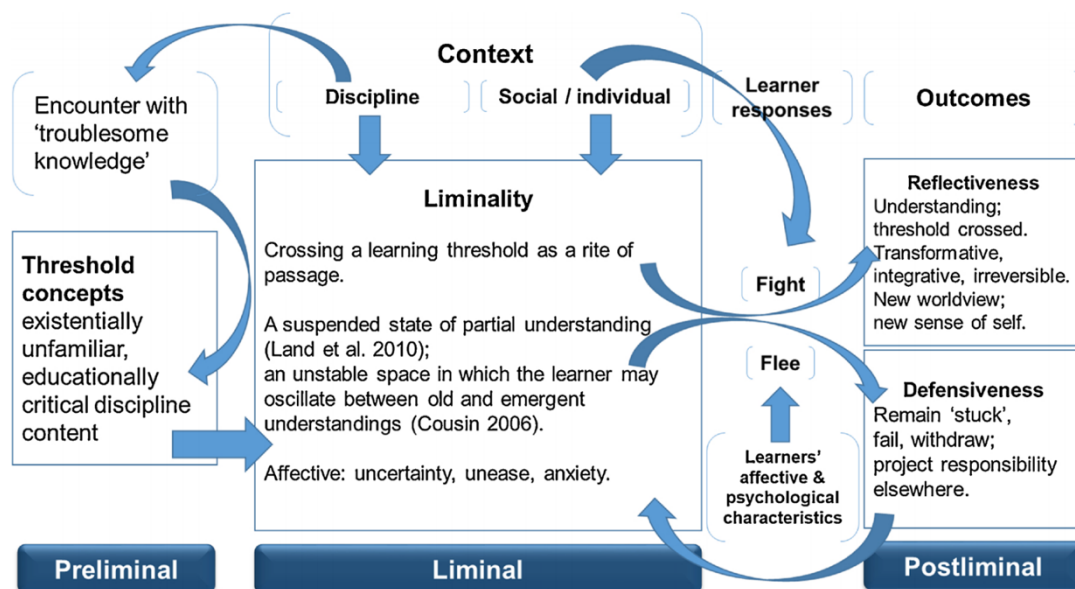


Fig. 1. A threshold concepts view of learning (based on Schwartzman, 2010 and Land et al., 2010).

Figure 3-1 A threshold concepts view of learning (Goebel & Maistry, 2019)

Rountree and Rountree (2009) view threshold concepts in two parts: firstly, as a model or framework, and secondly as “instance examples”. Threshold concepts provide academics with

a model to develop their teaching and support student learning. The conceptual framework is intended to re-situate teaching and learning within the context of its own discipline, in contrast to the role which learning outcomes have developed as a management tool to audit and monitor “success” (Hussey & Smith, 2003). In the traditional academic environment, an academic would state “by the end of the lesson the student should be able to...”; however, in the threshold concepts environment learners go through a “transformation”, after which they gradually acquire the identity of the community of practice; they begin to “think more like a computer scientist” (Rountree & Rountree, 2009). During this transformation, certain parts of the curriculum are pivotal: these represent the “portals” that learners must traverse in order to succeed. To be considered a member of the community of practice, mastery of these concepts is required, and the process of mastery is seen as a sort of rite of passage.

Threshold concepts may be viewed as “instance examples” and may be recognised by having (probably) all of the following five features: transformative; integrative; irreversible; bounded; and troublesome (Meyer & Land, 2005).

**Transformative:**

- They change the way a student looks at things in the discipline;
- They create in the learner a new way of viewing and describing the subject and may alter the learner’s perception of themselves and the world.

**Integrative:**

- They tie together concepts in ways that were previously unknown to students; learners make new connections.

**Irreversible:**

- They are difficult for the student to understand;
- They are irreversible since the fundamental qualitative change that occurs is unlikely to be unlearned.

Troublesome for students:

- Meyer & Land (2003) state that “these concepts are problematic for learners because they demand an integration of ideas...and this requires...a transformation of a student’s understanding”.
- Perkins (2006) has termed the concept of “the underlying game” as an activity of “a student that may grasp concepts but the barrier to their learning appears to lie at a deeper level of understanding”.

**Bounded:**

- They indicate the limits of a conceptual area or the discipline itself;
- This “helps define the boundaries of a subject area because it clarifies the scope of the subject community” (Land et al., 2005).

The programming approach of procedural programming and object-oriented programming complements the above threshold concepts. The programming concepts (selection, iteration, classes, objects, encapsulation and inheritance) are transformative, integrative, irreversible, and troublesome for students, and are boundary markers for them. Furthermore, the attributes of collaborative team teaching within a department or even across institutions and, in particular, the concept of “sharing teaching and programming knowledge” acquired from the sample population are examples of the “potentially troublesome for students (academics)” category. Academics were used to teaching and being assessed on their own, whilst collaborative teaching and the subsequent sharing of resources/knowledge proved troublesome to academics. This would be further elaborated upon in Chapter Five.

Land et al. (2005) included threshold concepts for course design and evaluation, with nine considerations based upon those that they felt were important in the design and subsequent evaluation of curricula in higher education. These nine considerations are discussed in greater detail below.

**3.2.5.1 Nine considerations for threshold concepts**

The nine considerations for threshold concepts are: jewels in the curriculum; the importance of engagement; listening for understanding; reconstitution of self; tolerating uncertainty; recursiveness and excursiveness; pre-liminal variation; unintended consequences of good pedagogy; and the underlying game or threshold conception. Each of the aforementioned

“considerations for threshold concepts” are summarised with specific reference to the data, however, and in-depth analysis of the nine considerations is expressed in the data analysis of Chapter Five.

- **Jewels in the curriculum**

By identifying threshold concepts as “jewels in the curriculum”, Land et al. (2005) give prominence to a crucial point which an academic must identify and serve to reinforce before proceeding onto a new section in the curriculum. With respect to teaching programming, crucial points that emerged from the overall data analysis are:

1. Teaching;
2. Learning and impact;
3. Challenges; and
4. Recommendations – teaching programming

Each set of crucial points may be further defined by subthemes of categories, for instance:

If one considers teaching as a crucial point; the following, then emerge as sub themes:

1. Concepts and topics;
2. Pair programming;
3. Programming lessons;
4. Assessments;
5. Support and engagement; and
6. Teaching support

An academic would be responsible for defining *what* concepts and topics are taught. This may be further explored and defined, for instance, by the syllabus, choice of programming language and programming paradigm taught, assessment criteria, setting up of pair programming, the number and type of assessments, and so forth. The data and research indicate that in defining a crucial point, this may be as in-depth as an objective of a lesson in programming, or as broad as a course outline for a programming module. The underlying characteristic is the student’s ability to master a particular “jewel” before proceeding to acquiring a new “jewel”.



- **The importance of engagement**

An academic should consider the specific forms of engagement which would be most appropriate to bring about particular transformative understandings at various points in the curriculum. Lather (1998) refers to these as “where the effort is to...provoke something else into happening – something other than the return of the same”.

Land et al. (2005) couple this point of engagement and a teacher’s need to provide a supportive liminal or transitional environment.

The programming concepts in the teaching and learning of CS reflect the importance of engagement between introductory programming students, between the student and the curriculum, and also between the academic and the curriculum. These are important engagements and surfaced in Chapter Two during the discussion of the literature on teaching methods associated with CS programming. The concepts of engagement will also be expanded upon in Chapter Five of the data analysis, under the sub-theme of teaching. Ultimately, the final product of engagement must be an improved understanding of and a transformation in a student’s understanding of difficult programming tasks and/or concepts.

- **Listening for understanding**

Teaching for understanding of threshold concepts needs to be preceded by listening for understanding. Ellsworth (1997) states that an academic must “cultivate a third ear that listens not for what a student knows but for the terms that shape a student’s knowledge”. With respect to teaching programming it was noted that an academic cannot adopt a passive role in a programming scenario but rather an eavesdropping role to intervene where necessary amongst students. Listening for understanding also forms part of the moderation process of assessments. An academic must ensure that he becomes acutely aware of common misunderstanding in programming assessments and provides necessary remedial strategies to rectify programming misunderstandings as soon as they occur and not allow them to fester into an ultimate failure on the part of the student.

Land et al. (2005) refer to the term “pre-liminal variation: the ways in which students approach or come to terms with a threshold concept”. They state that one cannot “second-guess where students are coming from or what their uncertainties are”. If an academic has to adopt the “pre-

liminal variation term” in a teaching environment, then very simplistically they will have to adopt the programming stance of their student and to “see” or “hear” the obstacles and difficulties of programming students. It may also refer indirectly to that which is unspoken during a “listening for understanding” consideration, for instance, during the data collection phase, it became apparent that the participants did not embrace the twenty-first century techniques of using technology in their repertoire of teaching CS programming. This highlights the concept of “listening for the missing/silent considerations”.

The aforementioned discussion equates to the sub theme from the data analysis which refers directly to a student’s background in computer programming. This is further explored in Chapter Five.

- **Reconstitution of self**

This term refers to either the student or the academic repositioning themselves in relation to their subject matter. Curriculum designers may want to redesign troublesome concepts or reposition the knowledge within a curriculum. Meyer and Land (2003) state that knowledge may be troublesome because it has become ritualised or inert, because it is conceptually difficult or alien, because it is tacit and perhaps requires awareness of an underlying game imperceptible to the student, or because of the discourse that has to be acquired for the concept to become meaningful.

Almost all of the participants indicated that the subject matter was a cause of concern and that to varying degrees of engagement they had to either learn or re-learn subject matter prior to student engagement in the lecture venue. It must be stated that CS is a dynamic subject and that participants did indicate that they experienced difficulty in keeping up with the latest technological trends and developments. Participants had to basically constantly upskill their qualifications and/or knowledge to keep abreast of the ever-changing technological advancements in CS. This is in in direct relation to the concept of reconstitution of self.

Perkins, (2006) describe reconstitution of self as follows:

as students acquire threshold concepts and extend their use of language in relation to these concepts, there occurs also a shift in the learner’s subjectivity, a repositioning of

the self. What is being emphasized here is the inter-relatedness of the learner's identity with thinking and language.

CS programming students have a unique language that defines their subject. The very spelling of the word program differs from the conventional spelling of programme. Similar examples and/or concepts are the words keyboard, windows, icon loop, recursion, iteration, and so forth.

Alston et al. (2015) identify the following threshold concepts in basic programming principles, namely: interrelationship of syntax; algorithms; and programming logic. Similarly, Miller et al. (2015) identify the passing of parameters in basic programming as a threshold concept that students need to master before proceeding to more advanced programming concepts.

In relation to this study during the data collection phase one participant remarked: "They don't start off as programmers, but they eventually get there".

This shift in the learners positioning of themselves firstly with the programming curriculum, and eventually with the subject as a whole, is sometimes seen as too sluggish and time-consuming by academics; however, it is the basis of education that each student is unique and that they would and should ultimately acquire the relevant threshold concepts and extend their knowledge and understanding of a programming language.

- **Tolerating uncertainty**

Land et al. (2005) discuss this point with reference to data collected from their participants and summarised their findings as "learners tend to discover that which is not clear initially often becomes clear over time". They quote a media student who almost abandoned her studies but continued and eventually did cope with the threshold concepts. Efklides (2006) emphasises the indispensable role of metacognition in the learning process "both directly by activating control processes and indirectly by influencing the self-regulation process that determines whether the student will get engaged in threshold concepts or not".

- **Recursiveness and excursiveness**

Land et al. (2005) argue for the notion of learning as excursive, a journey or excursion which will have intended direction and outcome but will also acknowledge deviations and unexpected

outcomes within the excursion; there would be digression and revisiting (recursion). Troublesome knowledge often requires revisiting and the “outcome of learning is not just the student would be able to but rather that the learner will have been transformed by the journey into one who thinks differently”.

- **Pre-liminal variation**

This consideration in the study refers to students negotiating the liminal place of understanding that is a state of confusion and anxiety with no apparent progress. Students present a partial, limited or superficial understanding of the concept to be learned, which Land et al. (2005) characterise as a form of “mimicry”. It is recognised that there would be a range of pre-liminal understandings that a cohort of students brings to the learning interactions. These are the concepts they carry into the liminal place where, in the case of teaching programming, there is partial understanding of concepts. The aim of this study was to investigate the teaching of programming at a ToT. This partial understanding of concepts was later extracted from the interviews and extrapolated with qualitative data analysis techniques and categorised as a challenge in teaching programming. This would be dealt with in greater detail in Chapter Five.

Suffice to state the partial understanding of concepts identified as challenges to teaching programming proved to be a mapping to threshold concepts in this study of teaching programming. This study further argues that the experiences of students and academics must focus on experiences of the challenges of teaching and learning of programming to meet the objectives not as an aftermath but as a prerequisite to transcending confusion, anxiety and misunderstanding of difficult computer programming concepts and to ultimately attain understanding. This is in contrast to Schwartzman (2010), who argues that descriptions of threshold characteristics and of liminality have tended to focus on the aftermath rather than the experience of students’ learning of challenging concepts.

- **Unintended consequences of generic “good pedagogy”**

Academics are at pains to simplify concepts they perceive to be difficult for learners to understand. Sometimes their efforts to simplify exceed the scope of the concept to be simplified and the meaning is lost, an experience supported by Shanahan and Meyer (2006). With respect to teaching programming it was noted by the researcher that in an effort by academics to complete the syllabus, they adopted a superficial and “quick fix” approach of providing

students with complete working programming tasks during a lecture or tutorial and then adapting and making slight alterations to a homework assignment. Although well intended, this proved dysfunctional. “What was traditionally thought of as good pedagogy may on occasion break down or prove to be dysfunctional in acquiring threshold concepts” (Shanahan & Meyer, 2006). Students missed the feeling of excitement, enjoyment and accomplishment that can be experienced when running a computer program which they have been struggling with for several days, weeks and even months and to see the fruition of a successful compilation.

- **The underlying game or threshold conception**

Perkins et al. (2006) define an “underlying game” or “threshold conception” in instances where students can grasp concepts but the barrier to learning appears to lie at a deeper level of understanding. Meyer and Land (2005) quote an example in CS programming, and this highlights the relevance for this study: students may grasp the concepts of object-oriented programming but may not appreciate the threshold conception of the underlying game of the interaction of all these elements in a process of ever-increasing complexity. Yeomans et al. (2019) express a similar dilemma and pose the following question in their research: “Are threshold concepts a suitable, and pedagogically useful, concept in understanding learning of (object-oriented) programming?”

They conclude that they faced difficulty in clearly identifying a set of threshold concepts and had identified concepts that were too broad and not useful for the continued improvement of pedagogical approaches. They did, however, argue that it is a good basis for identifying important concepts to be the focus of a curriculum.

Similarly, with teaching programming in this study, it was noted in the data collection that academics alike may grasp the concepts of the basic programming structures, namely assignment, decision-making and looping and this becomes the focus of the programming curriculum. It may not necessarily be defined as a threshold concept; however, it may be defined as an area of intense focus.

Section 3.2.6 traces grounded theory as initially proposed by Glasser and Strauss (1967), which was further developed by Charmaz (2003), who describes the theory as constructive grounded theory.

### **3.2.6 Grounded theory**

The methodology that formed this study's data analysis drew on Glaser's writing on grounded theory (Glaser, 2001; Glaser & Strauss, 2009). Glaser & Strauss (2009) state that "Grounded theory offers a rigorous, orderly guide for theory development and is designed to allow the researcher to be free of the structure of more forced methodologies. Its real strength lies in its open-ended approach to discovery". The goal of grounded theory in the research design of this study was to guide data collection from participants who were proponents of teaching programming to first-year IT students.

Using grounded theory, a researcher commences with data and uses them to develop theory that provides relevant interpretations, applications, predictions and explanations (Glaser & Strauss, 1967). Grounded theory specifies an analytical strategy to collect rich data from multiple sources. It involves a process of collecting data to fill conceptual gaps, applying constant comparative analysis, refining concepts, defining the properties of the categories and identifying their relevant contexts (Boyчук, Morgan, 2004). Grounded theory is an approach where theory is generated inductively from analysis of the data as concepts are formulated into a logical, systematic and explanatory scheme (Strauss & Corbin, 1998).

Qualitative research is one of the frequently used forms of research design in (Creswell, 2014). Charmaz (2006), Corbin and Strauss (2007) and Strauss and Corbin (1998) identify the procedures of grounded theory. This research design is a design of inquiry from sociology in which the researcher derives a general, abstract theory of a process, action, or interaction grounded on the views of participants. This process involves using multiple stages of data collection and the refinement and interrelationship of categories of information (Charmaz, 2006; Corbin & Strauss, 2007).

The grounded theory method is predicated on collecting, examining, and checking data. This method has focused on data analysis. (Charmaz, 2014, 2015). The notions of grounded theory played a significant role in this study because the data collected from the participants from a qualitative perspective answered the question: What should I do with the data? Grounded theory provided a strategy for collecting the exhaustive qualitative data and expedited the analysis of this research. The distinguishing characteristics of grounded theory methods (Glaser and Strauss, 1967; Strauss and Corbin, 1990) include:

1. Simultaneous involvement in data collection and analysis phases of research;
2. Creation of analytic codes and categories developed from data, not from preconceived hypotheses;
3. The development of middle-range theories to explain behaviour and processes;
4. memo-making, that is, writing analytic notes to explicate and fill out categories, the crucial intermediate step between coding data and writing first drafts of papers;
5. Theoretical sampling, that is, sampling for theory construction, not for representativeness of a given population, to check and refine the analyst's emerging conceptual categories; and
6. Delay of the literature review.

An important point of note with respect to the data collection phase is the concept of “silence”. (Charmaz & Belgrave, 2018). Data can bring silences into our coding and narratives. Qualitative inquiry has long valorised individuals’ stories and overlooked silences (Charmaz, 1990). This was evident from the participants who were absolutely silent regarding certain concepts in their teaching strategies; for instance, the concepts of robotics, artificial intelligence and machine learning. Silence on the part of the participants regarding robotics does not mean that it must not be studied.

I argue in favour of threshold concepts (Meyer & Land, 2005) and constructive grounded theory Charmaz (2003) for better understanding the teaching and learning of computer programming to ToT students and academics.

The technique that is central to grounded theory is coding. Coding refers to the analytical processes through which data are fractured, conceptualised and integrated to form theory (Strauss & Corbin, 1998). Charmaz (2003) and Leedy & Ormrod (2005) state that “Data analysis occurs through various coding procedures: open coding, axial coding, selective coding and development of a theory”.

- **Open coding**

This is the first analytical coding process. Its purpose is to mark text or other informative data and to associate codes with the marked segments of text. Data are deconstructed into concepts and marked line-by-line and coded. This process of grouping concepts into higher levels of abstraction is called categorising and this forms the basis of grounded theory construction

(Henning, 2004). The selection of groups for comparison highlights the various similarities and differences, which is vital for defining the different categories (Glaser & Strauss, 1967).

- **Axial coding**

This coding is defined as “the process of relating categories to their subcategories” (Strauss & Corbin, 1998; Glaser & Strauss, 2009). Its purpose is to refine the information about each category and its subcategories, determining more about a category in terms of its conditions, context, strategies and consequences (Leedy & Ormrod, 2005).

- **Selective coding**

This coding is defined as “the process whereby a main category is selected to which other categories are related” (Henning, 2004). It is a process where categories are organised around a central explanatory concept where major categories are related (Strauss & Corbin, 1998).

These coding procedures were used to guide the analytical process applied in Chapter Five.

### **3.2.6.1 Application of grounded theory methods (GTM) to this study**

Grounded theory was chosen for the data analysis in this study to facilitate the researcher’s need to answer the research questions based on the experience of learners and academics. In conducting this study, the researcher was searching for influences that may have enhanced or hindered the teaching and learning experience of academics and learners. Much of this data analysis will form part of theory that future academics of computer programming can take note of. Grounded theory strategies are just that – strategies for creating and interrogating our data, not routes to knowing an objective external reality (Charmaz, 2014). The strategies of grounded theory were used to create and interrogate the study’s data.

Data was collected from a tertiary institution. Learners and academics were exposed to programming concepts. The participants teach programming to first-year introductory programming students and this coincides with Strauss & Corbin (1998), who state that, “A key idea is that much of the theory development does not ‘come off the shelf’ but rather is generated or ‘grounded’ in data from participants who have experienced the process”.



According to Birks et al. (2013), a researcher can claim to have conducted grounded theory methodology if they have met the following criteria:

### 1. Theory Development

The objective of the study was to provide a rich description of a phenomenon based on a systematic exploration of the accounts of the phenomenon (through interviews, observations, archival materials or quantitative data sources) (Strauss & Corbin, 1990, 1998).

This study provided a rich description of the phenomenon of teaching and learning of computer programming to first-year IT students at a ToT through interviews with academics and observations of lectures and archival materials.

### 2. Constant comparison

Constant comparison was used to analyse data from different standpoints. In GTM, memos are critical as internal sense-making techniques through which constant comparison is achieved. Memos can take the form of diagrams, text narratives, propositions, mind maps and other techniques which are suitable to both the idea being documented and to the cognitive preferences of the researcher as sense-maker (Charmaz, 2006, 2014).

Chapter five of this study deals specifically with the grounded theory methods of constant comparison and memo-ing. There are text narratives of the data collected, including diagrams (word clouds, tree maps and cluster diagrams).

### 3. Iterative coding

Constant comparison led to theory developed through several iterations of data coding. In this process, concepts were defined, their dimensions developed and abstracted out. The concepts were then interrelated to each other and, potentially, to the extant literature.

Chapter Five provides an exhaustive analysis of the several iterations of the data coding. Chapter Six led to the theory development. Themes/sub-themes and categories have been constantly interrelated throughout the chapters including the Chapter Two literature review.

#### 4. Theoretical sampling

The data were collected based on theoretical sampling, with collection ceasing when the data reached theoretical saturation. The data were collected from academics at a ToT and data collection ceased when participants were repeating their responses to interview questions and observations ceased after repeated observations of identical teaching and learning phenomena.

#### 5. Management of preconceptions

GTM requires that we avoid using specific theories pertaining to the phenomenon under study as the starting point for data collection and analysis.

No specific theories were used as a starting point; however, as per Glaser (1978, 1998) and Charmaz (2006) an a priori theory of teaching and learning was compared against evidence of data collected and analysed, as it related to the literature reviewed in Chapter Two.

Furthermore, as a disclaimer the researcher is inextricably bound by his convictions as an academic and student of CS programming; hence my influence is inextricably interwoven in the chapters. According to Burrell and Morgan (2017), a theorist should be fully aware of the assumptions upon which his own perspective is based. My assumptions did influence recommendations of the study including that which was not said; as a point of note, almost *all* participants did not place much emphasis on technological advancements in teaching and learning, especially concepts of machine learning, deep learning and AI. This is highlighted in chapter six.

#### 6. Inextricable link between data collection and analysis

The data collection and analysis activities were intrinsically related. This would be discussed in the ensuing section.

The subsequent applied criteria of GTM, as noted in the previous section, are revisited in the section below. The steps are: open coding, axial coding, selective coding interpretation and development of a theory.

- Open coding

Data was deconstructed into parts. During the process of open coding the following questions were asked:

What did the participant say when teaching programming to first-year ToT students?

What actions did the participant take when confronted by difficulties/challenges in programming?

What did the participant recommend with regard to improving the understanding of teaching programming concepts?

The “code ability” to teach programming to first-year ToT students was later categorised and the following key themes emerged from the data analysis:

1. Teaching;
2. Learning and impact;
3. Challenges;
4. Recommendations – teaching programming

- Axial coding

Different categories were combined in new ways to make connections. The researcher focused on categories that may be clustered together or divided into sub-categories. Several closely related concepts can be organised into a major topic of interest. Furthermore, this may suggest dropping or adding a theme or examining other themes in more depth (Newman, 2012). For example, the categories that emerged from the “challenges” theme, are examined in greater detail under the following sub-themes:

- Background and diversity;
- Student ability and readiness; and
- Lecture and classroom.

This is explained in further detail in Chapter Five.

The academic can complete the syllabus faster and learners work faster because programming tasks are shared and completed, theoretically, in half the time.

- Selective coding

During selective coding a main category is selected to which other categories are related. Selective coding involves scanning data and previous codes and selectively looking for cases which illustrate themes (Henning & Van der Westhuizen, 2004; Newman, 2012).

- The selection of groups for comparison makes similarities and differences distinct. For example, a coded main category named “challenges” is a theme in Chapter Five and is comprised of several different codes that related to aspects within this theme. The theme “challenges” was coded, and the following sub-themes of different yet related codes emerged, namely:
  - Background and diversity;
  - Student ability and readiness; and
  - Lecture and classroom.

The afore-mentioned theme, together with the sub-themes, is discussed in Chapter 5.

- Interpretation and development of a theory

Qualitative interpretation with a final thematic pattern was constructed from the findings (Henning & Van der Westhuizen, 2004). During the process of interpretation, different methods and actions all came together to motivate and defend the interpretations, and to develop an emerging theory on the teaching and learning of computer programming. Chapter Six gives details on such an emerging theory relating to participants’ teaching experiences.

### **3.3 Conclusion**

Threshold concepts, grounded theory, self-study and practitioner research formed the theoretical basis for this study. As reported in the above sections and further expanded upon in Chapter Five, the themes extracted from the data analysis complemented the threshold concepts and grounded theory themes and categories. The data analysis is discussed further in Chapter Five and Chapter Six. The next chapter details the methodology employed in this study.

## Chapter Four: Research methodology

### 4.1 Introduction

This chapter details the research process which led to the collection of the data that were used to provide answers to the research questions. This chapter provides a discussion of the research context, research design, rationale behind the research methodologies and data collection instruments, data collection process, ethical considerations, sampling strategies and techniques that were used in the analysis and interpretation of data.

In order to gauge academics' perceptions of learning and teaching programming, interviews were conducted, and lessons observed. The participants were all IT academics who had lectured to first-year IT students during the course of their academic career. Due to workload allocations in the IT department it was not possible to source a participant who solely lectured to first-year IT students. Participants were, however, requested to confine their discussions and responses to their experiences as first-year academics. Participants indicated their willingness to be observed and interviewed. This was partially due to the purposive sample selected and the relationship between the researcher and the participants. The interviews and observations sought to determine participant perceptions of their own strategies when teaching IT at a ToT. The data collection sought responses in the following areas:

- Teaching strategies used by ToT IT academics when teaching IT programming?
- Perceptions of ToT academics regarding factors impeding or enhancing student IT programming?
- Including what the participants did *not* mention when teaching, for instance all participants did not include the use of YouTube videos or the production of videos from their lectures.

A discussion about participants and site might include four aspects identified by Miles and Huberman (1994):

(a) The setting (i.e., where the research will take place),

The research took place at ToT in the computer laboratory and lecture venues.

(b) The actors (i.e., participants observed or interviewed),

The actors were IT academics and students

(c) The events (i.e., what the actors were observed or interviewed doing); and

The focus of this study was the teaching and learning of IT programming experiences and events of the actors, and the perceptions and meaning attached to those experiences as expressed by the actors.

(d) The process (i.e., the evolving nature of events undertaken by the actors within the setting).

The actors would be teaching IT programming.

## **4.2 Chapter overview**

The research methodology and research context of this chapter examine and emphasise:

4.2.1 Research context and sampling;

4.2.2 Research questions and design;

4.2.3 Timeframes for data collection;

4.2.4 Data collection methods;

4.2.4.1 Observation;

4.2.4.2 Interviewing;

4.2.5 Design of the interview and observation;

4.2.6 Interview questions/observation criteria;

4.2.7 Data analysis and value of the question set;

4.2.8 Validity/reliability and trustworthiness;

An explanation of the research context and sampling is provided in section 4.2.1.

### **4.2.1 Research context and sampling**

The research was conducted at an institution of higher learning in KwaZulu-Natal. The participants were academics and students studying IT programming.

I used purposive sampling and backyard research to conduct this study. Using the purposive sampling technique, a researcher decides what needs to be known and sets out to find people who can and are willing to provide the information by virtue of knowledge or experience (Bernard, 2017). It is a non-random technique which does not need underlying theories or a set

number of participants. It is typically used in qualitative research to identify and select the information-rich cases for the most proper utilisation of available resources (Patton, 2002). This involves identification and selection of individuals or groups of individuals which are proficient and well-informed with a phenomenon of interest (Creswell & Clark, 2011).

In purposive sampling the importance of availability and willingness to participate, and the ability to communicate experiences and opinions in an articulate, expressive, and reflective manner are all important contributing factors (Stuttgart Spradley, 1979). Fortunately, in this study all the participants were willing, and the lecturing staff were expressive in discussing their practices in teaching and learning of IT programming because they all expressed a desire to contribute to increased throughput rates, increased student understanding, and decreased dropout rates.

This research sample needed to comprise naturally formed groups of first-year students studying IT programming and lecturers teaching IT programming to first-year students. The participants were willing and were knowledgeable with respect to teaching programming. All participants were seasoned experienced lecturers in IT programming. The rationale behind choosing the particular campus was the fact that the researcher would capture the participants' perceptions and experiences using a particular programming approach having a similar teaching curriculum across the spectrum of first-year students having the same facilities and equipment. This was crucial for this study since it minimised extraneous factors that could have affected the outcome of the research.

Backyard research (Glesne & Peshkin, 1992) involves studying researchers own organisation, or friends, or immediate work setting. I studied and researched my own ToT. Some of the participants were my colleagues and some were my friends; similarly, some of the students were from the immediate courses that I was lecturing. Section 4.9 discusses the ethical issues regarding this research with respect to sampling techniques used and methods employed to protect participants.

The class size at the ToT averaged sixty students per class. This was due to space constraints and computer laboratories that could accommodate only a maximum of sixty computers per venue. Each learner was assigned their individual personal computer, with Local Area Network

and Internet access. The ToT also had adequate infrastructure, namely data projectors, laptops, smart board technology and campus wide Wi-Fi.

All academics were adequately qualified and displayed a willingness to embrace the study. They expressed a desire to:

1. Increase their personal awareness and exposure of teaching IT programming at ToTs; and
2. Expand their teaching and learning strategies of IT programming, in an effort to identify best practices of teaching and learning introductory programming to ToT students.

The sample population is elaborated upon in greater detail in Chapter Five. Table 5-1 describes the biographical data of participants.

Participant number	Gender	Age category(years)	Highest Qualification
1	Male	30-35	Not stated
2	Male	35-40	Honours
3	Male	35-40	Masters Computer Sc.
4	Female	30-35	BSc Honours
5	Female	30-35	Masters IT
6	Female	55-60	PhD
7	Male	40-45	PhD
8	Male	40-45	Masters Commerce
9	Male	50-55	PhD
10	Male	55-60	PhD

*Table 5-1 Biographical data of sample population*

#### **4.2.2 Research questions and design**

This study used a qualitative design approach to address the research questions. Duffy (1987) summarises the relationship between quantitative and qualitative research: “quantitative research is used to evaluate objective data consisting of numbers, while qualitative research deals with subjective data that are produced by the minds of participants”. Qualitative research methods involve collecting textual or verbal data and observation of people followed by careful



description and analysis. The participants in this study verbalised their responses and their lessons were observed by the researcher. Furthermore, their responses were subjective in nature. The data collected from the study, that is the interviews and observations, were analysed using qualitative research methods.

Having identified the research topic for this study, it was appropriate to develop research questions which would provide the answers and possibly solutions needed to address the research at hand. The research questions provided a focused means of investigating the research area (Bauer & Gaskell, 2000; Cohen, Manion & Morrison, 2000).

In this study I was interested in the teaching strategies of IT academics at a ToT, and to achieve the research aims specific research questions were formulated. The research questions were:

- 1) What are the teaching strategies used by ToT IT academics when teaching IT programming?
- 2) What are the perceptions of students regarding the teaching methodologies used for ToT IT programming?
- 3) What are the perceptions of ToT academics regarding factors impeding student IT programming?
- 4) How do the teaching strategies employed by IT academics at the ToT influence the learning of IT programming?

The concept of the research questions was approached from the perspectives of the academics teaching computer programming to first-year students.

The first research question was aimed at finding out the participants' strategies of teaching programming to first-year introductory programming students. The second research question dealt with the students' perceptions and this was dealt with during the observation phase of the data collection. It also offered the researcher an opportunity to fine tune and add to the interview questions with the academics. Research questions three and four were aimed at participants' perceptions of factors and strategies that they felt afforded students the opportunity to either succeed or impede on the teaching and learning of IT programming.

The researcher chose to use qualitative methods for this study. Qualitative research involves collecting textual or verbal data (data which cannot be counted), and involves observation of people, followed by careful description and analysis (Boeree, 2006). Qualitative research methodology is an inductive and exploratory tool, and this was adhered to during the data collection phase. It is therefore an in-depth analysis of a problem in order to understand human behaviour (Hatch, 2002). Qualitative research is concerned with exploring social and human problems in a natural setting, with the intention of understanding what people feel and the experiences that have caused them to have these feelings. Since this study required in-depth knowledge of learners' experiences, perceptions and attitudes when using a teaching strategy at an academic institution, the researcher chose to conduct this study within the qualitative methods framework.

Quantitative research, by contrast, is more highly defined and closely related to research in the physical sciences (Mouton & Marais, 1988). Quantitative research involves collecting numerical data; the data from this type of research can be represented statistically and graphically. This type of research facilitates the analysis and comparison of data. This study did not employ quantitative research techniques.

#### **4.2.3 Timeframes for data collection**

The following timeframe was decided upon. The interview process and associated observations were conducted during the first part of the first semester (February 2018 - June 2018). The observation process continued for the duration of the first semester, for as long as the programming was being taught in the classroom under the participant's supervision. Due to logistical factors, the interview process was completed in the second semester (July/August 2018).

#### **4.2.4 Data collection methods**

In order to obtain data that would further the aims and strengthen this study, it was decided that it was necessary to use more than one data collection method. The study used interviews and observations. The researcher observed the academics and learners in their natural environments (computer laboratories and lecture venues). Academics were interviewed. The interview enabled the researcher to gauge academics' perceptions and experiences of teaching computer programming to first-year introductory IT students. Asking open-ended questions enabled the researcher to gain insight into the personal experiences of the participants, and also assisted in

obtaining information regarding how participants benefitted from the use of various teaching strategies. Observation allowed the researcher to observe the participants' interaction with teaching strategies and programming concepts in the participants natural setting – the computer laboratory. Ultimately, the observations and interviews enabled the researcher to provide answers to the research questions and afford recommendations to IT academics. (refer to Chapter Six for a detailed narrative).

The following sections discuss the data collection methods in greater detail.

#### **4.2.4.1 Observation**

Observational research involves the researcher making observations. This method of collecting data involves the researcher going into a classroom, school or university and observing what is actually taking place there. The researcher writes down a description of what they see happening in the classroom (Cohen et al., 2000).

Participants taught programming lessons. These lessons were observed because the researcher wanted to gauge the learners' initial reaction to and interaction with a programming task. This was structured observation, with a focus on the programming activity. While the learners were engaged in programming tasks, the researcher informally observed them to monitor and observe their interaction with the programming task. The researcher's presence in the classroom as an observer was nothing out of the ordinary because of his dual role (lecturer and researcher).

This suited the data collection method because the researcher wanted to remain as “invisible” as possible in order to get an idea of the learners' experiences without interfering with their behaviour. The observation process also provided the researcher with realistic data that the learners may not have been able to provide through the questionnaire or interview, such as the learners' hands-on interaction with the programming compiler/user interface. See Appendix B for the observation schedule.

The advantage of observing the learners in the classroom was that data was collected when and where the activity was occurring, and the researcher could directly observe what the learners were doing. The disadvantage to using the observation process was that it was susceptible to observer bias – especially with the observer being simultaneously both researcher and lecturer

or when observing other academics in the classroom/lecture room/computer lab. To avoid researcher bias, the observation was structured, and the presence of the researcher was unobtrusive during the observation and interview process.

#### **4.2.4.2 Interviewing**

Interviewing is a technique that employs questioning as its principal method of data collection (Leedy & Ormrod, 2005; Mouton & Marais, 1988; Terre Blanche & Durrheim, 2002). As a data collection method the interview may vary from those that are completely unstructured to those that are completely standardised and structured (McMillan & Schumacher, 1993; Cohen et al., 2000; Johnson & Waterfield, 2004). Seidman (1998) points out that the basis of interviewing is the desire to understand other people's experiences and what they make of such experiences. He states:

[Interviewing] provides access to the context of people's behaviour and thereby provides a way for researchers to understand the meaning of that behaviour. A basic assumption in in-depth interviewing research is that the meaning people make of their experience affects the way they carry out that experience...Interviewing allows us to put behaviour in context and provides access to understanding their action (Seidman, 1998, p. 4).

The researcher chose to use a semi-structured and open-ended interviewing method. In a semi-structured interview there is an incomplete script. The researcher may have prepared some questions beforehand, but there is a need for improvisation. The interviewer is the researcher or is one of a team. (Myers & Newman, 2007)

Henning (2004) describes standardised interviews as a data production method in which the interviewer is to control the process so as to ensure that the interviewee does not wander off the topic, yet allowing the participant(s) to "freely" give subjective answers (that yield information that represent reality more or less as it is through the response of the interviewee) to the questions posed by the interviewer. Thus, the interview method employed in this study took the form of a standardised open-ended interview which used semi-structured questions (Terre Blanche & Durrheim, 2002). All the interviews were guided by a set of questions (refer to Appendices A & B) and were recorded using a digital voice-recorder and later transcribed (available electronically).

#### **4.2.5 Design of the interview and observation**

An interview/observation timeline was drawn up indicating dates and times during which academics would be interviewed/observed. Dates for the interviews/observations were arranged, scheduled and fixed from January 2018 to July 2018. All interviews were recorded and later transcribed. Academics were contacted via email and were told a day in advance when they would be interviewed. Copies of the interview questions were emailed to the academics prior to commencement of the interview, so that they had time to peruse them. Academics were told that they could jot down their thoughts and ideas if they wished to. They were not compelled to write down responses on their interview sheet. At the start of the interview the researcher welcomed and thanked the academic and then explained the research process. The researcher reiterated their rights as participants and outlined the interview process to them. Academics were informed that the data collected from the interview would be analysed and used in the write-up of this dissertation, while respecting the confidentiality of the individual interviewees.

#### **4.2.6 Interview questions/observation criteria**

Questions and observation criteria were formulated based the researchers' experience, observations and review of existing literature. This was aimed at eliciting the basic views of participants towards teaching computer programming, their students and their experiences as seasoned IT academics. The data collection methods were administered by the researcher. Owing to strict time constraints at the ToT, the data collection had to be quick and efficient while ensuring reliability and validity. A structured set of interview questions met the researchers' needs as far as this was concerned, since the questions were pre-set, and the responses fell within a prescribed range.

The data collection methods proved valuable and facilitated the collection and analysis of participants' perceptions and attitudes towards programming. The question set and observation criteria appear in Appendix A and Appendix B.

#### **4.2.7 Data analysis and value of the question set**

Data analysis involved organising, analysing and interpreting data (McMillan & Schumacher, 1993). The analysis of data encompasses the breaking up of complex data into manageable

themes, patterns, trends and relationships (Mouton & Marais, 1988). Analysing what the participants have said in an interview requires the researcher to relive the interview and to link the responses with the underlying theories, while looking for evidence to support or contradict them (Bauer & Gaskell, 2000).

The data from the interviews were transcribed, and once the transcripts were completed the researcher looked for themes and categories that were associated with the theoretical framework, guided by the research questions. This part of the analysis began with producing the transcript of the IT academics' interviews. The researcher preferred to do the transcription himself because the voices could be easily recognised and he could recall what was said if the recording was unclear, since the interviews were still fresh in his mind. It also gave the researcher a chance to relive the interview process by going through every word and expression in an effort to try and make sense of the data. Once the transcripts were done, the researcher looked for themes and categories which were associated with the theoretical framework and the research questions.

The responses from the data were analysed using qualitative data analysis techniques. Nvivo software would be used to organize, analyze and visualize data by providing tools for classifying, sorting and arranging data and hence enable the identification of themes and patterns. Data was expressed using tree nodes, and word clouds cluster analysis. (Refer to Chapter Five for a pictorial representation thereof). The aforementioned pictorial representations generated nodes which were captured on a Microsoft Excel spreadsheet in terms of nodes (refer to Figure 4-1). The nodes facilitated the statistical representation of data in terms of themes/categories and sub-themes/sub-categories, which are discussed in greater detail in Chapter Five.

	A	B	C	D	E	F	G	H
1	Name	Sources	References	Created On	Created By	Modified On	Modified By	
2	Teaching	0	0	2018/12/18 9:07 AM	S	2018/12/18 10:10 AM	S	
3	Teaching support	0	0	2018/12/18 9:12 AM	S	2018/12/18 9:12 AM	S	
27	Support and engagement	0	0	2018/12/18 10:25 AM	S	2018/12/18 10:25 AM	S	
58	Programming lessons	0	0	2018/12/18 10:21 AM	S	2018/12/18 10:21 AM	S	
138	Pair programming	0	0	2018/11/20 3:41 PM	S	2018/12/18 10:10 AM	S	
156	concepts and topics	0	0	2018/12/18 10:15 AM	S	2018/12/18 10:15 AM	S	
204	Assessments	0	0	2018/12/18 10:16 AM	S	2018/12/18 10:16 AM	S	
250	Recommendations - teaching programming	0	0	2018/11/20 3:54 PM	S	2018/12/18 9:13 AM	S	
274	Learning and impact	0	0	2018/12/18 9:09 AM	S	2018/12/18 9:11 AM	S	
427	Challenges	0	0	2018/11/20 2:23 PM	S	2018/11/20 2:23 PM	S	

Figure 4-1 Excel spreadsheet denoting nodes from the data set

The above data collection instruments are not without their disadvantages, in that they are weighed down by the time taken to design, pilot and refine them (Cohen et al., 2000). A question set is also limited in terms of the scope of the questions which can be asked and the range of responses that can be anticipated (Heather, Rollnick & Bell, 1993).

According to Cohen et al. (2000), a properly designed set of questions facilitates the process of analysis, and this can be made even easier when the researcher is involved in the design. The question set was useful in this context because it was not time-consuming to administer by the researcher. The workload of the academics coupled with the timetable at the ToT did not warrant a method that made further demands on academics' and learners' overburdened workloads. In order to ensure that an adequate number of academics would respond to constitute a valid study, all of the first-year IT academics from the ToT were invited to participate, and eventually the study was conducted with a sample of ten participants.

All responses were used to ensure that the data were not skewed; however, incomplete and invalid responses in the questions had to be excluded. Detailed analysis is provided in Chapter Five. There were more males in the sample population. It was decided that the data reflected the reality of the situation and that the status quo should remain. Gender was not an issue. The questions sought to measure participants' experiences with regard to computer programming. A recommendation for further study may include gender biases in a ToT.

#### 4.2.8 Validity/reliability/trustworthiness

The application of a multi-methods approach allowed for a comparison of data – referred to as triangulation (Krefting, 1991). According to Denzin and Lincoln (2008) triangulation is a

means of ensuring concurrent validity and prevents personal bias. Validity refers to “the appropriateness of the conclusions claimed from the analysis of the collected data” (McMillan & Schumacher, 1993; Terre Blanche & Durrheim, 2002). This has to do with whether the research methods, approaches and techniques used were appropriate for the study conducted. A major objective in research is to generate valid inferences about issues addressed by it (Buchanan & Bryman, 2009). To ensure valid inferences in this study, the researcher interviewed (using a voice-recorder) the participants with the intention of gaining insight into their understandings and experiences of teaching and learning computer programming.

Furthermore, to ensuring internal validity, the following strategies were used in this study:

1. Triangulation of data – data were collected from multiple sources and multiple participants and at multiple sites; it included both interviews and observations. Data were also collected from observation of lessons using the structured observation schedule. The collection of data from different sources added to the strength of the validity, reliability and trustworthiness of the study. The researcher interviewed academics and observed learners. The interviews were then transcribed. Data were collected from observation of lessons. The reason for this was that the researcher was able to acquire the same data using different research instruments and different forms, thus enabling the research to acquire triangulation, which enhanced the validity of the data collected (Klopper, 2008).
2. Member checking – the participant served as a check throughout the analysis process. An ongoing dialogue regarding my interpretations of the participant’s reality and meanings will ensure the truth value of the data;
3. Long term and repeated observations were conducted at the research site for the duration of the semester.
4. Participatory modes of research – the participant involved in most phases of this study, from the design of the project to checking interpretations and conclusions; and
5. Clarification of researcher bias – at the outset of this study, researcher bias was articulated in writing in Chapter One under the heading, “The Researcher’s Role”.

In ensuring external validity, I sought the provision of rich, thick detailed descriptions so that anyone interested in transferability will have a solid framework for comparison as discussed by Merriam (1995).



Three techniques were used in this study:

- 1) Triangulation or multiple methods of data collection and analysis were used, which strengthened reliability as well as internal validity (Merriam, 1995).
- 2) Data collection and analysis strategies would be reported in detail in order to provide a clear and accurate picture of the methods used in this study. All phases of this project were subjected to scrutiny by editors, a research supervisor, and a data analyst who are all experienced in qualitative research methods.
- 3) I provided a detailed account of the focus of the study, the researcher's role, the participant's position and basis for selection, and the context from which data was gathered (Goetz & LeCompte, 1984).

In providing a detailed account I also reported the findings as suggested by Lofland (1974) and Miles and Huberman (1984). Data are displayed in a descriptive narrative form rather than as a scientific report. This is a naturalistic study. Thick descriptions would be used to communicate a holistic picture of the experiences of the IT academic staff and the first-year IT students. The final project would be a construction of all the actor's experiences and the meanings they attached to them. This will allow readers to vicariously experience the challenges encountered by the actors and provide a lens through which readers can view the actor's world.

#### **4.3 Ethical issues of the study**

According to McMillan and Schumacher (1993), "ethical issues refer to all the precautions, steps and efforts that researchers carefully put into practice to protect the research participants while interacting with them for data production." Bell (2014) argues for the establishment of ethics committees which can ensure that no badly designed or harmful research is permitted. A credible research design involves the selection of participants, effective research strategies, and ensuring that all of the steps of the research adhere to research ethics.

As mentioned in Section 4.2.1, this study employed purposive sampling and backyard research techniques (Glesne & Peshkin, 1992).

Whilst it is noted that backyard research may:

- Compromise the researcher's ability to disclose information;
- Raise issues of an imbalance of power between the inquirers and the participants; and
- Jeopardize the roles of the researchers and participants

In order not to place the participants or the researcher at risk and ensure that data was not compromised, the following measures were implemented: I sought ethical clearance and gatekeeper permission from all relevant stakeholders and adhered strictly to the terms and conditions therein. In qualitative research, gatekeepers are used to assist the researcher in gaining access and developing trust with the community of study (Hatch, 2002). Ethical clearance was obtained during the planning and implementation of this research project; consideration was given to ethical issues relating to using learners as part of the data collection method. The researcher first had to apply for permission from the ToT (Appendix D). In the application the researcher outlined the type of research that was going to be done, the research methods and the data collection instruments that were to be used. The application also included how ethical issues concerning participants were to be addressed. Once ethical clearance was issued, the data collection process began.

The ToT research department granted the researcher permission to use the ToT as the research facility, in accordance with ethical guidelines that were presented to them. Participants were assured that they were not compelled to participate in this research project. The consent form outlined the research title, including its broad aims and purposes. The consent form also assured participants of absolute confidentiality; this was a very important aspect in getting participants to answer truthfully. Seeking consent was necessary as it protected both the academic and researcher from any problems that may have arisen, and also provided proof of the authenticity of the data collected and the processes used (Cohen et al., 2000).

- This study carried the approval of UKZN Research Office to conduct this research with participants (refer to Ethical Clearance UKZN). Each participant participated willingly in the study and completed a consent form (Appendix G). The consent form also guaranteed confidentiality of participants. This study also carried the approval of the Ethical Committees and Research Directorates of the University of Technology where the research was conducted (Appendix D and Appendix E).
- All participants' identities are secret, and participants were assured that what they said during the interviews or during the observed lectures would be kept private.

Participants shared very sensitive and personal information, not only about themselves but also about management issues at the ToT. Sometimes some discussed things that they would never voice to another colleague or to upper management (e.g. management issues with respect to allocation of academic workload, department issues regarding promotion and affirmative action issues). Participants were assured that I would endeavour to keep their identities hidden as far as I could through changing names and the names of anyone they mentioned, as well as disguising places and particular identifying details. However, I did inform participants of the limits to anonymity as well as the challenges that can be posed in maintaining anonymity. Concealing identities can sometimes be virtually impossible (Van den Hoonaard, 2003). I also mentioned that my personal face-to-face interview was an attempt to gain in-depth qualitative research from their candid and informative responses.

Anonymization in this research study was a particular challenge. Firstly, I had designed the research, conducted all the interviews and observations and had “insider” information regarding the status of one of the ToTs. Insider information is associated with a set of particular benefits and dilemmas (Taylor, 2011). It posed a challenge for anonymity. In some interviews, the participants drew on shared knowledge about particular workload issues and academics. This makes their identities traceable. Secondly, there are relatively few ToTs in KZN offering IT programming. This was not unique to this research study. The “small population” problem has been discussed in relation to ethnographic studies by Walford (2005), in which he states that in a small town there is a risk that individuals may recognize themselves in the talk of others.

#### **4.4 Limitations of the study**

This study was conducted at the CS department at a University of Technology; therefore, the findings of this study cannot be generalised, since the sample is small. The IT curriculum at the ToT is rigid and IT academics generally use prescribed textbooks that adopt a particular programming approach (either PPA or OO programming); therefore, this study may also be limited by the methods adopted at a particular learning institution.

The following aspects were not the focus in this study

- Programming paradigms other than OO programming;
- Differences in programming language constructs;
- The curriculum at tertiary institutions.
- Assessment policies; and
- Machine learning, deep learning, robotics and artificial intelligence.

Furthermore, the study at the ToT involved academics, whose responses could not represent the entire learner population at the ToT. A further limitation of this study is that participants may not have been totally honest with the researcher for various reasons, such as shyness or wanting to protect privacy. Participants may have given responses which they considered appropriate, but which may not have been true or valid (De Vos et al., 2002). This was particularly true of responses from academics, who sought to defend their teaching strategy and justify reasons why they adopted a particular teaching strategy or why their students performed poorly. The results of this study cannot be generalised.

#### **4.5 Conclusion**

This chapter discussed the methodology used in the study. It also discussed ethical considerations and the instruments used in the data collection. The advantages and challenges accompanying the use of qualitative research methods were highlighted, and the limitations of the study were also discussed. The next chapter presents, analyses and interprets the findings of the study.

## **Chapter Five: Qualitative analysis and discussion**

### **5.1 Introduction**

**Study Phenomena:** IT Programming, teaching and learning, challenges, students.

This chapter reports the results of the observations from the data derived from multiple analysis techniques via the Nvivo software tool. The chapter includes sufficient results to enable interpretive analysis to streamline and derive value from the data, and to minimise researcher bias as much as possible.

The aim of this chapter is to indicate the overall responses to the various interview questions. The interview questions were built around the objectives, as well as the theoretical frameworks of the study. The results are presented primarily in charts, graphs and narrative descriptions.

### **5.2 The Participants**

Fifteen academics from a University of Technology were selected for the study. Ten academics agreed to participate. Interviews were conducted, and the academics' responses were recorded and transcribed.

The results (based on the case study of the academics from a university of technology) imply that various factors influence the challenges encountered in building strong programming students. Teaching and learning methodologies need to evolve accordingly, and some interesting suggestions were made.

### **5.3 Biographical data**

This section summarises the biographical characteristics of participants. The participants were made up of academics from a ToT in South Africa.

#### **5.3.1 Participants qualifications**

The qualifications of participants were required to determine their competency in teaching a programming language to the first-year group of students. All participants had a minimum of an honours degree in computer science and specialised in the field of IT, with a relatively higher number specialising in programming. This added value to the sample. Furthermore, all

the participants indicated that CS and IT are perceived as dynamic knowledge-based subjects which are continually evolving, and programming languages are continuously changing. Participants indicated that even though they are qualified and possess either an undergraduate or postgraduate qualification, programming language syntax and paradigms have changed to the notion that they have had to update their programming knowledge by attending programming boot camp workshops or self-study certification in a programming language. Hence, participants either avoided teaching a new programming language or remained with a programming language course which they felt comfortable and confident to teach. This observation and deduction has led to section 5.3.2: the length of teaching programming.

### 5.3.2 Length of time teaching programming

Figure 5-1 indicates the length of time that participants have been teaching programming at tertiary level.

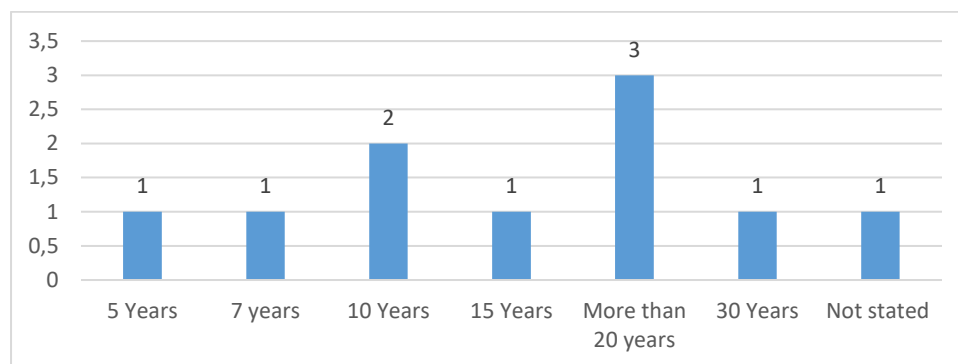


Figure 5-1 Length of teaching time

The majority of the participants have more than ten years of teaching experience. There were three participants having more than thirty years of experience. This shows that participants were very experienced in teaching and hence their contributions added value to the research. However, as indicated in the previous section, participants did indicate that they were repeatedly teaching the same programming language with the same problem sets and similar assessment criteria over a number of years. The advantage was the professional growth of the IT academic who could refine, re-structure and evolve the programming subject curriculum; however, the disadvantage was the underdevelopment of the programming module curriculum in relation to software industry requirements.

Mature IT academic staff were also schooled and studied programming in the structured programming/assembly language paradigms of programming languages. Participant 3

indicated that the language of choice when she initially started studying programming was assembly language. Whilst one acknowledges the historical background of participant 3's scholastic endeavours, the programming languages of 2019 are visual based and a millennial IT student has far more online resources available. Within hours of using visual programming techniques, together with a basic understanding of simple engineering techniques and a suitably qualified facilitator, a millennial IT student can program a simple robot to perform basic functions (by uploading and using an application from their smartphone). Considering the age and/or experience of IT staff, IT management/heads of departments may not necessarily appoint the oldest and most experienced staff to teach first-year students; young vibrant, enthusiastic IT staff who embrace the proponents of the Fourth Industrial Revolution may be far better suited to motivate and teach young vibrant first-year IT students. With respect to the sample population there was an almost equal distribution of lecturers teaching either first, second or third year IT students.

Section 5.4 describes the themes, as per the interview and observation sessions with the IT academic staff. Each theme includes at least one direct quotation from the participants to emphasise the theme or sub-theme under review, and where applicable a relevant review of the literature is included to either support or dispute the assertion of the participant's claim. The results are described in the themes given below.

#### **5.4 Thematic Analysis of Qualitative Data – Identification of Themes**

The objectives of the study were to examine the teaching strategies which are used in the IT classes at a ToT. This generated the relevant research questions. The interview schedule that was used as the qualitative data collection instrument provided rich in-depth data from the participants. By using NVIVO 10, inductive coding was done during which themes emerged from the data. A thematic analysis was performed to derive meaningful insight into the data.

Qualitative thematic analysis techniques such as word cloud, tree map and cluster analysis were used to analyse the transcribed data. The results are presented as pictorial representations of the narrative analysis of the data. All these tools assisted in identifying themes and sub-themes.





### 5.4.2 Treemap

Treemaps are a popular tool to visualise hierarchical data. Nested rectangles represent the items, and the area of each rectangle corresponds with the visualised data for this item. The treemap is one of the primary methods used in thematic analysis and plays a crucial role in generating the respective themes (Sondag et al., 2018).

The following treemap was generated from participants and provides a hierarchical structure of visualised data for two of the most frequently used words from the qualitative data with regard to students and programming.

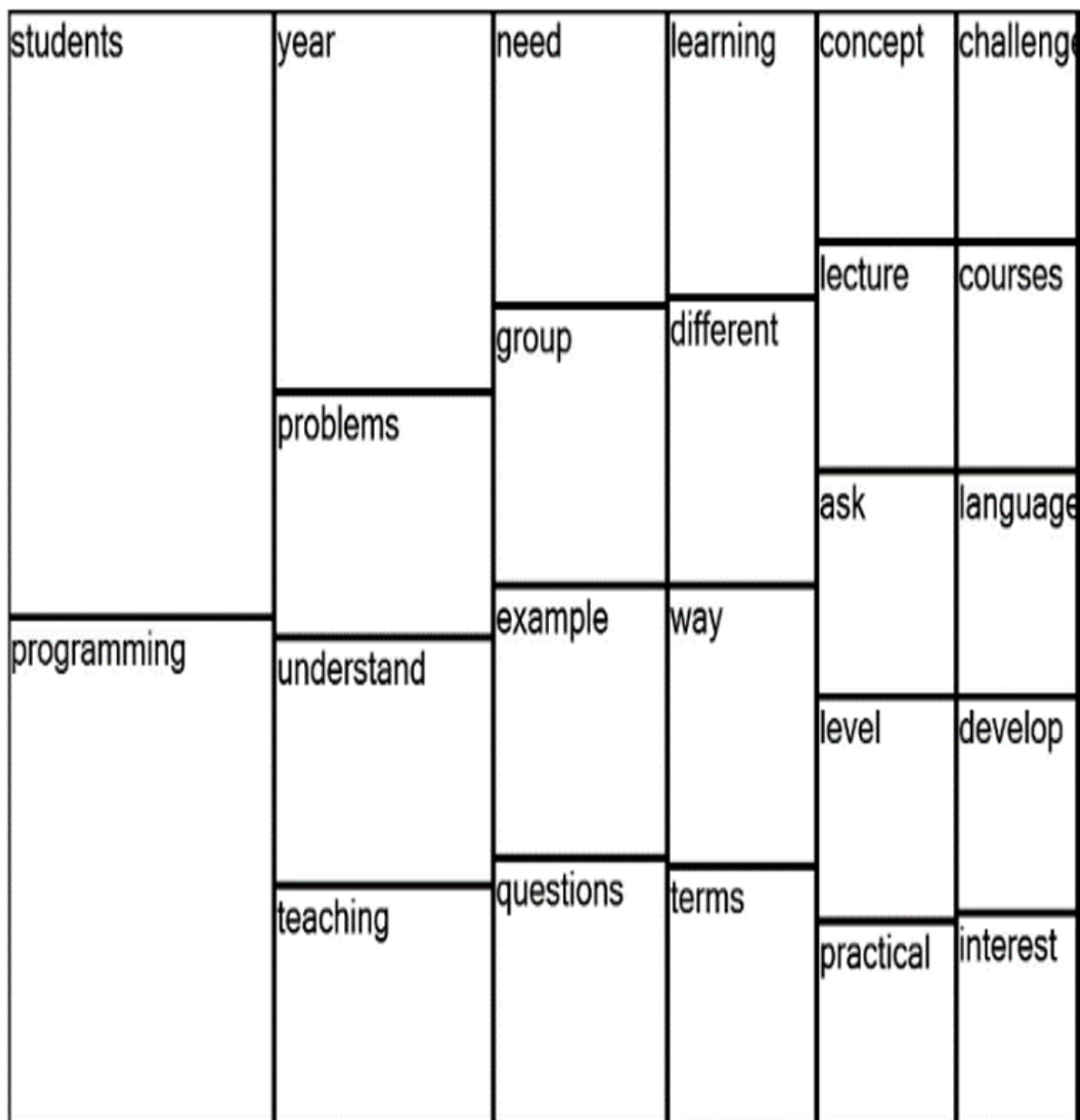


Figure 5-3 Treemap generated from qualitative data

Treemap name	Description
Students	<p>This was by far the largest tree map generated from the qualitative data. It immediately connects to the word “programming”. The map shows that all data obtained centred on the concept of programming students which further adds validity to the data obtained. Some of the phrases from the interviews with the participants linked to the words: year, problems, groups, challenge, and concept.</p>
Programming	<p>The key focus of the study was programming strategies. Therefore “programming” becomes an important variable within this equation. The word “programming” generated a very large tree. Some of the branches showed phrases such as understanding, example, level, practical, and assessment.</p>

*Table 6-1 Treemap Data derived from the data analysis*



## **5.5 Formulation of themes**

Based on Figures 5-2, 5-3 and 5-4, the following key themes were derived:

5.5.1 Teaching (*largest theme*)

5.5.2 Learning and impact (*second largest*)

5.5.3 Challenges

5.5.4 Recommendations – teaching programming

Some of these themes also gave rise to sub-themes. These sub-themes were equally important and allowed for a more in-depth analysis of the qualitative results.

Each theme is unpacked into its sub-themes in the sections that follow.

### **5.5.1 Teaching**

Teaching was the largest theme, which is logical as the participants did a deep-dive into the critical teaching impact of programming on students. It was informed by the following sub-themes:

5.5.1.1 *Concepts and topics*

5.5.1.2 *Assessments*

5.5.1.3 *Pair programming*

5.5.1.4 *Support and engagement*

5.5.1.5 *Teaching support*

#### **5.5.1.1 Concepts and topics**

This sub-theme examined the concepts and topics which needed to be taught to students and how lecturers adapted to this.

##### ***Teach a new concept***

When it came to be teaching a new concept, the following factors influenced this activity:

- **Context**

Context teaching and learning was used to make examples applicable. This included:

- o **Scenario-based**

Giving scenario-based examples that students can identify with is useful. This allows the students to understand how the concept applies to their context.

As per participant labelled 1:

Uh well before a lesson uh it's good to have a good scenario you know. You give the scenario to the student you just explain it to them so that they can understand it before they can start even coding you know because that's what we encourage them even during the test that if you don't understand the scenario then you won't be able to do even the simple calculation that are needed on the program.

- **Real-life examples**

Relating to the above, using real-life examples enables students to comprehend concepts more easily. Such real-life examples must be relevant to life experiences of first-year IT students. It is my conjecture that discussing pension pay-out scenarios and retirement annuities to first-year IT students is irrelevant and frivolous.

- **Use in society**

Showing how programming can benefit society increases students' motivation to learn new things. This appeared to an under-utilised teaching strategy. IT academics must include relevant/updated programming examples from the students' life situations.

- **Creativity**

Participant 3 suggested that bringing creativity to the classroom was pivotal when teaching a new concept. Gaming techniques were hence used by participant where the lesson was turned into an understandable and applicable game.

My famous strategy that I use, especially in first-year, when I get there I just tell them we're going to play. We're going to play a game now, and then they say we're here to play games? Then I say yes, we're here to play games. My games are simple as let's say we have three buckets or baskets, we have three types of foods, we're going to try as quick as possible to throw bananas in the first one, apples in the second one and oranges in the third one. And then imagine this big box that has a mixture of these fruits that after picking you must quickly make sure that you throw it where it belongs. They

say okay. Then I say to them that it is possible that you will make mistakes, right? They agree. It is possible that you pick an orange and then you because you want to do it faster, you put it where apples are supposed to go. If you make that mistake, because this is a game, all those apples, because you threw an orange there, they will get rotten. Then it's either game over, or you get an opportunity to recover. Then they say okay, where are we going with this? Then I go to the concept of data types. If your data type is a double or an integer, you are never ever allowed to throw in a string in there. Does it relate to the game? Then they agree. If you do that, then the double gets rotten. That rotting is what we call an error. So, when you get an error as a programmer, it is your responsibility to read the error and interpret the error so that you can recover from the error, otherwise, if you don't do that, it is game over. You choose to stop doing programming.

This simple yet effective and creative example from participant 4 relates to data types in programming languages and generating errors during program compilation.

#### - **Resources**

Teaching and learning resources such as online videos and tutorials were useful.

Online videos proved to be a most effective in consolidating practical lectures. This led to the creation of a YouTube channel. During the course of this study 95 videos were uploaded. There are 106 subscribers to the YouTube channel, and since inception the YouTube channel has had approximately 9943 views (31 July 2019).

The video can be viewed at

[https://www.youtube.com/channel/UCj2B-Lfva2Wpbmc0j\\_UN\\_0w](https://www.youtube.com/channel/UCj2B-Lfva2Wpbmc0j_UN_0w)

Students preferred videos that were created by the lecturer due to the personalised filming and sometimes inclusion of a student in the video.

Videos had to be approximately 3 to 5 minutes in length. This was because of smartphone data cellular costs.

My oldest and most popular YouTube video was published in March 2015 (incidentally the start of this research project) and since then it has had 2835 views. (31 July 2019).

```
1 //The municipality of a small tow
2 //for each home (or business). Th
3 //HTS The FFerst 20 units are free
4 //HTS A FFxed rate of R10 per uni
5 //HTS If more than 40 units but r
6 //additional units).
7 //HTS If more than 100 units are
8 #include <iostream>
9
10 using namespace std;
11
12 int main()
13 { float units, amount, temp;
14   const float RATE_0 = 10;
15   const float RATE 1 = RATE 0 * 1
```

## waterbill

2,835 views

10 2



**Prinavin Govender**

Published on Mar 12, 2015

*Figure 5-5 Most popular and most viewed video on my YouTube video channel.*

The video can be viewed at [https://www.youtube.com/watch?v=\\_YWLTPV8bk](https://www.youtube.com/watch?v=_YWLTPV8bk).

The video is a problem-solving scenario in the C++ programming language. The program highlights the “For... Loop” and the IF ... Then” programming structures.

Considering that in the last four years I must have only taught a maximum of 100 C++ programming students, 2835 views is phenomenal.

- **Preparation**

Strong preparation for lectures which entailed using lots of relevant examples until the students understood the concept.

- **Experience**

Previous experience in teaching, such as teaching first-year students who have not been exposed to programming before, stimulated ways of teaching new concepts. Hence, this promoted better teaching at other levels post first year.

As per participant 4:

That is a really good experience I had in my first year because in first year I got really good experience about teaching object oriented programming, that was a new topic which I had taken this year. But preparing for the new term, the new subject, it really helped me a lot because I literally came to know more about what actually the subject is about.

• **Comfortabilities and difficulties**

- **Comfortable with syllabus topics**

Six of the ten participants were comfortable with syllabus topics.

Two participants gave further explanations which included:

○ **More research needed at times**

It is good to do proper research on the topics before the lecture commences.

○ **Experience**

Experience plays a role in making lecturers feel comfortable and confident to teach.

However, three participants were not entirely comfortable with the syllabus as new concepts were always surfacing in the IT world. Similarly, module descriptors also changed at times and new items were introduced.



- **Topics not comfortable**

The following were topics and concepts that lecturers felt were challenging to teach.

○ **Decision making on what to use**

The decision-making on what to use in terms of “**selecting case**” seemed challenging for one participant. This was because it was usually a personal preference, but when teaching, one needed to base it on what would suit the scenario.

As per participant 4:

Like for example, I will give you an example, I will try my level best personally in terms of decision-making structures to use my e-statement compared to your, what's the other one, select case. It becomes a personal preference. But I'm very challenged as an academic not to show that to my students because I must give them both so that they can see which one works best for them, just like how I can you know, try to use my e-statement and get stuff done.

○ **Mobile apps**

Another participant felt that mobile apps were difficult to teach as it was something new and that they had never taught it before. Mobile apps were different from desktop apps which included new protocols and sessions. Such concepts were something new and not learned before and hence presented challenges.

○ **Security application to programming**

In addition, incorporating security application into programming was difficult because there was a variety of them to cater for different types of security threats (hackers). Hence, knowing which to use and how to apply it to the program was challenging.

○ **Helping learner with difficult topics**

Only two participants replied to helping learners with difficult topics. Their responses tied in with research and understanding of the problem.

By researching, through reading online and other resources, this allowed understanding different ways on how to address a problem and how to apply new techniques.

As per participant 10:

It is important to do your research. It is important to know basically, for example, what are the different ways that people hack into the system. One of the ways is maybe a SQL injection, and then you have got to know now how you write code to combat a SQL injection. And there are certain set ways that you can do that. So, you show them the set ways.

Participant 7 supported this notion:

Well, I research a lot, and I use again now when I'm having a theory lecture, I do a lot of research, and I go into simple Google basic understandings of like what is a session? What is a protocol? You know, and I take them to things like that. Then when we're doing transmissions, break it down. What is transmission? And how you must. And then it helps me as well.

- **Difficult concepts – programming**

There is a plethora of difficult concepts to teach in programming and this includes:

o **Looping**

Looping seemed to be a popular area of difficulty, and this included the **control flow** logic aspect. Coding process for the control flow needed to be properly understood to determine how the program would be developed.

As per participant 6:

the control flow logic it tends to be, because students, sometimes they think in a top-down approach because when you get that data you have to make decisions now we'll have to write codes that breaks here and here and here and some of them have a, a, a difficult uh time understanding how do I structure how the code actually flows from top to bottom, years ago people used to do what we call this thing top-down code right what was there they start from the top and then they do a go-to bridge around this, and then do this and go somewhere and then in the end they don't get a clear view of what the picture looks like the other thing in order to alleviate that what we tend to do is to

teach them uh what we call flow charting, draw these little boxes that'll give you a clear view of what your program will look like so that when you go start coding then you'll know I'm here, I'm here, I'm here, so I would say that control flow tends to be a problem for some.

- **Decision-making**

Decision making structures seem to be challenging for students to understand which in turn makes them difficult to teach.

As per one participant 9:

Concepts around [inaudible 24:19] decision making and this one I wouldn't say is difficult to teach, but it says a lot about where most of our students are coming from. They use of functions, methods in a program.

But the most difficult for me are you know, your decision-making structures, your [inaudible 25:07] and functions.

- **Mathematical concepts**

Students are battling with simple and basic mathematics. This implies that they may not have done mathematics at school level.

- **Diverse programming structures**

The diversity of programming structures makes it difficult for students to grasp all of them. Each structure has its purpose and use. Sometimes, students confuse the use of such structures.

As per participant 5:

But the difficulty comes in when you have different kinds of examples and the different way in which you can use programming structures. So, if you give students different kinds of examples, then they can be able to master certain programming constructs.

- **Object orientated programming**

Students did not understand the concept of "objects" in object orientated programming.

- **Development of problem-solving ability**

Students lack problem-solving abilities, and participants expressed difficulty trying to teach students to decipher problem statements and ultimately to arrive at a workable solution.

- **Variables**

Students experienced confusion using variables and numbers.

- **Arrays**

Arrays are also a difficult concept for students to grasp and therefore other ways of teaching them need to be found.

- **Logic**

The students grappled with logic and reasoning which could not really be taught. Lecturers provided examples to allow students to see the logic behind them, but they still found it difficult to comprehend.

As per participant 8:

Look, again it is the logical reasoning and the logical understanding. For years and years, we're trying to get them to see the logic, but remember logic is not something we can teach. We can probably give them examples and take them through solutions, but for them to see a solution and to build it logically, they must do that on their own. So that is the biggest challenge, the logical *reasoning*.

- **Outside the class**

Outside of the classroom learning is seen as crucial for student development and progress. The following sub-themes informed this.

- **Key points of outside the classroom learning**

Learning outside the classroom did derive some interesting aspects which included:

- **Healthy competition**

It helps to stimulate healthy competition amongst students. When they are informed that organisations hire the best programmers, they are motivated to work hard on projects for their group to stay ahead.

As per participant 2:

One of my interesting lessons? I think I had so many. It was during a project-based programming module. When I showed them that external companies, even institutions of higher learning are in competition, we want to attract the best students. Similarly, companies want to attract more clients. So, as you work in your groups, you want to develop a better solution than the next group. It is fine; it is called healthy competition. Think about group B and group A. Group B obtains the highest mark, would you like that? That's when I saw students working together as a group where they do become selfish in a sense that now they don't want to share with the other group because now they would divulge so much information that will make them very competent.

In addition, it also **stimulates students** to want to learn and make more time for lectures and to stay for the full duration just to learn. They also meet with the lecturers to get more feedback and ideas.

When I did that I saw my students staying so long, like staying for the whole duration of the lecture whereas before they would stay for, if it's a double, they would stay for one hour and convince me that no, they will still arrange meetings since it's a project that they will come and work on, on the project and show me what the meeting would be about. Then maybe sometimes I would let them go, if there's nothing, then I will also do. But on that day, I started to create the competition and show them that they can compete amongst themselves, they can be the best.

- **Providing future knowledge to students**

Another participant (numbered 6) made an interesting point whereby they provided future knowledge to the students in terms of telling them what they could expect at future levels of study.

I also, during that time I always refer to where they're going. If I'm in first year, I tell them what's happening in second year; I tell them what's happening in third year. I always refer to, you know what's happening in our DS3 when we have these projects and have the opportunity to present to industry people, I make them think about that and I tell them that it doesn't start overnight, it starts here in your first year, you see.

This then motivates students to work harder and be competitive. This further enhances learning.

- **Innovation to come from students**

It allows for lecturers to challenge students to become innovative in their work. Leaving students to tackle a project on their own with guidance from the lecturer (not being spoon-fed) allows them to think out of the box and use their imagination to conceptualise the system.

There was a group that built a fleet management system they actually went to a trucking company and interacted and asked all of these questions because for me when I was creating projects for them I said innovations will come from you I'm not going to tell you, I won't say step one do this, I'm all I did was say here's a big problem find a solution and how you are going to code it, uh I just want components but I do want you to use three-tier architecture but your implication is up to your imagination but then again to answer your question send them out there to talk to people who are going to be reading, using what they are going to build.

- **Finished product**

It allows the students to come back and deliver a product that is industry-related. Such products are very dynamic and positive.

- **Topics discussed this year – for outside the classroom**

The following were topics discussed in 2018 relating to outside the classroom learning.

- **Socio-economic**

One participant introduced a topic related to socio-economic issues. By asking students how understanding programming can address such issues and contribute to a better life for society, it helps them identify with topics.

Participant 4 stated:

Topics range from social economic issues, you know, I try to show them that if they don't learn to solve problems the better way, what type of employees will they become? I go as far as showing them how incompetent they will be when they are at the work environment, you know. So, there are various topics there. There are various topics that I use, or I discuss with them when I send them outside. I show them how much problems we have out there that need them as programmers so that they go out and start thinking outside the box and they say that there is always something out there to solve, so let me go and find something to solve, and they always come back with something.

- **Assignments**

Assignment topics for students' own time purposes are given which related to out of classroom work.

- **Multiple projects to choose**

Participant 9 gave students the freedom of choice by putting various projects on the table and allowing them to choose the one that was most appealing to them.

Uh here I haven't since I've, I'm new but what I did uh when I was dealing with projects, I designed five projects to select from and then what I wanted them to do is go out there.

- **Build application to help others**

Students were encouraged to build applications to help others and not just wait to be told what to build. An example was made with IBM.

I'm saying when you build an application you don't build it for yourself you build it for other people so you need to get input for them as to what do you want, what would you like to see because I remember years ago IBM used to be notorious for telling you that you want what we build for you, we're not interested in what you want.

- **Mobile apps**

Mobile apps are the current buzzword in the technology world, and hence students need to ensure that they can develop them. Mobile apps are not straightforward to build and cannot be built on just classroom knowledge. Therefore, students need to go out there and research new ways of developing apps.

### **Programming lessons**

Programming lessons was the largest subtheme of the teaching theme. This is logical as programming is built on the strength of programming lessons. It was therefore unpacked under the respective sections to follow.

- **Programming language**

The programming language was scrutinised to determine which language was most widely used by lecturers.

- **C# (pronounced c sharp)**

The most widely used was C# for the following reasons:

- **Experience**

Only two participants have prior experience and exposure to the “C” languages and hence built their expertise.

- **Curriculum and module descriptor**

Another three participants conveyed that C# is part of the curriculum and module descriptor, and that they were compelled to teach what was prescribed to them.

- **Widely used**

As per participant 5, C# appeared to be the most I most widely used language.

- **Very structured and easy to learn**

C# is very structured and easy to learn and includes the visual studio IDE with various tools.



- **Object orientated**

C# is object orientated which complements the current IT development world where everything revolved around objects.

- **Focusses on functionality**

C# is not a “fancy” language such a JavaScript, as C# is built primarily to enhance the functionality of the program. Functionality is the most important aspect rather than look and feel in all programs.

As per the statement from participant 8:

I would use I guess is C# because it's not fancy you're not worried about colours and all of that. You just want to say here is the algorithm that an insurance company uses to calculate the premium, so you have to understand that I'm taking this variable, and then I'm multiplying I'm doing all of these things I'm putting the data in the database but if I'm teaching someone that's building user interfaces C # is not the best language to use for that because it's designed to create objects that are inherent with other things whereas on the front and even though if you look at Java Script there is objects already in programming but most of the time you spend at the layouts so the language that are very good at doing that, Angular for example may be bootstrap and the other libraries that are available now, then I go back to the back end from the database then I won't use Java Script for example to do that, C# wouldn't be, you can depending on the platform because I can write C# modules that I can, that use, uh if I'm using Microsoft sequel server, uh to run those modules to process the data, but then I would use SQL which is a language for that particular environment, to answer your question, I hope I am.

- **Java**

Two participants preferred Java because:

- **Syllabus**

It was part of the syllabus

- **Best object orientated language**

Java was one of the most object orientated programs. The world is moving towards object-oriented programming such as mobile and gaming. Hence Java is one of the most compatible with these developments.

- **Visual basic**

Only one participant preferred Visual Basic as it was a very intuitive program.

- **Programming principles- algorithms**

Participants 2 and 7 made an interesting point. They preferred to focus on the “algorithm” writing first. Hence, this should be the most essential phase before moving to programming. If the student had solid grounding in writing algorithms, then programming in any language should not be difficult.

• **Preferred level of programming**

A majority of participants interviewed preferred teaching the senior level of students (second year onwards)

- **First years**

Three participants preferred first-year students because they mainly enjoyed teaching them new concepts for the first time and building their knowledge from the start. However, coming straight from school level and the lack of understanding of the tertiary environment was a disadvantage.

- **Second year**

Another three participants preferred second year students. Some of the reasons included that by the time they reached the second year, those who did not share a passion for programming dropped off and those that did make it into the second year were those who have the drive and passion for it and are thus able to cope and do well. Another participant felt that it was easier to teach the second-year students if they taught them in first year as well. This means that they would be acquainted with your style of teaching and able to track back to what you taught them in the first year.

As per participant 10:

Uh I think uh I would say second, but you know what we did last year – we carried our students from first year and then we moved them to second year, so to me that was very nice because you know, you know what you taught them in first year and now when you get to second year you know it becomes, you are available to refer them back to what you did first year. Now when you are in a second year, and you never taught them first year you find that you are referring, you trying to say “No you must’ve done this in first year” and they say “No we didn’t do that in our first year” so uh I think it would be nice to move your students from first year up to second year and then maybe you can leave them there you know, but yeah I would say second year”

- **Higher levels**

Four participants preferred higher level students, which meant mainly the third- and fourth-year students. Some of the reasons were because students were already acquainted with programming languages in their first and second years of study and were now at the level of good programmers. It was hence easier to teach without spending time on the basics.

• **Lessons vs reality**

This subtheme measured the phenomenon of programming lessons in comparison to the real-work environment. It was informed by three sub-themes listed and explained below.

- **Making lessons real**

The following was done to make lessons real.

○ **Translating text-book examples to local application**

The current textbooks mainly have examples from developed countries with different attributes. For example, the currency may be different. Hence it is important to make those examples relevant to South Africa. Students are then able to better understand the content of the lesson.

Building on this, keeping examples relevant to the BRICS context makes a positive difference in student understanding.

As per participant 6:

I will give you an example like last year, my final paper was about Brix that's going to happen this year, which is 2018 in Johannesburg, because I was sitting there listening to the news, watching the news actually and they spoke about that, then I said to myself okay, there's a scenario for me, let me bring it home. And, I created a scenario where a South African needs a program that will help them to capture you know, sales that they will make out of their products because this South African person wants to take advantage of the Brix event. Remember when you're talking Brix, Brazil is there, Russia is there, so what I'm saying again, bringing it to reality starts with even the smallest exercises as we build up to the bigger assessment. So, I tried to make everything real and most importantly for me to localise stuff for them, so it makes sense...

- **Teaching material – theory to practice**

It is always good to convert theory into practice as programming is a practical subject.

- **Social media**

One participant indicated that social media could be used as a strategy to inspire students. Social media such as Facebook use **Graph Theory**. This is in the background that gives Facebook the functionality it has. Hence, when a student becomes aware of this, they know that they could program platforms such as Facebook, and they get excited and inspired.

- **Skills-based – applicable to any context**

Programming is a skill-based subject and not only knowledge-based. Hence, by focusing on building that skill, it can then be applied to any context.

- **Real-life examples**

As mentioned before, utilising real-life examples promotes better identification with students and context.

- **Go down to students' level to promote understanding**

More effort is needed to go down to the students' level to get them to understand programming because of their limited exposure to subjects such as computers and mathematics at school level.

To quote participant 4:

In programming, actually since they have some basic knowledge about mathematics. I can say 50 percent of students; they literally understand what we are doing in programming. But even there also if they don't understand, we must literally go a step down and try to make them understand. We don't go and really teach them the proper maths of how to do an average and all those things, but I'm just saying we give an example like saying this type of programming, our mathematics, what we do in programming is just a basic mathematics that you've already learnt in your previous studies...

- **Strategies – make lessons real**

Based on the above, the strategies employed to make lessons real included:

○ **Multiple examples**

Providing multiple examples on the same programming structure promotes more understanding.

○ **Continuous teaching**

One participant believed that there was no strategy except for ongoing teaching.

○ **Context**

Providing examples relevant to students' contexts was a good teaching strategy: as indicated from participant's 4 statement:

You have to look at now where the students are. A very good example that we use is a simple university example where students are paying fees, calculation of marks for their subjects, in terms of DPs, calculating DPs, final marks, etcetera. So, we look at things that they're currently involved in, and they can relate to.

○ **Breaking down complex problems**

Showing students how to break down a complex problem into smaller components prevented students from becoming overwhelmed. This further allowed for them to see that any challenging problems can be solved if segregated to smaller components.

- **Bi-directional interaction**

Lecturing needed to be a two-way process. It was not just where a student passively absorbed, but instead bi-directional, where the lecturer describes a scenario and students define the problem and possible solution.

- **Application to be more practical vs abstract**

It was essential to focus on programming that is practical and that relates to daily life. Students need to be aware of how programming can create platforms (such as apps) which can be meaningful.

This was supported by a participant's comment (participant 3)

Yes, I find that your application must not be dealing with abstract things. It must be real-life things, and to show students how by developing an app you can make something easier for the end-user and more meaningful. So, you have to focus on the practical stuff that students encounter daily.

- **Learner's response to programming lessons – real teaching material**

This sub-theme examined how learners responded to programming lessons with relation to using real teaching material from the environment.

- **Good response**

Three participants conveyed that the response was good, and this was due to the students' **ability to relate to their own contexts**. This was based on the following:

- **Card systems**

Context related examples like calculating the cost of electricity and the card systems/meter systems make students identify better than if you talk about abstract concepts.

- **Related organisations**

Students related better with organisations that they know and can develop systems more effectively for such.

Participant 7 indicated the following regarding a Non-Government-Organization(NGO)

I find them to respond very well. They also get excited in a sense that after you give them one example that they can relate to, the next morning they come with scenarios of their own, and they will come and say hey you know, I was thinking of doing a program for our church, I've noticed that they're struggling with one, two or three. I was thinking of doing a program for the NGO next door. It is because you took this one example that they can relate to you know, and then you brought it to the context of programming and so it also makes them to be creative. So, they get really excited about that.

- **Social media**

As mentioned earlier, the example of Facebook and its Graph Theory stimulates students to think out of the box.

o **Not good response**

However, another three participants conveyed that the response was not good, and this was due to **school level knowledge lacking** in students. This was based on the following:

- **Accounting principles**

Students lack the basic accounting principles. This includes even the calculation of key items such as VAT and interest. Hence without these basic principles, it is hard to develop a system that needs to fulfil business needs.

- **Maths background**

Similarly, mathematical principles are important in programming to understand formulas and calculations. However, due to students with limited maths background, they do not respond well.

o **Irrespective of context – innate ability**

Two participants, however, also made very interesting points whereby they felt that programming is an innate ability. Irrespective of background and context, it was either the student had the ability or passion for programming or they did not.

As per participant 2:

Programming knowledge is independent. Programming skill is independent of the context because your programming skill is an innate ability, it's an ability that you are developing, and it's a problem-solving ability. It's the ability to look at a complex system or let's use the word a complex problem and dismember that, break that down into constitutes and then work through solving the problem.

Participant 7 agreed with participant 2 and stated:

...this question you know; I guess it would depend on how the students actually take to the concept if they're able to grasp it. If they are because there are a lot of students that are built for programming and then there are those that just don't get it. So, it would depend individually on the student themselves. It could go either way in this case.

- **Interesting lessons**

When it came to how participants made lectures interesting, the following factors emerged.

- **Real-life application**

As mentioned before, the application to real-life settings was important for students to identify with.

This stimulated their interest as testified to by participant 8:

The real-life applications then that is when students start becoming interested. For example, if you ask them to create a system to work out maybe the tax that the parent pays for a year and to create an app. So automatically the student there will be an interest there because they want to know how the tax system works and maybe they will use real-life examples like their parents. So, where you can bring real-life examples in students become interested.

- **Mobile app**

The development of mobile apps mainly stimulated students. Today the world is dominated by mobile apps and social media. The fact that they now can have the skills to develop such things keeps them motivated to learn.



- **Sensitive information – security and integrity**

Showing students, the importance of key aspects in the IT world such as cyber-security when it came to the security of information and confidentiality of data, highlights the importance of programming from a security, integrity and anti-hacking perspective.

- **Concept learning**

When students are taught a certain concept, and successfully apply that concept to the program, it stimulates excitement and a sense of accomplishment which makes the entire lesson interesting.

- **Eureka moments**

An interesting point was made by a participant when they conveyed that “Eureka moments” did exist in the class. This occurred when a student was able to discover the logic and essence of what was being taught.

o **Students discovering the logic and essence**

Students are taught various aspects in programming such as algorithms, process flows, branching and sequences, amongst others. Hence when a student finally grasps what these things mean and how they can be applied, it is defined as a “eureka” moment.

Participant 9 attested to the “eureka” moment, in the following quotation:

No well for example, if you're going through the, if you're discussing algorithms now and you're discussing process flows you know your sequence, repetition, uh branching, you know that type of thing and then it's, for me, it's when the students, when it dawns on the students, and you can see, ah this is what he means.

- **Thinking out of the box**

Participant 10 felt that one must stimulate students by having interesting lectures that make students to think outside the box at all times when doing examples and coding. This participant did indicate that it was difficult to achieve.

- **C#**

With respect to teaching the programming language C#, participant 4 felt that teaching C# was interesting to students especially to those from a Java background. Participant 4 stated that C# was simpler when it came to aspects such as syntax.

- **Lesson Preparations**

- **Frequency of lesson preparations**

Lecturer preparations were grouped in the following clusters.

- **Daily**

Two participants prepared daily for lectures.

- **Weekly**

Five participants prepared for lessons on a weekly basis.

- **Before lectures**

Two participants prepared before the lecture.

- **Structure of semester**

No participants prepared based on the structure of the semester. By a certain timeline, certain areas would have to be covered.

Participant 6 stated the following:

I will, at a higher level to get an overall picture of how the semester is structured, I will prepare at a higher level to say by the end of three weeks I must have covered this, then it comes down to weekly and then daily.

- **Thematically**

Participant 9 conveyed to the researcher that they prepared thematically, however, no explanation was given.

- **Preparation before any Programming lesson**

The results for preparation activities done by lecturers before programming lessons presented a variety of activities.

- **Writing programs before class**

Two participants wrote programs before class to ensure that they were well versed with the program before teaching. This allowed for all errors to be picked up so that students won't make mistakes or learn the program incorrectly.

As per one participant 2:

Secondly, you have got to write the programs before you go to the classroom. So, when you write the program before you go, you can pick up all the mistakes. If you write on paper, you might not pick up certain errors, so you got to go onto the computer. You have got to write the program; you got to run the program. You got to know where the problems were. And if you make those mistakes, the students are most likely to make those mistakes as well.

Responded 7 added to the above discussion, the quotation is as follows:

First of all, I try myself, before going to the class, I try working with the program, execute it and try to get more inputs so that I can give more examples to the students when we go to the class.

- **Use applications relative to the class**

It is important for students to identify with specific examples and concepts. Hence one must go through different types of examples to choose those that are applicable and relative to the class. This must be part of the preparation process.

Participant 3 had this to say about the use of applications in class:

And then you have got to pick out the proper kind of application for the students. You cannot pick out stuff that are unrelated to what you are doing. Or sometimes there might be a problem in the question, so you got to go through the application. You got to go through different kinds of applications, and you got to choose the application that you

want to give students, and you also got to know how many examples to give them. You cannot give them too much.

- **Full knowledge of exercises**

It is important to have full knowledge of the exercise beforehand to teach confidently without mistake or errors. Reading the textbook and understanding is important as well.

With respect to lesson preparation participant 1 stated:

Firstly, you're going to look at what you're going to teach. What's your topic? So, you got to be able to be prepared and understand that well because you can't went fumble when it comes to programming, that's just going to throw the students off. So, number one you got to be very clear about the syntax or the structure you're teaching.

- **Based on academic strength of group**

There is no “one size fits all” strategy to preparation as each group of students’ differed. There were stronger groups and weaker groups. Hence respondent1 based their lesson preparation on the academic strength of the group being taught.

- **Different streams**

Like above, there were also different streams of IT of which each had its own programming specialisations. Hence preparation needed to be done based on that.

Participant 9 had the following to say regarding teaching different streams of students:

But for this group, because they're like how they are, and I will give you an example, like in second year we have a group that is BA streamed and we have a group that is applications development, because the BA stream did a project that is business based, they lack HTML skills. But these ones that did an applications development project, they could get those skills in their project module. So, you see now, my preparation for the BA group is different for these ones. These ones, they know their HTML, they can grasp [inaudible 30:25] I have to start somewhere officially, this is where it differs.

- **Use multiple resources versus audience**

Sometimes multiple resources are needed to determine the best approach suited to the class.

- **Building data structure**

Participant 3 based their preparation on data structure. They conducted two full theory lectures followed by a practical. Hence notes and code must be prepared accordingly.

As participant 3 added to the above point: “I build whole data structure, have 2 theory lectures 3rd lecture is practical, prepare notes then code”.

- **Evaluate complexity**

Lecturer materials had to be prepared in relation to complexity respective to the lecture coverage and class.

- **Context and scenario**

A very important point was made by three participants and this related to context and scenario-based lesson preparation. It was helpful to create a scenario whereby all that is learned can be applied to the particular scenario. Such a scenario must be something that students identify with and is related to their context.

As per one participant 10:

Uh well before a lesson uh it's good to have a good scenario you know. You give the scenario to the student you just explain it to them so that they can understand it before they can start even coding you know because that's what we encourage them even during the test that if you don't understand the scenario then you won't be able to do even the simple calculation that are needed on the program.

Participant 6 concurred:

Uh I need to look at what the chapter is that I'm talking about, think of examples use tons and tons of examples, deliver to the students and then you need to draw them because again programming is not just about learning a syntax you have to put things in context where would I even use, let's say I'm looking at the IF statement where

would I use IF statement because then I would introduce them in such a way to students, give them a short narrative then I'd say if I'm giving different choices where I could choose choice A, or choice A, B, C or D. How do I write that? Why would I want to do that? I'll say I'm giving them an example I'm writing a code for insurance somebody's going to come in they want to uh get an insurance policy, there are different policies that we sell. It could be auto car insurance, could be health insurance, could be pat insurance, could be all of these other things so in my code I'm going to say if they want this do this if they want to do this, do that.

### **Frequency of tasks for learners**

Tasks given to learners were clustered into the following groupings.

- **Daily**

Three participants gave tasks to students daily.

- **Weekly**

Another three participants gave tasks and exercises on a weekly basis

- **Continuous**

Two participants continuously gave examples. This included placing tasks and examples on the blackboard on an ongoing basis.

- **Dependant on complexity of concepts**

Participant 4 made an interesting point whereby the task was driven by the complexity of concepts. Different tasks could be solved by applying concepts.

As per one participant 4 statement:

As often as every concept that I teach, depending on the complexity of the concept, I will try and give them different tasks to show them you can still solve task A, task B, task C with that concept. If it's a simple one, it will be like, if it's a concept, give them an exercise you know, the harder the concept, then I will try to show them that problem A, B, C, they still require you know, this particular concept.

### 5.5.1.2 Assessments

This sub-theme examined the area of assessments. Assessments are important with respect to the measurement of student performance. It was therefore informed by five further sub-themes. Each assessment is unpacked in its own section as follows.

- **Forms of assessments**

Most of the participants asserted that there were **three types** of assessment. However, this varied in terms of how the assessment was structured. Such assessments were as follows:

- **Group, oral and written** (asserted by 1 participant)
- **Examinations – main, practical and supplementary** (asserted by 1 participant)
- **Examinations – test-assignment** (asserted by 1 participant)
- **2 test and assignment** (asserted by 1 participant)
- **Group, practical, theory** (asserted by 1 participant)
- **Unstated** (2 participants stated that there were 3 assessments but didn't explain what they were)

- **Depends on level**

Five of the participants asserted that in the first year there were many assessments, but the assessments became smaller at higher levels of study.

- **Concept driven**

Participant 2 conveyed the notion that 'assessments were based on concepts'. Some were paper based assessments if the concepts were more theoretical. However, if concepts required the running of a program, then these were done at a practical level in a laboratory.

- **Make students think**

Participant 9 made an interesting point that assessments needed to challenge students to be able to think. Hence, in their assessments, they did not just add simple questions for marks, but also complex questions for extra marks.

As per one participant 9:

that I rely on, again I don't like to give them what we call toy programs because you have to make them think, you have to challenge them, you can give them maybe, some, the ones, if I wanted to force them a little bit further I would say here for extra credit, here are the harder problems to actually challenge uh some of them. The problem with that idea is that the ones that are doing well they'll always do the extra points ones it's the ones that are struggling they won't even attempt that.

- **Questions per test-practical-theory**

- **Complex scenarios: taking all into account**

Fifty per cent of the participants preferred setting tests that were based on one complex scenario or question, and all other sub-questions became components of the scenario. This allowed for better inter-related problem-solving.

Participant 6 asserted:

On average we would have one big scenario that will try and bring everything that we taught within a space of time... So, you rather make one scenario to test one component and then the second scenario to test the other two. But then maybe towards the final exam, you can bring in the bigger scenario to test all three of them because you've tested them already in separate scenarios, so the assessment complexity goes up.

Participant 10 added:

It just depends on the duration of the test and it depends on how many sections you covered basically. So, if your test is generally one and a half hours you have got to look at how many practical questions you can test. In programming, there are lots of sub-questions. So generally, sometimes there just might be one or two questions with a whole lot of sub-questions.

- **Four questions per assessment**

Two participants added that it's usually four questions with a mixture theory and code.



- **Three complex questions**

Participant 9 set three complex questions with sub questions.

• **Questioning styles**

The concept of question styles brought a plethora of variables to the fore. Each is unpacked in the list below.

- **Interaction and discussion**

Three participants promoted interaction and discussion which allowed for a two-way process of engagement during questions.

When asked, “Do you want to elaborate on that?” Participant 5 stated:

So, I move around my classroom. I’m one person who just doesn't stand at the front, you know, I move around, I interact with them. If I can go between them in a classroom, I do that. Questions are posed to the class and if anyone has a response also try and facilitate the discussion. So, the first response, if it’s not on the mark, can try and guide the rest, through discussion, try and guide the rest of the class towards where you want to go.

- **Multiple validation**

Multiple validation of questions served to bring about more engagement and facilitated understanding of the questions.

Participant 8 had this to say about multiple validation of questions:

Then after getting an answer I broadcast it across the lecture to say okay, he says, or she says this or that works like that. Do you agree or disagree? Then I can be able to get a response from the overall class, and I can quickly pick somebody up who is saying yes or no and then go straight to them and ask them okay, you said no. Why? I will ask a specific person, and then I will ask peers what they think about that person’s response, and then we will discuss it.

- **Ensuring student understood lecture through questions**

Two participants used questions to ensure that students understood the lecture. By posing questions and encouraging answers, students' level of understanding is revealed.

This is what was said by the participants:

First of all, I ask them a lot of times, but maybe, I don't know, this might seem a little bit scary to ask, but I literally tell them every time before me leaving the class, the last five to ten minutes I try to concentrate on whether they literally have understood what class I have taken on that day before I leave the class.

I ask a question and if I don't get a response then I definitely, I tell my students before the time as well that's one of the ground rules in the class. If you don't ask, I will ask you because then it means you know everything if you didn't ask me. So, if I ask and I don't get a response, well I will probe a little further.

- **Approach**

Approach is very important. A student should not feel intimidated or scared to answer a question. Hence cooperation is encouraged when approached in a subtler way.

One of the female participants stated:

I've noticed that if you're just going to stand in the front and ask them, you're never going to get an answer. Because they're all sitting and looking at who's going to answer that? I'm not going to answer that. But I approach them, I approach a certain student, nicely so not to intimidate them, you know, I may approach them to say your traditional shirt looks nice, you know, tell me this and that and that, how does it work? What did I say in the last lecture, blah, blah?

- **Flexible in answering language**

Due to many students not having English as a first language, it can become difficult to express their answers. Hence one participant allowed student flexibility in how they answered the questions. The participant was fortunate to understand various languages.

- **Posing questions to those that are weaker**

It was important to pose questions to the weaker students. The weaker students were more reserved, and stronger students tended to dominate the answers. However, if it's posed to weaker students, it gives them the opportunity to attempt an answer and learn in the process.

- **Target different areas**

Ensuring that questions were directed at different areas allowed for overall understanding.

As per one participant's 2 reply:

Because data structures are not pure programming it has, e.g. trees, graphs the logic of it usually if I have 5 questions 2 will be pure coding you have to solve a problem the other 3 will be about the logic of the structure, e.g. the tree the drawing choosing the short path and so on.

- **New – full scenario based**

A simple but effective way of asking questions is basically just putting a scenario to the student and then allowing them to work out the solution. They figure out critical aspects without being told what to do. Participant 8 quoted an instructor (from a course attended as a student) who had the following scenario in promoting self-learning:

Yeah, he kind of moved us from asking the student like question one and asking question two then question three you know. He kind of moved us into you put your scenario and then you just have one question that will say 'Develop this simplification, and it must be able to do this and this and this and that' so the student will have to develop a full application in that case we don't ask like questions, declare this variable, declare that variable, do that method, do that method.

- **Breaking down problems**

Breaking down problems through questions enabled students to understand the problem better and derive a solution.

- **Types of programming questions - Tests and Exams**

As above, the types of programming questions asked for tests and exams delivered a variety of results.

- **Diverse examples**

Due to programming being very application based, students need to be exposed to different types of problems. Hence, the types of examples varied. The aim was to ensure that students could apply their skills to any example or scenario.

- **Question to promote thinking**

Participant 5 made a very critical point that questions should be formulated to stimulate a student to think critically. Programming tests and exams are not about memorising aspects such as syntax and other concepts but more on how to think critically to solve a problem. Hence avoiding syntax type questions and focusing more on the actual problem-solving was a good measure of students' ability in programming.

As per one participant 5:

We do types of questions if I'm uh, oh if they are preparing to take a test because they could be two, I guess there could be different types I could give a type of a question that looks for an understanding of syntax, I don't like those, I see quite a bit of those, I want to give them the type of a question that forces them to think, given a real problem write a small code snippet not the entire thing, from maybe the entry point to where the program ends and we'll say write an algorithm to implement this. I'm giving them a statement let's say I want to sort numbers maybe in reverse order, all of those kinds of things you can write because, memorizing syntax is not good if they don't understand how to use it but by giving them a small problem I'm forcing them to think about all, here's how this and that I learned of, how to do sorting, how to do our order numbers, how to manipulate I guess data in several ways so again to answer your question I would like to, I prefer to give them small challenging questions where they need to use their knowledge to do it rather than say if ' $8x = 64$  what's the answer?' that's not good.

- **Entire programs**

Testing students on an entire programme which took everything that was taught into account, enabled the participants to test their collective knowledge of a concept.

- **Past year papers**

Taking examples from past year examination papers and exposing students to similar scenarios by adapting questions and scenarios served as good test and examination scenarios.

- **Defining a framework**

Participant 3 made an interesting point whereby they defined a framework at the outset of the semester and continued to build on that till the end of the semester. This allowed for the capture of critical points to test students on throughout the semester. It further satisfied the learning outcomes. Participant 3 stated:

That's not necessarily preparing them for the exam; I'm satisfying the outcomes. So, I'm, from the beginning to the end of the semester I'm flowing through my framework, my scaffolding that I've developed, my pyramid structure and along the way, there's way points where I'm satisfying the outcomes that are specified in the learner's guide. The assessment, the assessment is just one, the assessment is just one aspect of uh checking to see if you're satisfied with the outcomes.

- **Sketching problems**

Participant 7 preferred sketching a scenario/problem and then left students to approach it in their own way.

- **Similar structure of questions**

Giving students similar type and structure of questions for tests and exams on a regular basis prepared them for the exams.

- **Based on sections**

Participant 8 based their questions on the sections that were covered (e.g. Loops) for test preparations.

- **Preparatory exam paper**

Participant 9 set preparatory exam papers and ensured that these were tougher and more complex than the actual exam paper. This allowed for the students thinking levels to be trained and tested. If they could pass and do well in the preparatory paper, then it was highly probable that they would handle the exams well.

- **Increasing intensity of exercises**

Similarly, another participant increased the intensity of exercises.

- **Revision exercises**

Revision exercises taking all relevant work covered for the semester enhanced preparations for exams.

• **Practical vs theory assessments**

- **Practical**

The majority of participants preferred practical assessments. Some of the reasons included:

○ **More learning of programming**

Practical assessments supported more learning. More exposure to practical exercises enhanced learning ability because programming is a practical field.

○ **Problem-solving ability**

Practical assessments allowed for more problem-solving ability to be brought out in students as programming was built on problem-solving.

○ **Demonstrate the ability to design and build**

With respect to practical examples, participant 2 argued that, a programmer had to have practical examples to be able to design and build a strong program. One could teach much theory and concepts, but if this is not practiced, then students will not gain proficiency to design and build efficient and good programs.

The participant provided the following example:

Practical, yeah because at work nobody is going to quiz your theory they want you to do, show it to me actually one of the things that uh we were doing, most companies in the US now because I went at a (blank) when potential programmers came in we didn't ask them questions we gave them a problem to solve, they would say here's the room, here's the computer's it's got all the software that you can ever want if there's something you want we don't have it we'll go load it but here's the problem that we want you to solve and then you'll have to go and design and build and do that and then you'll come show it to us, so nobody's going to start to ask you, uh theoretical questions tell me what a loop does, no we don't do that anymore.

#### - **Theory**

Three participants favoured theory mainly for logistical reasons.

##### ○ **Resource and logistical constraints**

Electrical disruptions such as load shedding, coupled with other resource constraints hindered effective practical assessments. Such issues then disrupt the assessment leaving the student vulnerable to poor assessments.

##### ○ **Computer related problems**

Furthermore, computer related issues such as students not saving their work and then losing their work during the assessment also compromised the students' assessment marks.

#### **5.5.1.3 Pair programming**

The idea of pair programming was looked at, and it became a theme under teaching which was informed by the following sub-themes.

##### • **Use**

Most participants (six) did not use pair programming. Only four participants used it. Hence the following sub-themes are based on the responses of the four participants.

##### • **Pair programming versus pace of lessons**

It was shown that pair programming does increase the pace of lessons.

- **Pair programming versus assisting slow learners**

Results show that pair programming plays a role in assisting slow learners.

- **More interaction on questions**

When paired, students tend to interact more with each other and the lecturer. As one participant commented:

You're basically walking around, and you can see someone that's not doing anything and why, you start questioning and then of course step by step, okay do this. 'Do that, okay this is the question, so you know what, uh what do we mean by that? How would you go? Or what would you need to write?'

- **On par**

It allows for all learners to be on par when paired.

- **Pair programming versus time constraints for lessons**

When it came to whether pair programming affected timing for lessons, the following was established:

- **Planning becomes easier**

It becomes easier to plan lessons.

- **Timing per example**

It allows for better timing when it comes to examples. Students are timed to solve a problem and present it. One participant commented:

I time them, I tell them you only have five minutes to complete question 2, and then from there I will get two of you to come, and quickly present it and then we move on. You know?

- **Does not affect time in any way**

However, two participants felt that it did not affect timing in any way because with a practical lesson one is not teaching all the time and work given to students is part of the lecture time.



Furthermore, another participant asserted that concepts were still a challenge and more time needed to be spent on that.

- **Challenges – using pair programming**

The following challenges were found with relation to pair programming.

- **Incompatibility**

Students sometimes struggle to understand each other, and this leads to them not being able to work together.

- **Strong dominating over weak**

The stronger students seem to dominate the weak students.

- **Overcoming pair programming challenges**

One of the participants who used pair programming asserted that by increasing the number of students in a group, it served to overcome challenges.

Sometimes you find that these are unfortunately the weaker of programmers, then those need so much observation and intervention, maybe then you can make the pair to work with another pair, now maybe they become four.

#### ***5.5.1.4 Support and Engagement***

This is a significant subtheme that examined support and engagement between lecturers and students for programming.

- **Teaching support material for programming**

Results generated the following when it came to teaching support material for programming used by lecturers.

- **Textbooks**

Textbooks were one of the most common resources. However, it was important to ensure that the text book was recent, as programming is constantly changing.

- **Videos**

Online videos were excellent visual resources to teach programming. Sometimes students needed help on how to search for the relevant right and applicable ones.

- **Blogs**

Blogs were also interactive resources whereby one could ask questions and get answers online. Students could also respond to questions of others.

- **Websites**

Educational e-books and websites such as MSDN and other coding websites allowed for ongoing and updated learning for students.

- **Tutorials**

Tutorials in the form of tutorial booklets allowed for students to work independently.

- **Research articles**

Research materials such as journal articles were excellent learning resources that provided new research knowledge.

- **Supplementary materials**

Supplementary materials to textbooks and lecturer notes were also helpful.

- **Lecture notes**

Lecturer notes were a default resource.

- **Programming Code**

Programming source code was also handed out to students.

- **Past papers**

Past year papers were excellent resources for showing students the type of problems that tests/exams can present.

- **Scenario**

Scenario-based learning presented examples that students could relate to when programming. This enhanced understanding.

- Massive open online courses (**MOOCS**)

Massive open online courses such as UDACITY and COURSERA were excellent self-study resources for programming.

- **Promote positive attitude towards programming**

Results show that participants engaged in the following to promote a positive attitude towards programming

- **Diversity of IT**

Sometimes, a student may not enjoy programming but still wish to be an IT professional. Therefore, it was essential to convey the diversity of IT and that IT was made up of a vast number of different sectors. However, to get there, one needs to pass tertiary level, and that includes passing programming as well.

Participant 7 had the following to say regarding motivating students to study IT programming:

I tell them about how broad IT is. I tell them that as much as I am a programming lecturer, it does not mean that I like everything under the IT umbrella. I expect the same from them. If they feel like programming is not their favourite, that's a major reason why they should pass it and get it out of the way. You know? Just learn it nicely so that when you're sitting with programmers you can at least contribute to whatever solutions, yet you are interest is in something else because we still need networking personnel, we still need data administrators, very competent data administrators. In most cases, you will find that someone who is in networking does not have much interest in these things, your programming. Right, So I tried to build that positive attitude.

- **Popularity of developments**

Popular developments and having those developments being rated highly by users was a motivating factor. Examples of such allowed students to see that if they could develop popular products, then they too would get higher ratings.

Participant 5 stated:

The ones that like it so much, I always use an example of when you use a smartphone, you search for an application to do one, two, three, how many hits do you get? You get so many options, and you pay attention to the ratings of those apps. Have you thought about why is the other application rated 4.9 compared to the other one is rated 3.8 but both claim to do the same things? It's because the people who developed the 4.9 did their utmost best, the community out there is happy with their product. So, do you want to develop this one, the 4.9 or the 3.8? It's really up to you now.

- **Promote a continuous learning culture**

It was noteworthy to encourage students to continue learning even outside the classroom. This entailed Internet-based learning, as well as joining groups and forums that are specialised in certain areas.

As per participant 4, who supported the notion that students must evolve into “continuous learners” had the following to say:

So they must do lots of examples. I encourage them to read. I encourage them to go onto the Internet and just to extend themselves in certain areas of programming. I encourage them to basically look at groups, like OWASP they talk about software security. Basically, students can learn from these groups. There are groups that assist with hacking of systems. There is lots of interesting stuff there, and we encourage them to go and look at those websites and stuff like that to get more information and knowledge.

- **Advantages of programming post-university**

Highlighting the main advantages of learning programming and its benefits after university was a motivating factor for students.

- **Make coding simple**

One participant made coding simple, so students could understand it for themselves.

- **Achievable tasks**

By giving students achievable tasks, it served to build their morale and confidence.

- **Senior peers**

By bringing senior students to do systems demonstrations of what they have developed, and the prospects of employment served to encourage first- and second-year students to work harder and do the same.

- **Employment by top companies**

Students feel encouraged when lecturers inform them of top-ranked companies that seek to hire good programmers. An example of DERIVCO was made.

- **Make IT fun**

One participant made IT fun and enjoyable by introducing the **IOT (Internet of Things)** - an interactive platform that allowed students to use their skills on existing systems on the Internet and be able to make changes and tweaks.

Participant 7 was emphatic that an academic must make a lesson enjoyable and stated the following:

It has an operating system that runs it, you say okay let's get its open source, go tweak it then and upload it and make it do different things because all you want to do is generate an interest in students, say computers by themselves are not good until there's software that actually drives them if you can make a student make or maybe create a nice, pretty, animated something, whatever that may be it may be saying 'oh wow that's interesting' and then they begin to think about things beyond just that interesting exercise that they did in the classroom.

- **Feedback to learners**

Learner feedback was important to measure how feedback was disseminated. The following sub-themes were informed.

- **Feedback given regularly**

Two participants asserted that feedback is given regularly.

- **Classwork and tests**

Participant 6 gave feedback based on classwork and tests.

- **Dependant on syllabus**

Participant 8 felt that feedback was dependant on the syllabus.

And further stated that: “If I have a lot of syllabus that has to be finished, I may not take it weekly, I might take it two weeks”.

- **Solution based**

Participant 9 suggested that feedback to learners was in the form of solutions. The participant did not give marks but instead gave solutions.

#### ***5.5.1.5 Teaching support***

It was important to investigate if lecturers attained respective support for teaching. The following sub-themes generated from the results of the investigation are presented below.

- **Support – fellow academics, management, university of technology**

When it came to support from related entities, the following results were generated:

- **Internal colleagues**

There seemed to be substantial support from internal colleagues.

- **Support**

Collegial support was important and assisted when some lecturers fell behind on certain aspects of programming and programming languages. Furthermore, new concepts that needed clarification on almost any programming matter was possible with the assistance of colleagues and their opinions.

As per participant 5’s comment:

Well not from surrounding universities but from colleagues. I can’t function alone, I need my colleagues, especially if we have a larger group. Okay, even if I have my own smaller group, I still go next door to my colleague to get support in terms of something

maybe that's recently updated, and I fell behind on that thing. So, you will find that within the department we know each other in terms of who are doing programming at what level, who can I run to if I need a quick answer around maybe C# or Java.

Participant 10 agreed and added to the above statement: "Uh work colleagues we often get support, materials, advice uh if I'm unsure of a concept I'll go to one of the Profs for clarity uh we've got technical, some staff that are, have a good knowledge of technical aspects".

- **Teams**

Teamwork through colleagues was of great assistance.

- **Ideas**

New ideas were possible through teamwork.

Participant 10 explained a little further and stated: "Okay, I know that ... well, we basically formulated ... it was myself and two other team members, we came up with the whole mobile computing course".

- **Supportive meetings and discussions**

Meetings and discussions were excellent platforms to propose problems, issues and solutions.

Participant 4 stated the following with respect to team meetings:

In terms of fellow academics, we work in teams. So basically, whenever we have our team meetings, which is quite frequently, we have people discussing their problems during those team meetings, and we have people providing solutions during the team meetings. So, there is a lot of assistance from fellow academics.

- **Management**

Management aided in the following ways:

- **Training**

Management (HODs) did arrange different forms of training for staff development.

- **Tutors**

Management did aid in the form of tutors when lecturers needed assistance during lessons.

- **Lack of other ToTs support**

There seemed to be a total lack of support and collaboration from surrounding ToTs. The ToTs seemed to be working in silos with little to no interaction or collaboration.

- **Module descriptors**

A meeting is held among surround ToTs only when it came to building module descriptors where descriptors were issued.

- **Inviting resource personnel**

- **Not invited**

Most participants (6) did not invite resource personnel and some of them did not even know what/who resource personnel were.

- **Attempts made**

Only participant numbered 7 made an attempt to invite in a person who specialised in writing mobile computing apps.

- **Attendance in-service training –workshop in programming**

- **Does not attend**

Three participants asserted that they did not attend in-service training workshops for programming.

- **Scarcity**

Furthermore, four participants argued that there was a scarcity of in-service training workshops for programming.

- **Self-learning**

Two participants made efforts to learn new aspects of programming on their own.



- **Once a year**

Participant 8 asserted that most of the time courses are done annually whereby one course is organised in terms of programming. The participant was hesitant to divulge reasons for the lack of course attendance,

- **When available**

Participant 3 attended in-service training only when they were notified of such and when such training was available. The participant was also hesitant to divulge further details of reasons for non-availability of regular in-service training.

- **Conferences**

Only participant attended a conference to keep up with the latest research and developments in programming. The rest of the participants in the sample population were hesitant to divulge reasons for non-attendance at conferences.

### **5.5.2 Learning and Impact**

The themes were critical as it was important to examine current learning methods and their impact on performance. It was made up of the following key sub-themes:

- Background and knowledge;
- Learning methods;
- Group learning; and
- Performance.

#### ***5.5.2.1 Background and knowledge***

This section established the status of background knowledge and its importance in programming. It also examined if there was a difference in learner's abilities due to their backgrounds.

- **Previous – Basic Knowledge**

- **Basic knowledge**

The results clearly indicated that there was a **lack of prior basic knowledge** amongst students.

As per some of the participants (refer to dialogue):

Some of them come with absolutely no knowledge of even using the computer because that is not really a requirement for our students.

Most of them in the majority, they just don't have.

I don't think they have that knowledge.

- **Lack of IT at school**

There seemed to be a lack of IT exposure at school level. Some students could not correctly utilise a keyboard. One participant made an interesting point that in developed countries such as America, school learners were acutely exposed to IT and already starting businesses. However, in developing countries such as South Africa this was not the case and learners, especially those from disadvantaged communities, lacked exposure.

If you go to America, some of them already start companies because they can use their programming skills to develop applications. The tools are free there so conceptually they already know how to program, but in South Africa, I tend to think that maybe it's the opposite of that; students don't get exposed to what programming is, when I was in school many years ago nobody talked about computer programming.

- **Thought processes**

Another important point made was the issue of "thought processes" whereby IT orientated individuals are meant to have a specific thought process that is geared towards problem-solving and development. However, this was also lacking in current students.

• **Value of programming without previous knowledge**

When it came to the aspect of whether learning programming would be meaningful if it were not built on previous knowledge, it was found that programming is more meaningful based on the following factors:

- **Build on foundational concepts**

Programming needs to be built on previous knowledge to instil foundational concepts. Programming needs to be built on the foundational concepts to progress to more advanced levels.

- **Go down to their level**

Lecturers must be able to go down to the student's level to make programming understandable.

- **Treat equal as any language**

Programming should be treated as any other language at a school level. In developed countries, programming is taught as any other language such as English or French.

- **Subject background**

Apart from just IT exposure, other subject backgrounds are important. For example, someone with a maths and science background would be able to learn programming at a faster rate. However, most students were being admitted with maths literacy.

- **Coverage**

Due to a lack of previous knowledge, module coverage was progressing as fast as it should. More time needs to be spent for students to grasp the concepts.

- **Examples of IT value**

It was important to emphasise IT value in the world to build knowledge. Even though students lacked previous knowledge, current examples such as social media could be used to make them aware of programming value. This enabled them to identify with something that they already knew.

Participant 7 had the following to say regarding the afore mentioned statements:

Uh use things that they, they identify with to begin with, today with social media uh for example if I use Facebook because Facebook uses graph theory underneath to hook up all these different things if I were to say uh on your Facebook how many friends do you have? You'll say uh well I have a thousand friends uh how are they actually connected most of them tell you 'I don't know' but, if you say that there are links that link this student to that, to that, to that, to that so that by the time they look at graph theories

they'll say 'oh okay' so graph theory just is slap in your face right in front of you for doing something useful, so for them to introduce them in such a way that use things they already know.

Two participants had the following to say based on “dependency on individual” and “level of study”.

- **Depends on individual**

Some students were “built” for programming, and hence they could grasp concepts with or without previous knowledge. Hence it was based on the individual aptitude for programming.

- **Dependent – level of study**

Participant 2 felt that it was based on the level of study and not necessarily aptitude of the student. The next ensuing statements validated the response from participant 2, who went on to explain that by the second year of study IT students gained the maturity for IT programming and hence the aptitude for designing and developing complex programs.

▪ **First year – grounding of knowledge**

The first year was the most challenging as this was the grounding phase and hence learners without previous knowledge would be slower.

▪ **Second year builds on first year**

Post first year, from second year onwards, students built on knowledge gained in the first year. Hence the second year became easier.

But then again when you move to second year, like I always tell my second-year students, that never ever make a mistake of forgetting what you learned the previous year because you made sure in programming, for you to be sure you have to always remember what you learned so that you can improve on it and as we program further, we add up on those tools that we learned about previously like, I will give an example, in first year they taught them [inaudible 6:59] strategy statement. Nothing is stopping me at second year to give them a scenario that will challenge them on decision making structures but that does not mean I have to start decision making structures from scratch,

I can only remind them. So, it depends on the level of study in terms of whether they rely on the previous knowledge yes.

These examples show that each year is dependent on the previous. Just as how the second year is dependent on the first year. The first year is dependent on previous exposure at a school level.

Ultimately, results imply that while it is not necessary to have previous knowledge, it would be advantageous at the first-year level.

- **Approach to build on basic knowledge**

There was a plethora of approaches used to build on basic knowledge by lecturers. These included:

- **Continuous practice**

Ongoing practice on programming questions and exercises contributes to better understanding and progress.

- **Making algorithms applicable to life**

This enabled the student to see the applicability of IT programming in real-life situations.

- **Using different levels of examples**

It was always good to start with basic level examples to ensure that all students were at the introductory level before then escalating to more complex examples.

- **Knowledge sharing culture**

Creating a “knowledge sharing culture” by making the lecture environment more interactive where students can share their knowledge with each other. This allowed the stronger students to be able to give knowledge and ideas to weaker students.

Participant 5 stated:

I make sure that I create a culture of you know, working together, assisting each other. And the ones that know take pride in sharing their knowledge, you know, the fact that they get recognised as this person who knows better than us, they like that.

The point was emphasised with the following statement from participant 9:

And I make the ones that came with program language to understand that some of the things we get boring to them because they already know them, but I also get to ask them to assist me in terms of you know, helping the others now to get the basic knowledge across the classroom.

- **Getting to know students**

A crucial point was made by three participants: lecturers should get to know the students they are teaching. This will allow for proper assessing of how strong they were at programming and what type of previous exposure they have encountered. In addition, knowing their strengths and weakness will allow for how the class can be approached.

Statements from the three participants may be summarized as follows:

Well first we must assess what you are dealing with, you need to know the start point. That group of students that you have, what does the average start point. We got such a mixture of students, you've got those that are coming from private schools that will outshine even our third year-students with their current knowledge, and then you've got those that are coming from deep rural education systems, so you've got to find some sort of benchmark of what you're dealing with in the classroom, and then from there it depends on how much of the concept knowledge you want to cover. The principles of programming that need to be covered.

- **Preparation from basics**

Four participants concurred that you needed to start from the basics to ensure that all students could understand how to write programs. Disseminating complex terms and concepts at the beginning would confuse and intimidate the weaker students.

## **Social media**

A unique point was made by one participant, whereby they made students, irrespective of previous knowledge, be able to identify with current worldly examples such as social media. An example of Facebook, which uses “graph theory” was used to demonstrate to students how/why programming added so much value to the real world.

### ➤ **Difference – learners’ attentiveness abilities – background**

Investigating the difference between learner’s abilities and attentiveness with relation to their background revealed some interesting findings.

- **Different backgrounds**

A high number of participants agreed that different student backgrounds brought different factors to the classroom. There were some students from private schools and some from government schools. There were different quintile level students as well as various cultures in the classroom. All these factors impacted on learning. Hence critical assessment and review was essential to devise methods of teaching that could take all/most backgrounds’ context into account.

- **Language**

Relating to students’ background, came the issue of language.

- **English not first language**

Many students were not first-language English speakers, and this made understanding lectures that are delivered in English difficult.

- **Phrasing**

Relating to not understanding English, the phrasing of questions and exercises became difficult. Students may not understand it when questions are phrased due to their lack of understanding of English terms. The following statement emanated from a participant who considered English as his second language and related to this line of questioning to mother-tongue medium of

instruction. This was not part of the scope of the study, however, this has been included as follows:

But now the way you ask the questions, you know that was the main thing especially this year, they say the way we ask questions uh is very strange to them you know. If you, you see when you're doing a revision now in class for the exam seeing this question, this is what we wanted, they will say 'no you see if you ask this question like this then we will understand what you wanted to say', I mean wanted to ask, so I think there is a breach in terms of the language, or maybe the way we phrase questions you know all sorts of things."

- **Preparation versus non-preparation**

There are those learners who prepare (reading and doing their homework) before coming to class, as opposed to those who do not. Those who do prepare are the ones who are able to make more progress and understand programming better.

- **Lack of interest**

Some students just show a general lack of interest.

- **Lack of knowledge builds motivation**

An interesting key point from one participant was that some learners who lack previous knowledge show more enthusiasm to learn more. Hence this lack of knowledge builds motivation. The following statement was extracted from the interviews:

Because they may be, they are motivated to learn, and some of them just say 'Oh you know what I didn't expect it to be like this' and 'Oh it just doesn't interest me' but there are certain ones that want to see it work right and then, of course, you have those that did it in school and then now they feel like they don't have to attempt this exercise, they don't have to do this because they know it all.

- **No interaction**

It appears that in general, students in South Africa, especially those from disadvantaged backgrounds, tend to be more reserved in the classroom and do not interact with the lecturer or



other students in the classroom. In other countries such as America, students are vocal and will make their questions and ideas heard. In the current context, students are more passive absorbers with minimal to no interaction.

### **5.5.2.2 Learning methods**

It was important to examine the aspect of “learning” when it came to programming. Programming is a practical and applied subject. Learning methods become mandatory to evaluate.

#### **➤ Learning in isolation without social interaction**

- **Can be learned in isolation**

Almost 50 percent of the participants believed that you could learn programming in isolation.

- **Logic**

If one could understand the logic, then students would be able to learn and program on their own.

- **Determination**

Programming is also dependant on an individual’s determination. If they are determined to learn, then they would be able to on their own.

- **Cannot be learned in isolation**

However, the rest of the participants believed that it cannot be learnt in isolation for the following reasons.

- **Student -lecturer interaction**

If students were learning programming themselves, and reliant on online material, then there would be less student-lecturer engagement. Such engagement is necessary especially at the beginning and most students still need that human element with complex subjects.

- **First year**

It would be difficult, especially at first-year level.

- **Business operability**

It was important to know that programming is all about ensuring business operability through the programs that one builds. Hence this is dependent on social interaction to attain a good understanding of business needs.

- **Based on everyday life**

As related to the preceding point, apart from just business, programming needs to be applied to everyday life and hence the need to understand the dynamics of everyday life. This requires social interaction.

- **Learners to explore real work-related scenarios**

When it came to learners exploring real work scenarios, the following was found. Overall, results show that learners were not entirely being exposed to real work scenarios.

- **Limited opportunities**

There did not seem to be many opportunities for such exercises. However, lecturers tried to incorporate work scenarios into class exercises.

- **Not enough time**

Relating to above, there was also not enough time for such engagement

- **Third year level**

Work related learning was only done at third year level after students were experienced with programming.

- **Ethical issues**

Due to ethical issues at some organisations, it was difficult to send students to examine work-related scenarios.

- **Dependant on modules**

One participant asserted that it was dependant on modules.

- **Modules 2A and 2B – class-based**

These modules were class-based, hence no work-related scenarios.

- **Project-based is scenario related**

The project-based programming modules included learners finding scenarios to solve based on the knowledge that they obtained from modules 2A and 2B.

- **Must see how application works outside**

One participant argued that exposure to the outside world was very important to enable students to see how their work is applied to business and society. Building programming was not just about learning code, and concepts, such as syntax, through a book. It is important for students to see how programming drives the world with them being the engineers behind it.

Participant 8 had the following to say:

Yes uh case studies are very important uh you may, maybe watch a video and respond to what they saw or go online and read somebody's blog or maybe an introduction to something new which happens all the time read about it, again when I teach them to, to write code I don't want to teach them in isolation that okay it's me and the computer I write this code, you have to also interact because the code that you write you're writing it for other people actually use so if you look at how people use the applications that we actually build then it gives you a better sense because (oh) the other thing that's important, if I develop an application and put it out for somebody to use, I don't, I want my students to think about how the person is actually going to use that application, because if they struggle it tells me that I didn't code it correctly or if it generates an error, or it generates an error it looks very cryptic to the end user so that's why a fault, a problem, so I want them to go out there and actually observe uh, again I'm just labouring the point, if you build something you put it out there, so I used to challenge them, so you, run this application that you just built for me and then I say do this, go in here and enter 12345, xxxx, kkkk, enter, see what happens the application crashes and then I say 'Alright what do you have to say about that?' so you have to make them think.

- **Developing Learner Interest**

Results show that lecturers do engage in a variety of ways to develop learners' interest. These included:

- **Real-life examples**

Making examples applicable to real-life was most effective. This enabled student to identify better with the examples.

▪ **Make it relevant to reality – employment**

Students are primarily concerned about their employability post studies. Hence, by showing them how much programming can enhance their employability, they are motivated to learn more.

With respect to employability participant 7 stated:

I think for a student that's here; the primary interest is employment. So, you want to spark an interest in a technology that is relevant to the industry, economy at that point in time, so that's one and then two, you want to do sometimes maybe in the form of assignments or group tutorials or etcetera something that is relevant. So yes, you're teaching programming principles but the product must be something from which they can see value, you know a textbook exercise that solves a very intense problem puts them, puts 'blinkers' so to speak on the students but if you, if you equate it or situate it such that it talks to their reality then it makes more sense, it becomes slightly more enjoyable.

▪ **Daily life and localising examples**

Localising examples and making them applicable to daily life in the South African context allowed the students to identify more with their context and how well programming fits into it.

Participant 9 stated the following:

well I think uh in most cases real-life examples you know uh you can simulate a student. You just say if you got a store, you want to sell this, whatever you want to sell you know and you want a simple thing, a nice simple tool or a simple software that you can use that can help you to sell whatever you want to sell you know in your tuck-shop then you would develop something like a computer system that can help you to do the selling. In that case, you don't need to use your mind to calculate you know you just

take the thing and put it in the computer and then the computer will calculate. So, they become you know a bit interested. I would like to use that.

- **Freedom of choice**

Offering students freedom of choice enabled them to choose examples where their strengths were. These included:

- Business
- Mathematical
- Social

Hence students could relate to examples relevant to what they identified with.

- **More practical exposure versus pen and paper**

Programming was more effective if taught on the computer as opposed to pen and paper. On the computer, the student would be able to see the value of their work and how the program looks, feels and runs. They would also be able to identify errors in their work. This was not the case if they wrote out the program using pen and paper.

- **Teaching value of programming in the world**

Continuously showing students the value of programming in the outside world, by using worldly examples also played a role in motivating them to become good programmers.

- **Strong involvement with subject**

By engaging students strongly with the subject allowed for knowing why different solutions applied to different problems.

- **Working on their own**

Allowing students to figure out problems on their own was also a way to stimulate their minds to programming.

- **Based on individual interest**

One participant asserted that programming was also based on the learner's individual interests. Some learners have the passion and drive for it. Learners with an interest in programming

needed to be harnessed. However, more work was needed with those who did not show an interest, and in such cases, despite teaching, it was a matter of whether the student could grasp programming or not.

### **5.5.2.3 Group learning**

It was important to discover if group learning influenced learning when it came to IT programming. The following sub-themes influenced this theme.

#### **➤ Programming lesson when conducted in groups**

It was important to determine if the learning of programming can be successful when conducted in groups. However, the following factors provided a mixed reaction.

- **Learning from each other**

Three participants supported group learning and concurred that students learn from each other, especially those whose first language was not English.

- **Collaborative**

In addition, group learning was very interactive and collaborative which stimulated learning.

- **Motivating**

Individually, some students lacked the motivation to learn programming, but when put in groups, they would motivate each other.

- **Different approaches**

Different approaches are used when it comes to group learning, and this is based on the real-world scenario with each having its own method of team organisation.

- **Project manager approach**

Project manager approach needed a defined leader/manager.

Participant 9 was employed as a project manager and hence mentioned the ‘project manager approach’ and ‘agile self-organising’. This was stated:

chief programmer approach uh project manager approach. There's different ways in which you would organise your team so if you're looking at first-year students, which organisation will be best to promote learning of programming principles.

- **Agile – self organising**

Agile methods focused more on self-organising with an emphasis on strengths and weaknesses.

Participant 9 stated:

So you want an organisation where someone must not take the lead, so you won't have a lead programmer as such because you want an open playing field. At the same time on the other side of the coin, agile talks about self-organising or autonomous teams, so that means given a task the team know their own strengths and weaknesses having worked with each other for a long time. So, they know, you do this, you do that, you do that, they know how to organise themselves, so they can turn out the work in the most efficient manner.

- **Should not be group based**

However, a considerable number of participants did not support group-based learning for the following reasons.

- **Better measurement of students individually**

It was better to evaluate a student individually to know their strengths and weaknesses.

- **Group work dilutes engagement**

Group work dilutes the engagement process as everyone talks simultaneously, and it makes it difficult to determine knowledge levels of all students.

- **Politics**

There are certain politics in groups, where students form cliques and some students do not make a positive contribution to the task.

- **Not at lower levels**

At lower levels of study, weaker students will always be overpowered.

- **Personalities**

Personality traits of students also differed which made it difficult to work in groups.

**Type A – dominate**

Type A personalities are the more dominant personalities, and hence they can dominate the group leaving the weaker and more reserved students behind.

Participant 6 displayed ‘A type characteristics’ and stated the following:

Uh it depends again, if I’m looking at first-year students group work wouldn’t work very well part of the problem is that we have different personalities, some students will be the A type of personalities where they’ll just take over and the other reserved students who rarely tend to have an opinion then uh those students are going to be left behind because the ones with type “A Personalities” will just run all over all of them,

▪ **Psychology of teams**

There are different psychological and social levels of students considering that this may be the first time they are interacting in a group. The psychology of students is different, and hence they may respond differently.

Participant 6 continued the discussion regarding team work and stated the following:

Unfortunately, group work can be dangerous in a first-year team. One, as a lecturer not all lecturers have a good understanding of, of the psychological and the social undercurrents of bringing people together. So, we need criteria that must be carefully considered before you bring together a grouping of people and expect them to produce something efficiently. It’s compounded in a first-year lot because even they have never been together, they haven’t yet developed social relationships, friendships amongst themselves so they complete strangers, now whether that’s a good thing or a bad thing



I can't comment because I don't understand the social uh, the psychological things in play there.

- **Reserved students left behind**

Due to cultural backgrounds and other factors, some students are reserved by nature. Hence such students can be left behind if the group progresses too fast.

Participant 8 concurred with participant 6 and stated the following:

especially in our culture here we have a lot of that students tend to be reserved so if you put them in a group they will struggle but if I'm looking at third years or senior students I would assume that they have enough background information so if you put them in a group, again as a teacher I've done this is to look at them, everybody has to have a task, you do this, you do this, you do this, I want to say you guys do this build this and then I sit back and watch because again the type A personalities are going to go home and write this whole thing, come back the next day and say well it's done so the other people don't get a chance to actually participate so if you say group, everybody is responsible for a part of the entire project.

- **Observations during group discussion lessons**

Below are some the observations found in the study data with relation to group discussions amongst students.

- **Active involvement**

There seems to be active involvement amongst students in the groups.

- **Monitor contributions**

Contributions needed to be monitored as so that all group members participate equally.

- **Shared commonalities and goals**

Everyone has a different method/way of finding a solution. However, the goal remains the same, and that is to solve the problem. Hence group work allows students to discuss their ideas whilst understanding the goals.

- **Place stronger students in weaker groups for knowledge exchange**

Sometimes students like to band together and form their own groups and you may find that one group has an abundance of strong programmers and this does not help the weaker students. It was important to place at least one strong student amongst weaker students to ensure that learning takes place.

- **Stepping in at a practical level**

Sometimes, lecturers needed to step in at a practical level to guide the groups in their tasks.

- **Identify strengths and weaknesses**

It is important to walk around and be actively involved in identifying strengths and weaknesses of the group so future approaches can be informed.

- **Learn from friends/peers**

A student can learn from peers, especially those who are shy and unable to converse with the lecturer.

- **Some stronger students dominate the group**

However, four participants asserted, as mentioned before, that stronger students end up dominating the group. While this could help with problem-solving, weaker students get left behind.

- **Interaction with peers- affect learning**

Only four participants responded to this question, and results imply that interaction with peers did affect learning in the following ways.

- **Exciting inventions**

Referring examples to worldly digital innovations contributes in facilitating some exciting inventions and ideas. This motivates students to program and develop such inventions.

Participant 9 had the following to say:

Yeah, I think if you do something exciting where they say, ‘oh wow!’ or maybe teach them I don’t know like somebody said I’m going to be using this digital assistance where you can talk to a device if you don’t want to talk there’s Alexa over there, Amazon and Echo. I can write, I want to teach them how to write the Amazon skills, let’s call it skills, short snippet of code you’ll upload and then I can talk to the thing, back and forth, back and forth, if they can say oh wow that’s how I can make this thing do something that’s when I can say, well how do I use this, end users in a house where somebody is maybe is disabled because this thing can be hooked up to ten with the lights are on, I can say let’s turn the lights on, poof!

- **Supervision needed**

While supervision has a positive influence, it is still needed as students tend to drift off the topic and it would be difficult to establish what they are teaching each other.

- **Knowledge sharing and learning**

As mentioned earlier, positive influence happens in knowledge sharing from stronger to weaker students, and this promotes learning.

- **Allow group discussion during programming lessons**

When it came to lecturers allowing group discussion during programming lessons, the following was found:

- **Not often** – as asserted by one participant.
- **Lab session** – the lab session entailed group activity.
- **Frequently** –

- **All/most of the time** – as asserted by three participants that they allowed group discussion as much as possible.
- **Tutorial classes** – two participants asserted that they allowed group discussions during practical and tutorial lessons.
- **Twice a week** – one participant allowed group discussions twice a week.

- **Not recommended**

One participant rejected group work as it tends to become very **noisy**.

#### ***5.5.2.4 Performance***

Learning would not be able to be measured without performance. Hence examining performance was pivotal to determine the effectiveness of programming.

#### ➤ **Learner performance**

Learner performance ratings were varied between participants. However, if one looks at it holistically, results indicated average to weak performance.

- **Average**

Ninety percent of the participants rated the performance of their students as average. Their non-verbal cues indicated a despondency in their students' ability to attain exceptional programming skills

- **Below average**

Participant 3 stated the following reason for the below average performance which included mistakes and loopholes:

Picking so many loopholes, picking so much mistakes, there are things because I'm in second year, there are things I expected them to know very well before coming into second year.

- **Weak**

Participant 9 felt performance was weak as test results were bad especially at the beginning, but in general it improved as students matured.

- **Based on Aptitude**

Again, students who had the aptitude for programming performed better than those who did not. Hence from second year onwards, it became easier to identify who had the aptitude and who did not.

- **Higher level – good**

One participant asserted that students at higher levels were more experienced due to their prior years at programming and hence performed better.

The causes for current weak performance were listed as the following.

- **Wrong course selection**

Some students may have made the wrong course selection and were unaware of what programming was about.

- **Unaware of advantages**

Students are unaware of the advantage of the first year.

- **Software versions**

Software versions are changing continuously, and textbooks are not able to keep up. Hence students may learn from the textbook but could be working on different software versions with different features. This then causes confusion and impacts performance.

- **Part-timers**

Some part-time students de-register and return years later and are hence unable to initially adapt to programming.

- **Lack of prior knowledge – school level**

The lack of programming at school level contributes to students lacking prior knowledge which then makes it difficult for them to grasp programming and its concepts.

- **Different teaching styles**

Different lecturers teach differently. Hence, when students become accustomed to a certain way, it becomes difficult for them to adapt to another lecturer's way of teaching.

- **Second year**

Even second-year students were not performing as well as expected.

- **Forgetful of prior knowledge**

Students who are post first year appear to have forgotten first-year knowledge. First-year knowledge is important as all other years build on that. Forgetting the first-year work slows the students down at the second-year level and beyond.

- **Different levels of learning**

Furthermore, students did not realise that they were now in the second year and that things were getting more complex and different. In addition, assessments were also becoming different and more intense. Different levels of study brought with it different levels of learning. Students did not take this into account.

- **Confusion of languages**

Students were confusing programming languages such as C and C++ and Java and JavaScript.

- **Concepts**

Students were continuously grappling with programming concepts.

- **Reasons for good performance**

Good performance was attributed to:

- **Tutorials**

Tutorials were being enforced and those that took the time and effort to complete them performed well.

- **Second year brings filtered population**

The first year included a diverse population of students, and it was difficult to determine who had a passion for programming or not. However, by the time they reached the second year, most of the time, those who did not share such a drive for programming either failed or dropped out and those who did make it into the second year were those who had the drive and passion for it and were able to cope and do well.

Participant taught students across various year groups and had this to say:

I think when they come to second year, they are obviously weeded off from first year. So now I can say that programming is for certain people, it's not for others. So, I think in the first year the ones that really can't handle, they either drop off, or they fail. So, you're left with the better lot. I wouldn't say the cream of the crop, but there's somehow the better lot that were able to progress from first year to second year.

- **Satisfaction with learner's performance – tasks**

All participants expressed that they were **not satisfied** with learner performance of the assigned tasks. The following informed this finding.

- **Based on attendance**

Most participants asserted that performance was based on attendance. Those who attended lectures regularly performed better than those that did not.

- **Class size**

Class sizes were relatively large thus causing a problem for proper engagement. This seemed to be a national problem.

- **Copy and paste**

Plagiarising the work of other students was clear and evident.

- **Excuses for not doing tasks**

Students made excuses for not doing tasks. Stolen or forgotten USBs were among the excuses.

- **Lack of interest**

Some students just did not show the right interest in programming. Without such interest, there could be no progress.

- **Starting challenges**

Students have great difficulty in achieving tasks at the beginning.

- **Social factors**

Factors such as social and family issues and a lack of focus on studies seemed to be evident in students.

- **Task not completed**

Tasks were not completed, and the completion rate was not good

- **Promoting improvement**

The following was discovered to try and promote the improvement of performance of tasks.

- **Ensuring the same level before progressing**

It was important to ensure that all students were at the same level before progressing with the lecture/modules. Moving to more complex aspects is not recommended if most of the class is still struggling. Hence ensuring that all were on similar levels would promote better performance.

This point was discussed by more than one participant, however participant 10 stated the following and the quotation is as follows:

And I need to practice what I preach in terms of building up on your knowledge. There is no point for me if I can see most of them for example, struggling with unit testing or still struggling with their modules in their MVC application, it's not fair to move on to databases now because how do you bring another concept that needs the knowledge around models for it to function? So just like now, as I said we gave them the first test, I can't move on to other concepts until I cover those weaknesses that I saw. So, it's my way of giving them feedback and making sure we're all on the same level of thinking and understanding before I move onto the next concept.



- **Remediate and Re-test**

Students need to be remediated on where they went wrong, and then need to be re-tested on similar types of examples.

- **Versions of code to avoid copy and paste**

Different versions of the code can prevent copy and pasting amongst students.

- **Extra time**

If students could explain their areas of difficulty, then some lecturers would make allowance to spend extra time with them on those areas.

- **Partnering- other students**

Partnering weaker students with those who were stronger in programming was a means of promoting improvement. Sometimes students felt comfortable and were able to identify better with their peers as opposed to lecturers.

➤ **Impact of programming on learners' interest, enthusiasm and motivation – low achievers**

When it came to motivate low achievers through the impact of programming, the following factors were determined:

- **Driven by aptitude**

Aptitude for programming was key. Those with strong aptitudes were naturally motivated. However, those with minimal aptitude only aimed for average marks such as 50 percent.

- **Driven by the IT world**

Due to the world being so IT-driven and orientated, some students acknowledged this, and it gave them the motivation to be a part of this.

- **Full understanding of IT**

Once students understood what IT was about and grounded themselves in that understanding; then they became motivated to learn more and perform better at programming.

Participant 8 stated the following:

Okay, I think once they finally understand what IT is and what their role in IT is and what their whole purpose of doing an IT Diploma is, once they have that as a grounded understanding...because many of them don't know what this application development is all about. They think it's about we're using Word, Excel, PowerPoint. So, I think once they get that proper understanding then yes that would obviously help them there.

- **Dropout rates**

Currently, the 40 percent dropout rate for low achievers at the first-year level is a problem.

- **Multi-factored**

The dropout rate for IT was multi-factored, and it came down to factors that influenced a student's performance. This included whether:

- The student had the ability to get into IT;
- The student had the drive for programming;
- IT was the first or last choice on their application form, and they only chose it because there was nothing else they could get into; and
- Financial and social reasons.

### **5.5.3 Challenges**

Challenges were grouped in the following sub-themes:

5.5.3.1 Background and diversity

5.5.3.2 Student ability and readiness

5.5.3.3 Lecture and classroom

Each is unpacked in the sections that follow.

#### ***5.5.3.1 Background and diversity***

These challenges are related to the students' background and diversity.

- **Student diversity**

The diversity of the students presented a challenge in the sense that most of them were never exposed to computers, especially those from disadvantaged backgrounds.

- **Disadvantaged background**

Arising from the previous point, most students from disadvantaged backgrounds had never seen a computer before and certainly had no exposure to programming. This meant that they did not know anything about computers holistically and this therefore presented a challenge.

- **Weak background**

Relating to the above, students had a weak scholarly academic background, and the lack of **maths** made it difficult for them to solve problems in programming examples.

#### *5.5.3.2 Student ability and readiness*

These challenges pertain to the student's ability and readiness for programming.

- **Resources and restrictions**

Students did not seem ready for programming, and this was informed by the following factors. The schools that students attended, restricted students' access to computers due to a lack of resources and this led to:

- **Lack of exposure**

Lack of previous exposure created barriers to learning programming.

- **Limited background in programming**

The limited background in programming was built on the lack of exposure. This again hindered progress in terms of programming.

- **Scenario interpretation**

Students just couldn't interpret problem/examples/question scenarios. The **logic** aspect presented difficulties to them.

- **Critical thinking for problem-solving**

Participant 7 stated rather emphatically that students seemed to lack critical thinking ability. Even if you helped them interpret a scenario, they were not able to apply their minds and program the solution. Critical thinking is not something that can be taught. Students need to have that instinctive ability to be able to problem solve by applying their minds and thinking out of the box. Programming is built on the ability to think critically.

As participant 7:

Then let's say maybe you help them to understand the scenario, then you move to another set of challenges like your, how to program solve now, to critically think, you know. I find that problem-solving, and critical thinking is the biggest challenge in terms of teaching them how to do that because it's not something that you can only teach from a textbook, it's also something that comes naturally or personally. It depends so much on personal factors than you just saying critical thinker does this does that, does that, but what about the person you expect to be a critical thinker, a problem solver, so those are basically the challenges.

- **Problem-solving**

Students were unable to understand a problem and solve it accordingly by breaking it down into sub-problems.

- **Rote-learning**

Students were rote-learning solutions to programming problems. Programming is not a rote learning subject but more an application of knowledge. Hence, when the problem scenarios were changed, then the student battled to apply their memorised knowledge.

- **Self-learning**

Students were not learning out of the classroom. When the lecture ended, so did the learning. More self-learning is needed as programming requires such.

- **Programming – skills-based rather than knowledge-based**

Participant 6 made an interesting point whereby programming was not a knowledge-based subject but more a skills-based subject. One needed to have such skills which could then be

developed and harnessed by the lecturers. Programming could not be “taught”, it is an intrinsic ability to be able to program.

Participant 6 stated:

Uh second, is the ... the... the perspective of programming, Programming is not a knowledge-based subject, it's more linked to a skill. So, you can't teach programming or deliver a lecture for example and then expect some sort of development to occur, excuse the pun though uhm instead we should gear the programming teaching, learning and assessment, so it's more aligned with developing the skills. The same way you would train an Olympic swimmer, you can't put him in a classroom and deliver a lecture and expect results, but you need him in the pool uhm identifying his, the challenges that are unique to him and then helping that individual overcome that unique challenge. Pretty much how a coach would train an Olympic swimmer.”

- **Lack of programming interest**

Some students just didn't have an interest in programming.

- **Adaptation to programming language**

Students experienced difficulties adapting to programming languages and all the concepts that came with them, such as syntax and generally how the program works.

### ***5.6.3.3 Lecture and classroom***

The following challenges were from a lecturer/classroom perspective.

- **Resources**

Resource constraint seemed to be a highly ranked challenge in the form of:

- **Teaching materials**

There were inadequate teaching materials beyond the classroom to teach programming.

- **Population per lecture**

Lecture venues were crowded where, for example, a lab would have 60 to 65 people with an hour to teach. This meant that the time: population ratio is compromised for personal attention and time to deliver the lecture. Hence this becomes a challenge in developing programming skills.

• **Teaching skills**

Teaching methods need to be revised to develop programming skills to the current generation. Programming is different from other subjects, and it requires a different teaching methodology/ pedagogy. Hence new teaching skills are needed by lecturers.

Participant 5 had the following to say regarding Teaching Learning Assessment (TLA) of programming:

What about, the other challenge is also within the TLA ambit, but it deals with the ability and the approach adopted by the staff. You need a very, very different uh TLA you call it a “Teaching Pedagogy” uhm to facilitate developing a skill compared to doing a history lecture. That’s a very very different skill set that’s required by the lecturer.

• **Teaching Learning and Assessment reassessment**

Building on the above point, the entire TLA needs rethinking for programming languages. An example was made to Massachusetts Institute of Technology (MIT) and how both physical, human and technological resources in and out of the classroom were part of their TLA strategy for programming quality.

Participant 5 further stated:

For example, MIT programming in the first year is a class size of almost a million students, and there’s one Prof and maybe eight assistants that run around with the Prof. How do they do it? And really MIT wouldn’t do any practice to compromise quality because they are you know one of the leading IT institutions in the world MIT. So how do they do it? It goes well beyond classrooms, uh having students physically in front of

you, it goes beyond technologies, it's a whole new, different outlook on the way we think about teaching programming?

- **Un-complicating programming**

Student's need not fear programming, but the challenges of how to make programming not appear complex and foreign remain. Lecturers needed to apply their minds, as well as determine ways to simplify programming, for example, by relating it to everyday things that students do.

- **Communication**

One participant asserted that communication was a challenge especially due to the diversity of students and English not being their first language. In addition, the same participant asserted that the main challenge pertained to their accent. Students didn't understand the participant's accent, and so the participant tried to reduce her tempo to improve communication in class.

#### ***5.6.3.4 Overcoming challenges***

In relation to overcoming challenges, the following sub-themes emerged.

- Support;
- Teaching and learning; and
- Communication and interaction.

Each is unpacked in the sections to follow.

#### **➤ Support**

Challenges can be alleviated if the following support factors are considered:

- **Tools**

Tools such as input/output and flowcharts allow for students to be guided during programming lessons.

- **Early identification of at-risk students**

If "at-risk" students can be identified early and this information is communicated to the respective lecturers, then additional remedial work can be assigned to them.

- **Additional tutors**

Tutor programmes can be run whereby more tutors can be used to assist students who are experiencing difficulties.

- **Bridging course**

It is recommended that a bridging course be implemented before students start programming. This will allow for more effective preparation.

- **Teaching and learning**

When it came to actual teaching and learning, the following is a possible method to overcome challenges.

- **More problem-solving**

More exercises and problem-solving can assist in building programming skills.

- **Teaching style**

Seeing that no two classes are the same, it is recommended that TLA should be adapted to suit the class being taught.

- **Visual programming**

Using techniques such as visual programming, where students can see what they are building through code would motivate them further. This would help them understand the logic and flow of the program by visually seeing what needs to be done.

Participant 8 stated the following with respect to visual programming:

Uh by using maybe visual programming helps a lot because in the past when that technology wasn't uh that great to where you could use uh maybe things like you'd dragged, and uh like if you're using Scratch. In Scratch, you use objects that students could join and then they can take a pick to see the code that gets generated. In the past you used to show them codes without the visual, so they didn't understand like if you say here's a variable, what's a variable but if I say here's this box where I'm going to store something let's give it a name, we give it a name, by the way, that name is called a variable. So, if you wish you may use visuals to first-year students just to get a concept



of I guess the flow of the program, the logic of uh how it gets executed, uh even if you're using loops, you're using conditional statements and all of those.

- **Simple exercises**

Starting with simple exercises would expedite learning and then slowly moving to combined and complex examples once students were more comfortable with the simple ones.

- **Pseudo-code**

Pseudo-code assists in helping learners to solve problems because it is done through normal English and not programming language. Only after it is fully understood, it is then converted to a programming language. Hence pseudo-code is useful in helping students to understand programming.

- **Keeping abreast of technology**

The lecturer must keep abreast with all the technologies they are teaching to ensure that they are teaching the most up-to-date versions.

- **Open book tests**

Open book tests will prevent students from learning by rote, and in turn, allow them to apply their knowledge and focus on the logic of the program.

- **Communicating slower**

Communicating at a slower pace helps students to understand better when English is not their first language

- **Diversify tasks**

Tasks should be diversified to ensure that all students, whether weak or strong in programming, are able to identify and understand the lesson, exercise and problems.

Participant 6 supported diversifying tasks and stated:

Sometimes the students that come in and do know it all, don't know it all, so you start from the very basics, but the only way that you can accommodate them in one classroom is to diversify what they need to do the tasks. There must be some tasks that

are very easy to do, and there are some tasks that are very challenging, otherwise, you lose the student.

- **Skills-based**

As mentioned before, programming is a skill-based subject, therefore, if lecturers are fully aware of that and in turn, understand that each student will have unique challenges, then they can find other ways of dealing with such challenges.

- **Communication and interaction**

Communication and interaction can also bring out a means to alleviate challenges in the following ways.

- **Engaging with senior students**

First-year students are encouraged to engage with senior students such as those who are in second or third year so they can be assisted with difficulties and be exposed to more advanced programming topics to acquaint them with what is expected of them at that level.

- **Group interaction in class**

Group work in class assists student to come together and share ideas and be able to solve problems together. This also allows students to learn from each other.

- **Language outside the classroom for learning**

Seeing that most students are not English first-language students, some lecturers do allow students to use their main language of communication when discussing lecture content, so they can identify better with the concepts. The lecture reverts to English content after the concepts are understood.

#### **5.5.4 Recommendations to lecturers teaching programming**

The following recommendations are made with relation to lecturers teaching programming for the present and the future. These are grouped in the following sub-themes.

*5.6.4.1 Teaching*

*5.6.4.2 Students*

*5.6.4.3 Resources and support*

#### 5.5.4.1 Teaching

This was a core theme in the research and the following discussion serves as teaching recommendations for prospective computer science lecturers.

- **Good preparation**

This was a highly ranked recommendation where lecturers needed to be well prepared before class. They needed to master the content, as well as understand the nature of the class/students. Lecturers could not afford to make mistakes in class.

- **Application of theory**

Application of the knowledge in a practical setting, apart from just teaching, is important. Hence lecturers needed to ensure that students were taught applicability and given opportunities to do practical examples in the computer laboratories.

- **Teaching skills**

Teaching programming is a skill as it is not like any other knowledge-based subject. Hence how to teach such a skill is important. It's not just about passing on knowledge.

- **Promote self-learning**

Lecturers must encourage students to continue learning outside the classroom. Learning should not end when the lecture is over. Programming is a concept that is ever changing, and hence ongoing self-learning is mandatory.

Lecturers need to treat the class as a **community of practice** where self-learning is encouraged, facilitated and developed.

- **Modernise examples**

Lecturers also need to ensure that they are keeping up with the latest applicable examples. In the current epoch, technology dominates the world, and it is critical to keep up to technology. You cannot be giving students old examples to work with. In addition, you needed to stimulate students to think of modern ways to apply programming.

Participant 7 said the following:

the other thing that I do in the classroom I try not to make the material so dry that it's about this chapter and nothing else because you'll lose them what I would do is put in some adopted stories around these things like an example I used, because for me when I drive around I look at things and say 'oh wow that'll be interesting to students because it's a problem'. The other day I was talking to a colleague, I was at a gas station and saw them using these dipsticks to take readings, I guess so they know how much gasoline is in the tanks underground, and I said that is old nobody does that anymore, I take my camera I take a picture I said this would be interesting to point out to students so I that if I go into the classroom I said, I would say put it up and say 'What's wrong with that? What are these people doing?' maybe I would assume they have seen it or I would say maybe management at the end of the day wants to see how much gasoline is in the tank so they use a dipstick and do all of this I'll say no, there's sensors over here that can tell you all of that you don't need to do that so then for them I'll say now, let's go solve this problem so they say management at that gas station calls and says I know you're studying this IT, it means people are still dipping these things, how can you modernize this?

- **Facilitate development processes**

It is important for lecturers to facilitate development processes as students come from a variety of socio-economic factored backgrounds. Especially when it comes to group work, it is important to ensure that students are grouped well, taking into account factors that facilitate that development. Furthermore, facilitating the development of skills is necessary.

Participant 10 stated:

You will have groupings because now people start to become familiar with one another, learning styles, economic backgrounds, uh social standings and they group themselves according to that and you will, we call it friends, a group of friends. This happens all the time, and these people come together in your class, you need to facilitate that process.

Another participant concurred with participant 10:

Let that process happen as well, and you also have to pay in mind that when you're developing a skill or rather, the correct term is you are facilitating the development of the skill. You can't teach swimming by giving an hour lecture or sitting in an hour lecture or listening to a podcast or a YouTube video. You learn to swim by getting into the pool.

- **Creativity**

Creativity is essential for a subject such as programming. Students need to be stimulated in a creative way and not the traditional book-only way. Hence lecturers need to make the subject creative and enjoyable by using application examples such as gaming and other innovative ways to spark interest.

A participant who enjoyed gaming stated:

In terms of programming to encourage interest in programming, there is a lot of gaming stuff that you can use. Where students can, by play, they can learn. So, you can use those kinds of applications to teach programming where it will spark students' interest. So, look for those kinds of applications. There is stuff out there where you can teach programming in a fun way. Where you can use a novel way of introducing programming by introducing these applications to students. So, it will spark a lot of interest in the students.

- **Bi-directional**

Programming needs a multi-modal type of learning, and hence interaction must be bi-directional and not just uni-directional.

- **Novice teachers have the best abilities**

One of the younger participants made an interesting point where novice lecturers may have the best abilities at teaching because they were recently students themselves and could identify with the current students whom they teach. They can understand the mistakes that students are prone to make, unlike the older and experienced lecturers who are so acquainted with the

subject that they often take certain things for granted and forget the mistakes that students make.

This younger participant stated:

I always, that's why I have always no problem when I've been given like new courses or something of the sort. Why, because in a way, I'm learning it, they're learning it, I know exactly where they're going to have problems and I can be able to envision those problems and be able to help them but for a person that's already done it like lots and lots and lots of times, it's like, you take certain things for granted, you forget certain mistakes that these students would make and uh and then of course you know, I don't know what, I know when I first started.

#### **5.5.4.2 Students**

Recommendations from a student perspective entail:

- **Understanding students**

The majority of the participants concurred that making efforts to understand students was crucial to the teaching and learning of computer programming. This would enable lecturers to pitch their lectures according to the students' abilities. In addition, this was important in building relationships with students and contributed to their success.

Consequently, if you have students who are experiencing difficulty with programming, you have to understand their socio-economic background and use this understanding as an opportunity to devise a suitable approach to your lesson.

- **Different backgrounds**

It was important to know that students come from different backgrounds to be able to understand them. The students are not the same and you need to apply different approaches and strategies when lecturing. It was important for lecturers to know this at the outset and prepare accordingly.

Participant 4 stated:

So for me, I would advise somebody to really get to know what they're dealing with in terms of the students. We just spoke about their differences, the students, they come from different backgrounds, so there's no point for me to treat them all the same whereas that's not a reality. They're not the same. So if you start acknowledging that, it's not like I'm saying you need to know each one of them, but you can classify them to say I have a group of this nature of students, a group of this nature of students. Therefore, I need to prepare this way; I need to strategize this way in terms of approaching my lessons; I need to use my students to teach each other to facilitate my lessons, you know, so those are one of the things which I believe are working very well.

- **Student-centeredness**

Student-centeredness was also an important factor when dealing with students. After all, being an academic is about teaching and building knowledge for students. Being student-centred entailed the following:

▪ **Understand how students think and learn**

It was important to get to know students and the behavioural patterns in terms of how they socialise, think, and learn. By getting into that space of students, it would make it easier to identify with them and devise plans to enhance lessons accordingly.

Participant 7 referred to “understanding how students think and learn as student-centeredness and had the following to say about it:

You see there is this thing now called, student-centeredness, I think you see it took me long to understand this whole student-centeredness thing. I would think that would be maybe the first to go, because you see sometimes if you understand how the students think and how they socialise, I think it would be good to just get into that space and try and understand how they understand things, understand how they learn you know and like I said you know we only realise late as to how we should ask the question because I'm sure they students would pass if we had to ask the questions the way they understand you know so well that would be my advice that you know just into their space, understand how they socialise, understand how they learn the language how they

understand it and then I think it will be easier for them, even for you to pass this course so that they can give you the answers that you are expecting, well and also you learn in different uh what different styles you know uh because you don't know what type of student will come to you and also the students are not the same and also the way they learn is not the same so uh I think just to observe them, see how they do things then you can try to adjust to you know how they learn. I think that's the whole concept of student-centeredness.

- **Be on their level**

It was important to ensure that you are at the level of the student. Sometimes the traditional way of covering chapters per lecture would not work as some would fall behind. Hence by making more effort to know which students are ahead and who is falling behind would ensure improved lecture preparation/delivery and ultimately greater student understanding.

As per one participant:

My advice when starting of: be prepared to begin with, at the same time be prepared and don't try to sugar on everything to students especially let's say first-years they are coming in they have never done programming before so it's going to be difficult to let's say okay today maybe in the syllabus it says we are teaching chapters one, two and three and then you want to go one, two and three and you can tell when you are getting through to students and when you are not so rather say well I don't want to fall behind because I do this all the time if I don't feel like I need to spend more time on this chapter I know I have other chapters to do it's fine I can catch up on the other ones, but I want to make sure I get through on this one, so for a novice teacher is that you need to be patient, be prepared and don't sugar on things.

- **Engagement**

Building engagement was key to student-centeredness. Creating platforms for engagement both inside and outside lessons such as time after lessons to engage with their questions was recommended to stimulate engagement. Student engagement allowed for students to feel comfortable and it went beyond just a lecturer-student relationship.

Participant 7 continued with the discussion of student-centeredness and expanded on the concept of student engagement as follows:



Get a level of their engagement, are they getting, ask questions again which is a challenge for everybody, if I'm asking questions in a classroom nobody wants to raise their hand and say of so if you can try and find a way to make the students feels comfortable talking to you uh you make them comfortable even coming after well after a classroom I usually do pack up and leave I'll stay there because I'm sure there's students who want to ask the questions but they don't want to do this in front of the whole class but if I'm there and then they'll come and have this question okay sometimes I may say later I'll be in my office just come and talk to me later, if you can build that ability for students to trust and uh don't have a problem approaching you then you have a better chance of being successful when you go into the classroom because they are no longer looking at who are you, but they are want you to hear more from you about certain things.

- **Student independence**

Students also needed to become more independent and be able to work on their own. Hence the progress from the first to the second year was important as this was when students learned to work more independently. This facilitated learning.

- **Focus on logic behind programming**

Students need to understand the logic behind programming and not just solve the problem for the sake of it.

#### **5.5.4.3 Resources and support**

From a resource and support perspective, the following recommendations are made.

- **Online Resources**

Platforms such as the Internet which brings a plethora of resources, such as interactive videos, applications, and online communities, can significantly contribute to the students' learning process.

- **Collegial support**

Collegial support amongst lecturers was important as it allowed for discussions on how to approach classes and teaching methods.

Participant 8 did reminiscence of the times gone by and had the following to say:

Actually yes, it was, at one point in time I myself was a new person and as I told you, my colleagues and fellow mates really helped me a lot to make me understand what the process is all about. Maybe I too will do the same thing. They did give me a proper idea of how actually the system works and how you need to teach the students and basically how the basic structures of how you need to go and teach the lessons and that sort of thing.

## **5.6 Conclusion**

This chapter presented the qualitative analysis and discussion, therefore making it the most informative and detailed chapter of the entire study. This also contributed to the significant length of the chapter. The qualitative analysis was exhaustive, as an in-depth analysis was done on the qualitative data which included word clouds, treemaps, cluster analysis and thematic analysis which all lead to the building of relevant themes. The themes and sub-themes generated by the qualitative analysis not only contributed to satisfying the objectives and research questions of the study but also provided other discoveries from the qualitative data.

Due to this chapter being very rich and exhaustive with analysis and discussions of both quantitative and qualitative data, the investigator sees it fitting to highlight the findings of the study in the next chapter. The next chapter will, therefore, present the findings of the study in a structured way with relevant references to discussion and arguments made in this chapter. Furthermore, the qualitative results would be brought together and tied in to further support the findings. The findings would be presented in relation to the study objectives, research questions and frameworks as well as the other discoveries made through the inductive nature of the qualitative analysis.

## Chapter Six: Summary, Recommendations and Conclusion

### 6.1 Introduction

The aim of this study was to investigate teaching and learning of an introductory programming course at a ToT. This chapter presents a summarised analysis of the data and a concise discussion of the findings gathered from the various sources of data collection. This chapter concludes the study. The summary, conclusions and recommendations are based solely on the research findings of this particular sample population of learners' and academics' experiences of teaching and learning computer programming at an introductory level at a ToT. The research questions from Chapter One are answered and finally key areas for potential future study are included.

The following paradigms, concepts and theories guided the analysis of this research:

- Qualitative research;
- Interpretivist paradigm;
- Grounded theory;
- The nine considerations of threshold concepts; and
- The research questions.

The research methodology was guided by:

- The literature;
- The sample population;
- The interview questions;
- Observations; and
- The researcher's teaching practice.

The interview questions together with observations were used to determine academics' attitudes, their perceptions and their teaching strategies when teaching programming to first-year introductory students. This study aimed to analyse the burning and most important and crucial issues surrounding the adoption of teaching strategies when teaching and learning programming and analysed their adoption amongst participants.

The attitude of the participants did conclusively influence their intentions to use a teaching strategy and most participants did present a positive attitude with respect to improving the throughput rate of first-year IT students and were open-minded to viable innovations and instructional techniques for programming.

The findings which emanated from the participants are now summarised according to the research questions which guided the study.

For the sake of cohesion with Chapter One and the study, the research questions are repeated:

- 1) What are the teaching strategies used by ToT IT academics when teaching IT programming?
- 2) What are the perceptions of students regarding the teaching methodologies used for ToT IT programming?
- 3) What are the perceptions of ToT academics regarding factors impeding student IT programming?
- 4) How do the teaching strategies employed by IT academics at a ToT influence the learning of programming?

The common threads that permeated the research question were:

- Teaching strategies, methodologies and styles
- Perceptions of students and academics
- Teaching and learning of computer programming at an introductory first-year ToT level.

## **6.2 Research question one and research question four**

Research question one and research question four are related by a common thread of a teaching strategy. The questions probed the “WHAT?” and “HOW?” of teaching programming to first-year introductory programming students, hence the questions were analysed in conjunction with each other.

What are the teaching strategies used by ToT IT academics when teaching IT programming? and How do the teaching strategies employed by IT academics at a ToT influence the learning of programming?

In answering the above questions, I had to first identify my personal strategy/style of teaching programming to first-year IT students, then observe traits in the teaching styles of academics from the sample population. Observational data and interview data (a snapshot of teaching) were used to isolate academics' levels of competence in using teaching techniques. The immediate teaching styles were obtained from introspection and a "snapshot" of observations during data collection.

The following teaching styles were observed:

- Lecturer style;
- Exhibitor style;
- Helper strategy;
- Deputy style; and
- Mixture of above styles/strategies in a lesson (Mixed method)

I will describe each strategy observed, from the perspective of student engagement, and highlight how the particular strategy had a positive or negative influence on IT programming.

### **6.2.1 Lecturer style**

This was the most common strategy, and not surprisingly, considering the participants were all lecturers at a tertiary institution. Participants would present one-way presentations for up to 30 minutes at a time. Students would write copious notes or try and follow the lecturer as they moved from one PowerPoint slide to the next. In the computer laboratory of 60 students this did not prove effective because it was observed that some students would either not pay attention by surfing the Internet or simply have a dazed/blank look. It was also observed that some students would even have one ear piece of a head phone in their ear. The assumption is that the student would be listening to music and not the lecturer. It was also assumed that the lecturer would be the authority on the subject matter, and this proved incorrect when students

would surf the Internet and either provide an alternate solution/subject matter or correct the lecturer.

### **6.2.2 Exhibitor**

The instructor or academic would adopt the characteristics similar to an exhibitor at a news/exhibition stand, hence, the label “Exhibitor”. This was also a popular teaching style and adopted by many participants when they exhibited their worked solution of a particular programming task. The lecturer would stand before students and methodically compile and execute the programming task or project. The students would stare in awe at the knowledgeable exhibitor or lines of program code. Occasionally a student would attempt to cross question the lecturer and provide erroneous input for the worked example, for instance, requesting that the lecturer (exhibitor) include non-numeric input or alphanumeric input in a text box that clearly requires only numeric integer data type. In a class of 60 the lecturer would now attempt to answer the failed program execution leaving 59 other students to either ponder the new dilemma or check their social media feeds. It must also be noted that the lecturer would focus on the success and correctness of syntax compilation of a program during the exhibition phase, however, unsuccessful compilations and run time errors provided just as important a teaching/learning opportunity.

### **6.2.3 Helper**

In this strategy, the lecturer is not the all-knowing authority on the subject matter, as in the lecturer style or exhibitor. Students were presented with a programming task and the lecturer would assist/help either individual students or groups of students with logic, syntax or run time errors. Due to the intensive demands on the lecturer and the large class size, this style was seldom used. The class was also noisier and the classroom setup prevented effective communication between students. The lesson was also more time consuming; however, students were actively engaged with the programming task.

### **6.2.4 The Deputy**

This term is coined from the Western influence of a sheriff and his deputy. The lecturer identified high flyers in his class and deputised the students to assist individual students or groups of students with their assigned programming tasks. The lecturer created more time for himself and the “deputy lecturer” enjoyed the prestige of teaching their fellow students. This teaching style also encompasses the tutor model of teaching. The problem was identifying

effective deputy students/tutors in a lecture venue and the consultative role of the lecturer. Students were engaged with the programme task and there were less demands on the lecturer. The deputised student would also tend to overstep his role and sometimes did mislead students in assessment criteria or programming concepts. This proved problematic and it was the lecturer's skilful negotiation and diplomacy that avoided an escalation of the problem to the echelons of a student protest.

### **6.2.5 Mixed Methods Strategy**

This style of teaching was prevalent amongst the more experienced and older teaching staff. The lecturer initially started the lecture either in lecture mode or exhibitor mode and then adopted either the helper or deputising style. During a double-period lecture (approximately 2 hour), this appeared to be most effective and conducive to increased student engagement. Again, the setup of the computer lab with 3 computers per row and 10 rows on either side of the computer room, left little interaction for group discussion. Students were actively engaged in learning; however, they were much noisier in comparison to the lecture style.

As a summation for the teaching strategies observed in the lecture venues, it must be noted that given the demands on academics to complete the curriculum, the varying backgrounds of students, inclusive of class size, availability of resources and time constraints, it is a challenge to choose and recommend just one teaching strategy/style over another. Suffice to state, based on this study the mixed method style appeared to be the most desirable strategy. Learners' experiences of the use of mixed methods of teaching served as a crucial window into successful teaching strategies in programming and served as a vital focus for the study.

Based on the academics' teaching methodologies and their experiences in the classroom, data analysis showed that the learners and academics had informal knowledge of how to use teaching styles in IT and of its potential and challenges in teaching and learning. The research findings revealed that formal introduction of teaching strategies not only supported teaching and learning but also enhanced the teaching and learning experience of this selected group of learners. The mixed methods strategy provided access to collaborative learning strategies, encouraged independent student learning, and created an environment for learner diversity and increased interaction and peer learning amongst the learners. Mixed methods must be viewed as a technique to add to the academics' arsenal of teaching strategies.

### 6.3 Research Question Two and Research Question Three

Research question two and research question three are related by a common thread of a teaching/learning perception. The questions probed the “WHAT?” of teaching/learning programming perceptions of first-year introductory programming students and their academics, hence the questions would be analysed in conjunction with each other. The interview questions together with observations were used to determine academics’ attitudes, their perceptions and their teaching strategies when teaching programming to first-year introductory students.

2) What are the perceptions of students regarding the teaching methodologies used for ToT IT programming?

3) What are the perceptions of ToT academics regarding factors impeding student IT programming?

Or stated differently and in trying to obtain similar outcomes research question three also determined perceptions of academics regarding factors facilitating successful teaching and learning of IT programming. The purpose of this question was to determine factors impeding student/academic success/satisfaction in IT programming. Conversely, it could be stated as what factors promoted IT programming?

In answering the above questions, I had to first identify my personal perceptions of teaching and learning programming to first-year IT students, then observe, document and analyse the perceptions of traits in the sample population. Upon introspection, observation and analysis, the following summarised and overwhelmingly dominant perceptions are discussed in the ensuing paragraphs.

The academics perceived the following shortcomings in their teaching of introductory programming to first-year students as major hurdles to success at an introductory level:

- The large numbers of students in IT classrooms at tertiary level;
- The number of assessments; and
- Support of management.



To be successful in teaching programming, academics require a knowledge of programming concepts, the methodology/style of teaching programming and of learning theories. The findings of this study, which distinguish between the benefits and shortcomings of teaching programming, indicate the need for a framework/strategy to support academics and students.

The research findings indicated that overall the learners and academics had positive experiences regarding the teaching and learning of computer programming concepts. The numerous observation sessions revealed that overall learners enjoyed the challenges of programming and the successful compilation of programming tasks. This was an indication that the learners looked forward to and enjoyed programming. The data showed that the mixed teaching strategy is a beneficial and supportive technique for teaching and learning.

The overall perception from the interview questions and observations was that academics felt that introductory programming students had little or superficial, if not no exposure to programming concepts prior to registering for a programming qualification. According to the participants, students were ill-prepared to cope with the rigors of programming at a ToT. This was evident in the high first-year dropout rate. The sample population indicated a dropout rate of more than 50 percent. There was general consensus amongst participants that an intervention amongst first-year academics is of paramount importance to curb the high first-year dropout rate.

The majority of participants perceived their teaching strategy to be pedagogically sound in order to enable students to acquire the necessary programming skills. This is, however, debatable considering the psychological concepts of self-preservation and participant biases. The data disclosed that participants felt that a positive attitude coupled with a willingness to adapt to the ever challenging and progressive nature of the IT industry would hold present and future IT academics in good stead towards creating a positive learning environment in IT lecture rooms. Furthermore, the responses to the use of varying teaching strategies including and coupled with the use of technology amongst participants suggested the need to publicise and distribute the research findings amongst tertiary institutions. Responses to the interview questions indicated that overall teaching strategies amongst academics was easy to implement if the use of technology was coupled to their existing arsenal of strategies; however, this posed both an infrastructure and economic dilemma and was unlikely to be adopted by IT academics.

The participants indicated that there is a definite need for intervention strategies by academics that they would adopt when confronted by difficulties in programming tasks. The participants' positive attitude towards the teaching of computer programming significantly affected learners' intentions to use teaching and learning strategies in computer programming. The study revealed that use of technology is not a current practice in introductory programming courses at the ToT and the perception was that teaching with technology was cumbersome, time consuming and disruptive to the lesson. The findings recommended that academics should implement a teaching strategy coupled to teaching with technology and that it would enhance learning of programming concepts. This confirmed that teaching with technology would be suitable to support learning.

The findings of this study will help academics understand learner perceptions regarding adoption of technology-based teaching in an IT lecture room. The educational importance of technology teaching and cooperative learning cannot be dismissed. The methodology employed in this study demonstrated a valid and reliable method for supporting the adoption and diffusion of technology teaching and cooperative learning as a teaching strategy in the classroom.

The data from academics with respect to teaching programming with a fellow academic indicated that team teaching influenced self-improvement of academics. Findings suggested that academics perceived higher levels of enjoyment with team teaching programming with a fellow academic, be it in the same institution or a neighbouring tertiary institution. Observational data suggested that the academics on a similar level of academic status preferred and were more likely to select and teach programming with an academic of similar stature. Data suggested that participants were extremely confident that availability of resources, and institutional academic support enhanced their teaching of programming. Interview data from the academics complemented these findings. It was noted from academics that resources, knowledge of implementing team teaching across disciplines, and reduced class sizes enhanced the overall teaching experience.

It was evident, based on observations, that in all cases academics perceived and used the "chalk and talk" (white-board marker and talk) teaching methodology for the teaching of programming. There was minimal evidence of the use of collaborative teaching styles within the classroom setting. It was evident that there are academics that do have an informal

knowledge of teaching strategies, and in some cases, this was applied in the lecture venue. After the interview sessions it was noted that academics with a high lecturing load preferred the advantage of “less consultation time” and more self-study by students. This was extracted from observations and responses to interview questions. Academics indicated implementing innovative collaborative teaching techniques freed up more time for them to catch up with administration work. When academics allowed for collaborative group discussion amongst students from similar language backgrounds, academics indicated that students tended to speak to each other in their vernacular. This was also noted in the observation schedule. Lecturers also perceived team teaching, and pair programming as teaching strategies that could generate less contact teaching time; however, they were reluctant to implement due to constrained and tight curriculum deadlines and schedules and the perception from colleagues that they had less contact time with students. However, it was evident that learners and academics were very positive about their experiences with cooperative teaching and learning strategies. Learners were observed enjoying programming with a partner. With the exception of one academic it was stated that cooperative programming techniques did indeed help learners to better learn programming concepts. The positive and favourable outcomes of this study will make IT academics more interested in adopting a cooperative teaching strategy and more willing to cope with the transition from solitary programming to cooperative programming strategies and from lecture style teaching to mixed methods teaching. The findings of this research augur well for such a transition at an introductory programming level.

An important perception that was discussed in almost all the interview sessions was class size.

The number of students per class determined the effectiveness of teaching techniques and student understanding and ultimately throughput rates. Classes with a relatively smaller number of programmers (less than 25) proved more conducive to applying innovative and differing teaching strategies, while larger classes were noisy and were, at times, unmanageable. The class size also had a direct influence on the number of assessments. Participants indicated that having smaller class sizes meant that academics could provide more frequent tests and faster turnaround time in reviewing and moderating assessments. This was almost impossible with the larger class sizes. This had an influence on the final pass rate and eventual throughput rate. Participants indicated despondence with regard to their input in determining the number of students they could have in a class. It was a management decision that determined the

number of students who would be enrolled at first year level and this decision influenced subsequent registrations.

One of the objectives of the research questions was to determine whether the academics had specific knowledge related to the programming language or programming paradigm that they taught and the support that academics received. It was noted by all participants that programming paradigms and languages will continually evolve and become a more interactive part of the IT classroom, and as technology improves programming languages and paradigms will continue to evolve. Participants indicated that they must embrace change and surprisingly, they were open to change. They stated that the changing syllabi of IT very often meant they had to “pick up a textbook and teach oneself before teaching the class”. This also raised questions for the researcher about most of the educational institutions’ management approaches to training.

An opportunity exists for management at the tertiary institutions to implement “ongoing training workshops” to cater for this need. It was observed and duly noted that when an academic lacked the necessary skills to teach a particular concept, they tended to resist using the technique. As mentioned earlier, participants were divided and resisted the implementation of a technology-based teaching strategy. Participants were also wary of management support. It must, however, be noted that participants expressed their appreciation for learning formal methods of implementing innovative teaching strategies. The overall perception was that different managerial styles approached academic support with different and varying degrees of support.

In summation of students’ motivation, the study revealed that students appeared motivated when the lecturer engaged with programming tasks and with their diverse needs. They also appeared to enjoy active participation and having meaningful programming tasks which they could relate to, for instance, programming tasks that computed the maturity of compound interest on a retirement policy was simply too far-fetched for young first-year students to comprehend because they did not even have permanent employment. The first-year introductory student sitting in the computer laboratory is technology savvy and when confronted with heaps of PowerPoint slides or white boards filled with programming syntax would prefer to take a photograph of the information with their smartphone and digest it at their leisure. Students craved “knowledge of the information”. This implied that knowledge

implies an understanding and comprehension of the programming concepts taught. Most often in the programming computer laboratories I observed students “Googling” a program construct and simply copying and pasting code in the IDE without a clear understanding of the rationale of the programming construct. The student appeared motivated to complete the programming task in the minimal time and with the least effort; however, during formal assessments of programming, the student was now expected to program from rote memory and envision a scenario under stressful examination conditions with time constraints and no Internet access. The criteria for assessment of computer programming tasks and the ensuing conditions requires a “renaissance” in the teaching, learning and assessment of the subject.

#### **6.4 Summary: Knowledge to Practice**

In summation, this study has ascertained the following knowledge to practice with respect to teaching IT programming to first year students:

- Academics MUST “buy” into the idea of implementing innovative teaching strategies/ styles, and line managers and coordinators must allow room for academics to use such innovations and possibly even fail at them.
- Academics should remove the awarding of marks when implementing teaching strategies and concentrate on the learners’ acquisition of programming concepts. There must be a review of the number of assessments, duration of assessments and the type of assessments in programming.
- Academics should avoid monotonous PowerPoint slide presentations and replace presentations with interactive desktop programming environments. The academic must program/live code in the presence of the students inclusive of the use of erroneous test data to demonstrate program language syntax.
- Pair programming was found to be a useful software development tool in the software industry, and academics can benefit from implementing pair programming in the classroom/ lecture room (Govender, 2014). Academics should ensure that learners are paired according to some predetermined criteria and not randomly, e.g. by gender or other cultural factors.

- Tasks should be carefully assigned to students and discussion should follow to ensure that all learners benefit. Tasks must be assigned according to the concepts of problem-based learning and/or experiential learning from the software industry.
- An academic must allow for constructive criticism and predictions of program output amongst students or groups of students. This discussion ties in with the previous recommendation of a live programming session. The programming task now has several iterations with different inputs, where students are allowed to predict the successful or unsuccessful program output. Instead of avoiding programming mishaps, such as syntax, run time or logic errors, it should be the norm that academics aim to deliberately make programming mishaps and use them as a threshold concept in teaching a lesson.
- An academic must guard against assigning students a task without supervision. The academic must ensure control amongst learners during the programming session, and terminate the session, or decide on an alternate method of teaching the lesson. This is especially crucial in an introductory programming lesson, more mature programming students (level 2 or level 3) may be left unsupervised.
- Academics must rearrange the seating of classrooms and laboratories so that programmers can conveniently program without disturbing other programming students, but still have the ease of open discussion amongst programmers and the academic
- A first-year introductory programming student must master the syntax and constructs of one programming language, only then should the student be introduced to a new programming syntax and its constructs. Alternatively, the fundamentals of programming should be taught devoid of a particular programming language syntax.
- Sometimes it may be necessary to halt one teaching style and allow for individual programming or multiple styles within a single lesson. Varied and continuously modifying your teaching style motivates students to complete a programming task or even learn a new programming concept, and the probability of successful compilation and completion is greatly increased by using the mixed method style of teaching.
- All learning and teaching environments are not conducive to implementing mixed method styles of teaching programming; for effective programming experiences academics need to rearrange the classroom and computer facilities.
- Team teaching can afford the academic more time and create a more conducive, stress-free learning environment, whereas previously the academic was inundated with programming queries. With team teaching the query is discussed within a team of programming academics

and most often solved within the team partnership. Furthermore, the inexperienced academic develops his programming ability by acquiring programming skills from a peer. Similarly, the experienced academic acquires the humility of learning to keep his ego in check and learns to alternate programming solutions/teaching techniques for a programming task. Team teaching develops strong relationships of friendship that go beyond the programming task assigned to them, and the continuous discussion of a programming task or concept makes for more sociable and better programmers.

- Programming “mistakes” or software bugs assist the understanding of programming concepts and must not be avoided but rather pro-actively used as a teaching tool in programming lessons.
- Academics must support the use of technology in the classroom, since without such support teaching programming is doomed from its inception.
- Competition amongst learners or academics is not always healthy. First-year programming students do not necessarily share their programming expertise with each other due to competition; however, if the academic removes the extrinsic motivation of awarding marks, then there is a greater likelihood of successfully sharing programming knowledge and hence successful program compilation and understanding.

## **6.5 Concluding remarks & final implications to the findings from the study**

Teaching IT programming to first-year students at a ToT refers specifically to academics using a programming language to educate first-year students in the basics of the three most fundamental principles of programming, namely: assignment; selection; and iteration. The data suggests that teaching IT programming to first-year students becomes problematic when an academic overlaps the concepts of Information Systems (IS) and theorises programming concepts. IT academics must acknowledge the theory of programming and IS; however, they must be motivated to ensure the pragmatic practising of programming.

To quote Alan Turing, (often called the father of modern computing):

Programming is a skill best acquired by practice and example rather than from books.  
(Cooper, 2013).

This stance reflects Alan Turing's insight into programming, which is still applicable in today's programming classes. IT students must practice and practice programming tasks to become proficient in a particular programming concept or programming language.

In keeping with programming by practice, IT academics must ensure that collaborative and cooperative learning and teaching techniques are used in their computer programming classes. It could be as simple as the physical setup of the computer room to ensure that students can either work in groups or in pairs, or if they prefer to simply work alone, or the assigning of industry-related software case-based projects.

Furthermore, academics must refocus their teaching styles from that of an all-knowing sage of programming, an exhibitionist, to the role of a facilitator of programming knowledge employing intensive problem solving/project-based learning and the use of technology in a programming class. Previously, centuries in the past, most knowledge was gained through the "all knowledgeable" academic, the sage who "spoon-fed" an ignorant learner. Today, the incredible power of the Internet coupled with advancements in handheld PC tablets is creating challenges to academics and learners alike. Information is virtually at our fingertips and IT academics must find novel teaching strategies to keep the programming students of today interested and eager to solve programming tasks.

Introduction of IT programming in our schools and tertiary institutions has not only created new possibilities for our learners to engage in new ways of learning, but also provides them with job-related software development skills currently being used in the software development industry. At a simplistic level it may even provide academics with another avenue to broaden the cognitive abilities of twenty-first century learners.

With respect to the concepts of teaching with technology, the role of the academic in the computer lab is not replaced by technology. The IT academic must be a facilitator of technology and ensure that students use technology as a tool to assist in the learning of programming concepts. They must not sit passively and watch a video without active participation. IT programming students, and their lecturers must fully immerse themselves and embrace the concepts of learning with technology which is inclusive of, but not limited to, social networking/social media platforms, learning management systems, video technology,



blogs, machine learning concepts, artificial intelligence and robotics in the teaching of IT programming.

There must be a definite purposeful renaissance of the IT teaching curriculum, not just an addition of technology-style learning. It must also embody elements of critical thinking that must be inseparable from the media or technology being used. As one participant indicated, “I don’t have the time to use technology or watch videos I am constrained by time and curriculum completion”.

The literature is conclusive with respect to teaching with technology:

Technology will never replace teachers, but a teacher who cannot teach with technology will be replaced by another one who can.” – Zuzana Molčanová, Microsoft, Slovakia November 2, 2017

It is a fervent hope that none of the participants in the study would want to be replaced, nor would they want to be replaced by a robotic academic. Academics have the empathy that technology and robotics and AI lack and a motivated and dedicated IT academic would aim to motivate and to improve IT learners’ understanding of programming concepts and ultimately to improve the pass rate in programming subjects at an introductory programming level at a ToT.

## **6.6 Concluding remarks & a key message of the study**

In conclusion, this study has exposed teaching and learning concepts in programming at a ToT at an introductory first-year level. It has certainly changed the researcher’s point of view and created new threshold concepts in the schema of IT programming concepts, and expanded teaching and learning opportunities/strategies ultimately to the pedagogy of IT programming at an introductory level at a ToT. The results are encouraging and will contribute to the growing body of literature related to the teaching and learning of programming in an IT classroom/lecture room.

Recommendations for teaching and learning computer programming based on analysis of data from this study include:

Further research must be conducted by academics to investigate both learner and academic experiences when teaching and learning computer programming. This may also require a search for appropriate assessment tools and criteria to facilitate the testing and examination of first-year introductory programming students.

Furthermore, in order to achieve the benefits of programming, it is recommended that tertiary institutions become more actively involved in the concepts of AI, robotics and cloud computing. As an initial point of departure, academics require explicit knowledge of implementing programming code to control a robotic device or developing and designing a website based on cloud computing concepts. This could be as simple as a programming workshop on AI/robotics or website design with academics who are keen and who are actively engaged in programming activities. Students must be introduced to implementing programming code to control a robotic device from their very first practical lesson when assigned a programming task.

The following points represent some key areas for potential future study:

- A study on teaching and learning computer programming amongst learners from different year groups; e.g. level 2.
- Pilot studies in reducing the number of formal assessments at an introductory programming level and the awarding of marks and promotion based on group assessments or project-based assessments.
- Inclusion of open resources during assessments and a renaissance in computer programming assessments.
- Investigating variations and differences in how software companies develop and design software and the impact on implementing such strategies at a ToT.

## References

- Alston, P., Walsh, D., & Westhead, G. (2015). Uncovering “threshold concepts” in Web development: an instructor perspective. *ACM Transactions on Computing Education (TOCE)*, 15(1), 1-18.
- Anderson, G. L., & Herr, K. (1999). The new paradigm wars: Is there room for rigorous practitioner knowledge in schools and universities? *Educational researcher*, 28(5), 12-40.
- Ardichvili, A. (2003). Constructing socially situated learning experiences in human resource development: an activity theory perspective. *Human Resource Development International*, 6(1), 5-20.
- Ausubel, D. P., Novak, J. D., & Hanesian, H. (1968). Educational psychology: A cognitive view.
- Babbie, E. R. (2015). *The basics of social research*. Nelson Education.
- Badat, S. (2004). Transforming South African higher education, 1990-2003: goals, policy initiatives and critical challenges and issues. *National policy and a regional response in South African higher education*, 1-50.
- Barrows, H. S. (1996). Problem-based learning in medicine and beyond: A brief overview. *New directions for teaching and learning*, 1996(68), 3-12.
- Bati, T. B., Gelderblom, H., & Van Biljon, J. (2014). A blended learning approach for teaching computer programming: design for large classes in Sub-Saharan Africa. *Computer Science Education*, 24(1), 71-99.
- Bauer, M. W., & Gaskell, G. (Eds.). (2000). *Qualitative researching with text, image and sound: A practical handbook for social research*. Sage.
- Begel, A., & Nagappan, N. (2008, October). Pair programming: what's in it for me? In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement* (pp. 120-128).
- Bell, J. (2014). *Doing Your Research Project: A guide for first-time researchers*. McGraw-Hill Education (UK).
- Berg, M., & Seeber, B. K. (2016). *The slow professor: Challenging the culture of speed in the academy*. University of Toronto Press.
- Bernard, H. R. (2017). *Research methods in anthropology: Qualitative and quantitative approaches*. Rowman & Littlefield.
- Bhana, A. (1999). Participatory action research: A practical guide for realistic radicals. *Research in practice: Applied methods for the social sciences*, 227-238.
- Birks, D. F., Fernandez, W., Levina, N., & Nasirin, S. (2013). Grounded theory method in information systems research: its nature, diversity and opportunities. *European Journal of Information Systems*, 22(1), 1-8.
- Boeree, C. G. (2006). Personality theories: An introduction. *Psychology Department*.
- Boustedt, J., Eckerdal, A., McCartney, R., Moström, J. E., Ratcliffe, M., Sanders, K., & Zander, C. (2007). Threshold concepts in computer science: do they exist and are they useful? *ACM SIGCSE Bulletin*, 39(1), 504-508.
- Boyчук Duchscher, J. E., & Morgan, D. (2004). Grounded theory: reflections on the emergence vs. forcing debate. *Journal of advanced nursing*, 48(6), 605-612.
- Boyer, E. L. (1990). *Scholarship reconsidered: Priorities of the professoriate*. Princeton University Press, 3175 Princeton Pike, Lawrenceville, NJ 08648.
- Brunlin, G. (2001). The third task of universities or how to get universities to serve their communities. *Handbook of action research*, 440-446.
- Bruner, J. S. (1966). *Toward a theory of instruction* (Vol. 59). Harvard University Press.
- Buchanan, D., & Bryman, A. (Eds.). (2009). *The Sage handbook of organizational research methods*. Sage Publications Ltd.

- Bullough Jr, R. V., & Pinnegar, S. (2001). Guidelines for quality in autobiographical forms of self-study research. *Educational researcher*, 30(3), 13-21.
- Burrell, G., & Morgan, G. (2017). *Sociological paradigms and organisational analysis: Elements of the sociology of corporate life*. Routledge.
- Carbone, A., & Kaasbøll, J. J. (1998, August). A survey of methods used to evaluate computer science teaching. In *Proceedings of the 6th annual conference on the teaching of computing and the 3rd annual conference on Integrating technology into computer science education: Changing the delivery of computer science education* (pp. 41-45).
- Carter, J., & Dewan, P. (2018, June). Contextualizing inferred programming difficulties. In *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering* (pp. 32-38).
- Caspersen, M. E., & Bennedsen, J. (2007, September). Instructional design of a programming course: a learning theoretic approach. In *Proceedings of the third international workshop on Computing education research* (pp. 111-122).
- Charmaz, K. (1996). The Search for Meanings-Grounded Theory In: Smith JA, L VL, editors. *Rethinking Methods in Psychology*.
- Charmaz, K. (1990). 'Discovering' chronic illness: using grounded theory. *Social science & medicine*, 30(11), 1161-1172.
- Charmaz, K. (2003). Grounded theory: objectivist and constructivist methods. In 'Strategies for Qualitative Inquiry'. (Eds NK Denzin, YS Lincoln) pp. 249–291.
- Charmaz, K. (2006). *Constructing grounded theory: A practical guide through qualitative analysis*. sage.
- Charmaz, K. (2014). *Constructing grounded theory*. sage.
- Charmaz, K. (2015). Teaching theory construction with initial grounded theory tools: A reflection on lessons and learning. *Qualitative health research*, 25(12), 1610-1622.
- Charmaz, K., & Belgrave, L. L. (2019). Thinking about data with grounded theory. *Qualitative Inquiry*, 25(8), 743-753.
- Denzin, N. K., & Lincoln, Y. S. (2008). *Strategies of qualitative inquiry* (Vol. 2). Sage.
- Choy, M., Nazir, U., Poon, C. K., & Yu, Y. T. (2005, July). Experiences in using an automated system for improving students' learning of computer programming. In *International Conference on Web-Based Learning* (pp. 267-272). Springer, Berlin, Heidelberg.
- Clancey, W. (1995). 'A tutorial on situated learning'. In *Proceedings of the International Conference on Computers and Education* (Taiwan). Charlottesville, VA: AACE, 49–70.
- Clement, T. E. (2018). Coding Literacy: How Computer Programming Is Changing Writing. Annette Vee. Cambridge, MA: The MIT Press, 2017. *Journal of the Association for Information Science & Technology*, 69(9), 1174-1176.
- Cockburn, A., & Williams, L. (2000). The costs and benefits of pair programming, in eXtreme Programming and Flexible Processes in Software Engineering—XP2000. *Google Scholar Google Scholar Digital Library Digital Library*.
- Code.org (2019a). "What will you create?". 2019, Retrieved from <https://code.org/>.
- Code.org. (2019b). Leaders and trend-setters all agree on one thing. Retrieved from <https://code.org/quotes>
- Cohen, L., Manion, L., & Morrison, K. (2000). *Research methods in education* [5 th edn] London: Routledge Falmer. *Teaching in higher education*, 41, 21.
- Cooper, S. B., & Van Leeuwen, J. (Eds.). (2013). *Alan Turing: His work and impact*. Elsevier.

- Corbin, S., & Strauss, A. (2007). *Business Research Method*. In: New York: Mc Graw Hill Co. Inc.
- cplusplus.com (2019). "Welcome to cplusplus.com." 2019, from [http://www.cplusplus.com/doc/tutorial/basic\\_io/](http://www.cplusplus.com/doc/tutorial/basic_io/).
- Creswell, J. W. (1998). *Qualitative research and research design: Choosing among five traditions*. London: Thousand Oaks.
- Creswell, J. W. (2014). *Research design: International student edition*. In: Los Angeles: SAGE.
- Creswell, J. W., & Plano Clark, V. L. (2011). *Designing and conducting mixed method research*. 2nd Sage. Thousand Oaks, CA, 201.
- Dawson, J. W. (1985). Andrew Hodges. Alan Turing: the enigma. Burnett Books, London, and Simon and Schuster, New York, 1983, ix+ 587 pp. *The Journal of Symbolic Logic*, 50(4), 1065-1067.
- Devine, J., Finney, J., de Halleux, P., Moskal, M., Ball, T., & Hodges, S. (2019). MakeCode and CODAL: Intuitive and efficient embedded systems programming for education. *Journal of Systems Architecture*, 98, 468-483.
- De Vos, A. S., Strydom, H., Fouché, C. B., & Delpont, C. S. L. (2002). Research at grass roots: For the social sciences and human services. *Pretoria: Van Schaik*.
- DHET (2018). "A National Framework of Enhancing Academics as University Teachers." 2019, from <http://heltasa.org.za/wp-content/uploads/2019/03/March.pdf>.
- Dickey, M. R. (2016). "President Obama Wants \$4 Billion To Bring Computer Science Education To Every K-12 School." Retrieved 8 March 2019, 2019, from <https://techcrunch.com/2016/01/30/president-obama-wants-4-billion-to-bring-computer-science-education-to-every-k-12-school/>.
- Driscoll, M. P. (2005). *Psychology of learning for instruction*.
- Duffy, M. E. (1987). Methodological triangulation: a vehicle for merging quantitative and qualitative research methods. *Image: The Journal of Nursing Scholarship*, 19(3), 130-133.
- Duncan, C., Bell, T., & Tanimoto, S. (2014, November). Should your 8-year-old learn coding? In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education* (pp. 60-69).
- Du Plooy, G. M. (2009). *Communication research: Techniques, methods and applications*. Juta and Company Ltd.
- DUT. (2015). DUT STRATEGIC PLAN 2015 - 2019. Retrieved from <https://www.dut.ac.za/wp-content/uploads/2016/09/DUT-strategic-plan-2015.pdf>
- DUT. (2018). *Faculty Handbook*. In. Retrieved from <https://www.dut.ac.za/wp-content/uploads/handbooks/ACCINFO%20IT.pdf>
- East, J. P., Thomas, S. R., Wallingford, E., Beck, W., & Drake, J. (1996). Pattern-based programming instruction. *age*, 1, 1.
- Efklides, A. (2006). Metacognition and affect: What can metacognitive experiences tell us about the learning process? *Educational research review*, 1(1), 3-14.
- Ellsworth, E. (1997). *Teaching positions: Difference, pedagogy, and the power of address*. Teachers College Press, 1234 Amsterdam Avenue, New York, NY 10027.
- Euchner, J. (2018). The internet of things. *Research-Technology Management*, 61(5), 10.
- Eun, B. (2008). Making connections: Grounding professional development in the developmental theories of Vygotsky. *The teacher educator*, 43(2), 134-155.
- Fach, A. (2017). "Coding and design The art behind computer technology." 2018, from <http://1111arts.org/coding-and-design/>.

- Feldman, A., Paugh, P., & Mills, G. (2004). Self-study through action research. In *International handbook of self-study of teaching and teacher education practices* (pp. 943-977): Springer.
- Feldman, A., Paugh, P., & Mills, G. (2004). Self-study through action research. In *International handbook of self-study of teaching and teacher education practices* (pp. 943-977). Springer, Dordrecht.
- Feurzeig, W. (1981). Microcomputers in Education. Report No. 4798.
- Feurzeig, W., Papert, S. A., & Lawler, B. (2011). Programming-languages as a conceptual framework for teaching mathematics. *Interactive Learning Environments, 19*(5), 487-501.
- Flick, L. B. (1998). Teaching Practices That Provide Cognitive Scaffolding for Classroom Inquiry.
- Galorath, D. D., & Evans, M. W. (2006). *Software sizing, estimation, and risk management: when performance is measured performance improves*. CRC Press.
- Gary, K. (2015). Project-based learning. *Computer, 48*(9), 98-100.
- Gatto, J. T. (2003). *The underground history of American education*. New York, NY: Oxford Village Press.
- Gergen, K., Gergen, M., & Steier, F. (1991). Research and reflexivity.
- Glaser, B. G., & Strauss, A. (1967). The Discovery of Grounded Theory: Strategies for Qualitative Research (Weidenfeld and Nicholson, London). *Google Scholar*.
- Glaser, B. G. (2001). *The grounded theory perspective: Conceptualization contrasted with description*. sociology press.
- Glaser, B. G., & Strauss, A. L. (2009). *Discovery of grounded theory: Strategies for qualitative research*. Transaction Books
- Glesne, C., & Peshkin, A. (1992). *Becoming qualitative researchers*. White Plains, NY.
- Goebel, J., & Maistry, S. (2019). Recounting the role of emotions in learning economics: Using the Threshold Concepts Framework to explore affective dimensions of students' learning. *International Review of Economics Education, 30*, 100145.
- Goetz, J. P., & LeCompte, M. D. (1984). *Ethnography and qualitative designs in ethnographic research*. New York: Academic.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.
- Govender, I. (2006). *Learning to Program, Learning to Teach Programming: Pre-and In-service Teachers' Experiences of an Object-oriented Language* (Doctoral dissertation, University of South Africa).
- Govender, T P. [Prinavin Govender]. (2010, October 10). Prinavin Govender [video file] Retrieved from <https://www.youtube.com/user/prinaving1/>
- Govender, D. W., & Govender, T. P. (2014). Using a collaborative learning technique as a pedagogic intervention for the effective teaching and learning of a programming course. *Mediterranean Journal of Social Sciences, 5*(20), 1077.
- Govender, T. P. (2018). "Prinavins Project0 15 Feb." 2019, from <https://scratch.mit.edu/projects/96970522/>.
- Greeno, J. G., Collins, A. M., & Resnick, L. B. (1996). Cognition and learning. *Handbook of educational psychology, 77*, 15-46.
- Guzdial, M., & Guo, P. (2014). The difficulty of teaching programming languages, and the benefits of hands-on learning. *57*(7 %J Commun. ACM), 10–11. doi:10.1145/2617658
- Harvard. (2019). CS50: Introduction to Computer Science. Retrieved from <https://online-learning.harvard.edu/course/cs50-introduction-computer-science>
- Hatch, J. A. (2002). *Doing qualitative research in education settings*. Suny Press.

- Havenga, M., & Mentz, E. (2009, June). The school subject information technology: A South African perspective. In *Proceedings of the 2009 Annual Conference of the Southern African Computer Lecturers' Association* (pp. 76-80).
- Heather, N., Rollnick, S., & Bell, A. (1993). Predictive validity of the Readiness to Change Questionnaire. *Addiction*, 88(12), 1667-1677.
- Heick, T. (2015). "The Gradual Release of Responsibility Model In 6 Simple Words." 2018, from <https://www.teachthought.com/pedagogy/the-gradual-release-of-responsibility-model-in-6-simple-words/>.
- Henning, E., Van Rensburg, W., & Smit, B. (2004). Finding your way in qualitative research, Van Schaik
- Henning, E., & Van der Westhuizen, D. (2004). Crossing the digital divide safely and trustingly: how ecologies of learning scaffold the journey. *Computers & Education*, 42(4), 333-352.
- Henri, M., Johnson, M. D., & Nepal, B. (2017). A review of competency-based learning: Tools, assessments, and recommendations. *Journal of engineering education*, 106(4), 607-638.
- Hodson, D., & Hodson, J. (1998). From constructivism to social constructivism: A Vygotskian perspective on teaching and learning science. *School science review*, 79(289), 33-41.
- Hussey, T., & Smith, P. (2003). The uses of learning outcomes. *Teaching in higher education*, 8(3), 357-368.
- Hutchison, A., Nadolny, L., & Estapa, A. (2016). Using coding apps to support literacy instruction and develop coding literacy. *The Reading Teacher*, 69(5), 493-503.
- Ismail, M. N., Ngah, N. A., & Umar, I. N. (2010). Instructional strategy in the teaching of computer programming: a need assessment analyses. *TOJET: The Turkish Online Journal of Educational Technology*, 9(2).
- Jantjies, M., & Joy, M. (2015). Mobile enhanced learning in a South African context. International Forum of Educational Technology and Society.
- Javidi, G., & Sheybani, E. (2014). Teaching computer programming through game design: A game-first approach. *GSTF Journal on Computing (JoC)*, 4(1), 17-22.
- Jobs, Steve. "Steve Jobs on Computer Science". Online video clip. YouTube. <https://www.youtube.com/watch?v=IY7EsTnUSxY> , October 22, 2013. 2018.
- Johnson, R., & Waterfield, J. (2004). Making words count: the value of qualitative research. *Physiotherapy Research International*, 9(3), 121-131.
- Jørgensen, M. (2014). Failure factors of small software projects at a global outsourcing marketplace. *Journal of Systems and Software*, 92, 157-169.
- Kallia, M., & Sentance, S. (2017, November). Computing Teachers' Perspectives on Threshold Concepts: Functions and Procedural Abstraction. In *Proceedings of the 12th Workshop on Primary and Secondary Computing Education* (pp. 15-24).
- Káta, Z. (2015). The challenge of promoting algorithmic thinking of both sciences-and humanities-oriented learners. *Journal of Computer Assisted Learning*, 31(4), 287-299.
- 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.
- Klopper, H. (2008). The qualitative research proposal. *Curationis*, 31(4), 62-72.
- Kosnik, C., Lassonde, C., & Galman, S. (2009). What Does Self-Study Research Offer Teacher Educators? In *Self-study research methodologies for teacher educators* (pp. 225-239). Brill Sense.
- Krefting, L. (1991). Rigor in qualitative research: The assessment of trustworthiness. *American journal of occupational therapy*, 45(3), 214-222.

- Lahtinen, E., Ala-Mutka, K., & Järvinen, H. M. (2005). A study of the difficulties of novice programmers. *Acm sigcse bulletin*, 37(3), 14-18.
- Land, R., Cousin, G., Meyer, J. H., & Davies, P. (2005). Threshold concepts and troublesome knowledge (3): implications for course design and evaluation. *Improving student learning diversity and inclusivity*, 4, 53-64.
- Lather, P. (1998). Critical pedagogy and its complicitities: A praxis of stuck places. *Educational theory*, 48(4), 487.
- Lave, J., & Wenger, E. (1991). *Situated learning: Legitimate peripheral participation*. Cambridge university press.
- Lee, H. C., & Blanchard, M. R. (2019). Why teach with PBL? Motivational factors underlying middle and high school teachers' use of problem-based learning. *Interdisciplinary Journal of Problem-Based Learning*, 13(1), 2.
- Leedy, P. D., & Ormrod, J. E. (2005). *Practical research*. Pearson Custom.
- Lofland, J. (1974). Styles of reporting qualitative field research. *The American Sociologist*, 101-111.
- Lunt, B. M., Ekstrom, J. J., Gorka, S., Hislop, G., Kamali, R., Lawson, E., ... & Reichgelt, H. (2008). Curriculum guidelines for undergraduate degree programs in information technology. Retrieved March, 2(2015).
- Macgregor, K. (2007). South Africa: Student drop-out rates alarming. *University World News*, 3.
- Massoud, L., Hallman, S., Plaisent, M., & Bernard, P. (2018). Applying Multidisciplinary Teaching Techniques to the Computer Programming/Coding Classroom. *Journal of Information Technology & Economic Development*, 9(1).
- MathWorks (2019a). "Classify Webcam Images Using Deep Learning." 2019, from <https://www.mathworks.com/help/deeplearning/examples/classify-images-from-webcam-using-deep-learning.html>.
- MathWorks (2019b). "MATLAB and Simulink Training." 2019, from <https://matlabacademy.mathworks.com/>
- Mavunga, G. (2014). The Contribution of Under-Preparedness to Low First Year Success Rates as Perceived by Lecturers and second Year Students and at a Comprehensive South African University. *Mediterranean Journal of Social Sciences*, 5(20), 1748-1748.
- Maxwell, D. (2016, July 4). Coding is a Necessary Part of 21st Century Education. Retrieved from The Educator: <https://www.theeducator.com/blog/coding-necessary-part-21st-century-education/>
- McAllister, M., & Stockhausen, L. (2001). Using action research within a school of nursing: exposing tensions in ideologies. *Australian Journal of Advanced Nursing*, The, 18(4), 15.
- McGaghie, W. C. (2015). When I say... mastery learning. *Medical education*, 49(6), 558-559.
- McMillan, J. H., Schumacher, S., & Singh, J. (1993). *Study Guide to Accompany MacMillan, Schumacher, Research in Education: A Conceptual Introduction*. HarperCollins College Publishers.
- McWilliam, E. (2004). W (h)ither Practitioner Research? *The Australian Educational Researcher*, 31(2), 113-126.
- Mendes, A. J., Paquete, L., Cardoso, A., & Gomes, A. (2012, October). Increasing student commitment in introductory programming learning. In *2012 Frontiers in Education Conference Proceedings* (pp. 1-6). IEEE.
- Merriam, S. B. (1995). What can you tell from an N of 1? Issues of validity and reliability in qualitative research. *PAACE Journal of lifelong learning*, 4, 51-60.



- Meyer, J., & Land, R. (2003). *Threshold concepts and troublesome knowledge: Linkages to ways of thinking and practising within the disciplines* (pp. 412-424). Edinburgh: University of Edinburgh.
- Meyer, F., Jan, H., & Land, R. (2005). Threshold concepts and troublesome knowledge (2): Epistemological considerations and a conceptual framework for teaching and learning. *Higher Education*, 49(3), 373.
- Myers, M. D., & Newman, M. (2007). The qualitative interview in IS research: Examining the craft. *Information and organization*, 17(1), 2-26.
- Miles, M. B., & Huberman, A. M. (1984). Qualitative data analysis: A sourcebook of new methods. In *Qualitative data analysis: a sourcebook of new methods*. Sage publications.
- Miles, M. B., & Huberman, A. M. (1994). *Qualitative data analysis: An expanded sourcebook*. sage.
- Miller, C. S., Settle, A., & Lalor, J. (2015, September). Learning object-oriented programming in python: Towards an inventory of difficulties and testing pitfalls. In *Proceedings of the 16th Annual Conference on Information Technology Education* (pp. 59-64).
- Mintzes, J. J., Wandersee, J. H., & Novak, J. D. (1998). Meaningful learning in science: The human constructivist view. *Teaching science for understanding*, 328-350.
- Molokken, K., & Jorgensen, M. (2003, September). A review of software surveys on software effort estimation. In *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.* (pp. 223-230). IEEE.
- Moore, S., Bruck, D., Hood, B., MacDonald, I., & Hallett, R. (2001). Enhancing student satisfaction in higher education: the creation of staff teaching communities. *Australian Educational Researcher*, 28(2), 79.
- Morrow, S. L., & Smith, M. L. (2000). Qualitative research for counseling psychology. *Handbook of counseling psychology*, 3, 199-230.
- Motshekga, A. (2019). "Remarks at the media briefing hosted by Minister of Basic Education, on the occasion of the last post-Council of Education Ministers' meeting." Retrieved 8 March 2019, 2019, from <https://www.education.gov.za/Newsroom/Speeches/tabid/950/ctl/Details/mid/8127/ItemID/6002/Default.aspx>.
- Mouton, J., & Marais, H. C. (1988). *Basic concepts in the methodology of the social sciences*. Hsrc Press.
- Newman, D. S. (2012). A grounded theory study of supervision of preservice consultation training. *Journal of educational and psychological consultation*, 22(4), 247-279.
- Nicholson, A. E., & Fraser, K. M. (1997, July). Methodologies for teaching new programming languages: A case study teaching LISP. In *Proceedings of the 2nd Australasian conference on Computer science education* (pp. 84-90).
- Onurkan Aliusta, G., & Özer, B. (2017). Student-centred learning (SCL): roles changed? *Teachers and Teaching*, 23(4), 422-435.
- Papert, S. (1980). *Mindstorms: children, computers, and powerful ideas* Basic Books. Inc. New York, NY.
- Patton, M. Q. (2002). Two decades of developments in qualitative inquiry: A personal, experiential perspective. *Qualitative social work*, 1(3), 261-283.
- Pearson, P. D., & Gallagher, M. C. (1983). The instruction of reading comprehension. *Contemporary educational psychology*, 8(3), 317-344.
- Perkins, D. (2006). Constructivism and troublesome knowledge. *Overcoming barriers to student understanding: Threshold concepts and troublesome knowledge*, 1, 33-47.

- PİŞKİN, O. (2019). Alan Turing. Retrieved from <https://powerzeka.com/scientists/alan-turing/>
- Pressley, M. (1996). The Challenges of Instructional Scaffolding: The Challenges of Instruction That Supports Student Thinking. *Learning Disabilities Research and Practice, 11*(3), 138-46.
- Ramaphosa, C. (2019). State of the Nation. S. A. Government.
- Ramsden, P. (2003). *Learning to teach in higher education*. Routledge.
- Ramsden, A., & Bate, A. (2008). Using word clouds in teaching and learning. Retrieved on September, 1, 2013.
- Reinharz, S., & Davidman, L. (1992). *Feminist methods in social research*. Oxford University Press.
- Reitzes, M. (2009). The impact of democracy on development: The case of South Africa.
- Reynolds, G. W., & Stair, R. M. (2016). *Fundamentals of Information Systems*. Cengage Learning.
- Robinson, K., & Aronica, L. (2015). *Creative schools: Revolutionizing education from the ground up*. Penguin UK.
- Roe, E. A., & Bartelt, T. (2015). Converting a Traditional Engineering Technology Program to a Competency-based, Self-paced, Open-entry/Open-exit Format. In *ASEE Annual Conference & Exposition, Seattle*.
- Rountree, J., & Rountree, N. (2009, January). Issues regarding threshold concepts in computer science. In *Proceedings of the Eleventh Australasian Conference on Computing Education-Volume 95* (pp. 139-146).
- Sadeghi, S. H. (2015). Cultural context and e-practice: an assessment on USA institutions. *International Journal of Advanced and Applied Sciences, 2*(9), 32-36.
- Sandelowski, M., & Barroso, J. (2002). Finding the findings in qualitative studies. *Journal of nursing scholarship, 34*(3), 213-219.
- Sarstedt, M. and E. Mooi (2019). Cluster Analysis. A Concise Guide to Market Research: The Process, Data, and Methods Using IBM SPSS Statistics. Berlin, Heidelberg, Springer Berlin Heidelberg: 301-354.
- Sarstedt, M., & Mooi, E. (2019). A Concise Guide to Market Research: The Process, Data, and Methods using IBM SPSS Statistics.
- Savery, J. R. (2015). Overview of problem-based learning: Definitions and distinctions. *Essential readings in problem-based learning: Exploring and extending the legacy of Howard S. Barrows, 9*, 5-15.
- Scaruffi (2016). "A Brief History of Electrical Technology Part 4: Mainframes and Minicomputers." Retrieved 20 February 2019, from <https://www.scaruffi.com/science/elec4.html>.
- Schon, D. (1983). A. (1983). The reflective practitioner: How professionals think in action.
- Schwartzman, L. (2010). Transcending disciplinary boundaries: A proposed theoretical foundation for threshold concepts. In *Threshold concepts and transformational learning* (pp. 21-44). Brill Sense.
- Seidman, I. (1998). Interviewing as qualitative research New York. NY: Teachers.
- Shanahan, M., & Meyer, J. H. (2006). The troublesome nature of a threshold concept in economics. In *Overcoming barriers to student understanding* (pp. 124-138). Routledge.
- Slyozko, T., & Zahorodnya, N. (2016). The Fourth Industrial Revolution: The Present and Future of Accounting and the Accounting Profession. *Polgari Szemle*.
- Smit, J., AA van Eerde, H., & Bakker, A. (2013). A conceptualisation of whole-class scaffolding. *British Educational Research Journal, 39*(5), 817-834.
- Sondag, M., Speckmann, B., & Verbeek, K. (2017). Stable treemaps via local moves. *IEEE Transactions on Visualization and Computer Graphics, 24*(1), 729-738.

- Steier, F. (1991). Research and reflexivity.
- Stein, D. (1998). *Situated learning in adult education* (pp. 1998-3). ERIC Clearinghouse on Adult, Career, and Vocational Education, Center on Education and Training for Employment, College of Education, the Ohio State University.
- Strauss, A. C., & Corbin, J. J. N. P. C., Sage. (1990). J. (1990). Basics of qualitative research: Grounded theory procedures and techniques.
- Strauss, A., & Corbin, J. (1998). *Basics of qualitative research techniques*. Thousand Oaks, CA: Sage publications.
- Stuttgart Spradley, J. P. (1979). The ethnographic interview. *Holt, Ri*.
- Szlávi, P., Zsakó, L., & Törley, G. (2016). The thinking toolkit of programming.
- Taylor, J. (2011). The intimate insider: Negotiating the ethics of friendship when doing insider research. *Qualitative research*, 11(1), 3-22.
- Terre Blanche, M., Durrheim, K., & Painter, D. (2002). Research in practice: MoonStats CD and user guide, applied methods for the social sciences. In: Cape Town: University of Cape Town Press.
- Training, D. o. H. E. a. (2018). Framework for enhancing academics as university teachers. H. E. a. Training. Pretoria, Government Printer: 5,6.
- Tsai, M. J., Wang, C. Y., & Hsu, P. F. (2019). Developing the computer programming self-efficacy scale for computer literacy education. *Journal of Educational Computing Research*, 56(8), 1345-1360.
- Ültanir, E. (2012). An epistemologic glance at the constructivist approach: Constructivist learning in Dewey, Piaget, and Montessori.
- 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 Den Hoonaard, W. C. (2003). Is anonymity an artifact in ethnographic research? *Journal of academic Ethics*, 1(2), 141-151.
- Van der Westhuizen, D., & Barlow-Jones, G. (2015). High school mathematics marks as an admission criterion for entry into programming courses at a South African university. *The Independent Journal of Teaching and Learning*, 10(1), 37-50.
- Van Zyl, A. (2015). "DUT Centre for Excellence in Learning and Teaching (CELT) First Year Student Experience." REDUCING DROPOUT RATES. Retrieved 6 June 2016, 2016, from <https://www.dut.ac.za/reducing-dropout-rates/>.
- von Glasersfeld, E. (2012). Sensory experience, abstraction, and teaching. In *Constructivism in education* (pp. 387-402). Routledge.
- Vygotsky, L. (1978). Interaction between learning and development. *Readings on the development of children*, 23(3), 34-41.
- Vygotsky, L. S. (1987). Collected works, Vol. 1. Problems of general psychology. Ed., Robert W. Rieber and Aaron S. Carton. Trans., Norris Minick. New York: Plenum Press. and *Masculinity*, 2004-2008.
- w3schools.com (2019a). "PHP Tutorial." 2019, from <https://www.w3schools.com/php/default.asp>.
- w3schools.com (2019b). "SQL Stored Procedures for SQL Server." 2019, from [https://www.w3schools.com/sql/sql\\_stored\\_procedures.asp](https://www.w3schools.com/sql/sql_stored_procedures.asp).
- Walford\*, G. (2005). Research ethical guidelines and anonymity. *International Journal of research & method in education*, 28(1), 83-93.
- Watson, C., & Li, F. W. (2014, June). Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 39-44).
- Weimer, M. (2002). *Learner-centered teaching: Five key changes to practice*. John Wiley & Sons.

- Wexelblat, R. L. (1978). "Preprints| History of Programming Languages Conference, Los Angeles, California. ACM, June 1978." *ACM SIGPLAN Notices* 13(8).
- Wexelblat, R. L. (1978). New books III. *ACM SIGPLAN Notices*, 13(11), 8-10.
- Whitehead, J., & McNiff, J. (2006). *Action research: Living theory*. Sage.
- Wood, R. E., Mento, A. J., & Locke, E. A. (1987). Task complexity as a moderator of goal effects: A meta-analysis. *Journal of applied psychology*, 72(3), 416.
- Yeomans, L., Zschaler, S., & Coate, K. (2019). Transformative and Troublesome? Students' and Professional Programmers' Perspectives on Difficult Concepts in Programming. *ACM Transactions on Computing Education (TOCE)*, 19(3), 1-27.
- Zeichner, K. M., & Noffke, S. E. (2001). Practitioner research. *Handbook of research on teaching*, 4, 298-330.
- Zwass, V. (2018). "Encyclopaedia Britannica Information Systems." Encyclopaedia Britannica Retrieved 2018, from <https://www.britannica.com/topic/information-system>.

## **Appendix A: Interview Schedule for the academics**

### **Introduction**

The researcher starts off by introducing himself and explaining the purpose of the interview and the origins of his interest in the research topic, and to ask for permission to tape-record the interview as well as assuring confidentiality of information emanating from the interview.

- purpose of interview
- background to the study
- permission to tape-record
- reassurance about confidentiality of information

### ***Suggested Questions during One- on-One- Interview***

1. Please, can you give me your full name?
2. What is your highest qualification?
3. What is your major subject specialization?
4. For how long have you been teaching programming?
5. Which year of study are you teaching presently?
6. What are some of the challenges you face as far as teaching programming in first year is concerned?
7. How are you trying to overcome these challenges?
8. Learners have basic knowledge about programming before coming to school, do you agree with that? Explain.
9. How do you approach your lessons in order to build on the basic knowledge?
10. Do you think learning programming will be meaningful, if it is not built on the previous knowledge? Explain.
11. How do your learners respond to programming lessons when you are using real teaching material from the environment?

12. Reality is one of the important aspects of teaching programming. If you agree with this, can you tell me how you make your lessons real?
13. What are some of the strategies you use to make your lesson real?
14. How do you develop your learners' interest in Programming?
15. Which programming language do you normally use when teaching programming and why?
16. How do you see Programming lesson when conducted in groups?
17. Which types of programming questions do you give your learners in order for them prepare for tests and/or examinations?
18. Do you think programming can be learned in isolation without social interaction?
19. What kind of teaching support material do you normally use when teaching programming?
20. Do you invite resource personnel during some of your lessons?
21. How often do you allow your learners to discuss in groups during programming lessons?
22. What are some of your observations during group discussion lessons?
23. Do you send your learners out during some of your programming lessons to explore real work related scenarios?
24. Give me some of the topics you discussed with them this year which compelled you to send them outside the classroom?
25. What do you like about the lessons which normally make you send your learners outside the classroom?
26. Describe one of your interesting lessons and what makes it so interesting?
27. How do you help your learners to maintain a positive attitude towards programming?
28. What are the concepts that you consider difficult to teach in Programming?
29. Which level of programming do you prefer to teach?

30. What can you say about the learner's performance?
31. What are possible causes?
32. How do you prepare yourself before any Programming lesson?
33. Do you prepare lesson plan every day, weekly or monthly?
34. Are you comfortable with every topic in the Syllabus?
35. If no, what are the topics you are not comfortable with?
36. How do you help the learners with such topics?
37. What strategies do you use during Programming lessons?
38. How do you teach a new concept for your learners to understand?
39. What are your questioning styles?
40. How often do you give tasks to your learners?
41. Are you satisfied with your learners' performance after giving them tasks? If no, how do you make them improve?
42. How often do you attend in-service training /workshop in programming?
43. Do you give feedback to your learners regularly?
44. How many forms of assessments do you conduct?
45. How many questions per test/practical/theory?
46. Do you prefer practical/theory assessments and why?
47. What support have you received from fellow academics, management in your dept. and from surrounding ToTs?
48. Do you use pair programming?

49. Does the use of pair programming speed up the pace of your lessons?
50. Does the use of pair programming aid you in assisting slow learners? How?
51. How does the use of pair programming affect the planning of your lessons and assessments with regard to time constraints?
52. What challenges did you experience initially when using pair programming in your classroom? How have you overcome them?
53. Are you currently experiencing any further challenges? List them.
54. Is there a difference with regard to learners' attentiveness/abilities/ background / etc.?
55. In your opinion when the learners are able to interact with their peers, does this affect their learning?
56. In your opinion how did programming impact on learners' interest, enthusiasm and motivation, especially the lower achieving ones?
57. In what ways if any does programming address the following modalities of learning: visual and auditory?  
Visual:  
Auditory:
58. What advice/recommendations would you give to a teacher who is a novice "programmer"?



## Appendix B: Observation Schedule

observation Schedule during  
programming lesson

Respondent No : \_\_\_\_\_

date : \_\_\_\_\_  
venue: \_\_\_\_\_

	YES	NO	N/A
questioned prior knowledge of a programming concept			
rigidity of teaching method/style			
built on existing programming concept/knowledge			
Fostering multiple modes of questioning			
explored value of programming in industry			
Learning-by-exploring			
individualised teaching/programming tasks			
Learning-by-creating			
exposed to challenging programming tasks			
Alignment of programming tasks to daily student experiences			
Empowering self-regulated learning			
culture based programming tasks/examples			
Building on individual student strengths and preferences			
Fostering soft skills e.g. oral presentation			
Facilitating entrepreneurship of a programming app			
Applying in practice social inclusion of different student cultures			
Recognizing informal and non-formal learning			
Monitoring programming tasks			
Innovating programming tasks			
use of existing ICT infrastructure			
Rearranging physical setup of pcs			
Learning across IS/IT/ICT curriculum			
Personalized learning/teaching			
Meaningful activities			
Facilitating peer-to-peer collaboration			
Facilitating Group work			
Pair programming			
Using Open Educational Resources (OER)			
Varying assessment formats			
Varying formative assessments			
use of social media/social networks			
innovative lecture room management			
Networking with real-word context and Industry software developers			
Learning-by-playing pc games e.g. Minecraft/etc.			
Addressing multiple learning styles			
Fostering emotional well-being of student			
Fostering critical thinking			
fostering self-learning			

varying teaching styles/methodology/skills			
reassessing student misunderstanding			
support for problem solving			
additional tutor support			
visual programming scenarios/examples			
pseudocode/input; output processing design			
rote learning vs open booking /skills transfer			
student-centeredness			
use of online resources e.g. YouTube videos			
use of deep learning /machine learning			
Use of Robotics / AI concepts /etc.			

### Appendix C: Interview Schedule with Academic participant 3

Interview with Academic Participant 3, Date: 22 February 2018 Time: \_\_\_\_\_

Questions/Interviewer	Commentary/Interviewee
<b>1. Please, can you give me your full name?</b>	Academic Participant 3
<b>(Progress Mtshali)</b>	
<b>2. What is your highest qualification?</b>	Uh it's a PhD in Computer Information Systems.
<b>(Thank you, major subject Computer Information Systems)</b>	
<b>3. How long have you been teaching Programming?</b>	Uh for about twenty-five years
<b>(Twenty-five years)</b>	
<b>4. Which year of study are you teaching presently?</b>	Uh the third years
<b>(Third years)</b>	(Mm)
<b>5. What are some of the challenges you face as far as teaching programming?</b>	
<b>(Now I'd like you to confine it to first year if you can okay please)</b>	(Mm)
	The, the, the challenge is to make Programming not to sound so complex and so foreign uh because students tend to look at Programming as, just as you look at

	<p>mathematics. Oh this is abstract, it's out of my realm of understanding things so you as a teacher you have to try and break it down to where it just seems like something they see every day maybe related to things that they do.</p>
<p>(Okay)</p>	
<p><b>6. How are you trying to overcome these challenges?</b></p>	<p>Uh by using maybe visual programming helps a lot 'cause in the past when that technology wasn't uh that great to where you could use uh maybe things like you'd dragged, and uh like if you're using Scratch. In Scratch you use objects that students could join and then they can take a pick to see the code that gets generated. In the past you used to show them codes without the visual so they didn't understand like if you say here's a variable, what's a variable but if I say here's this box where I'm going to store something let's give it a name, we give it a name, by the way that name is called a variable. So if you use visuals to students first year students just to get a concept of I guess the flow of the program, the logic of uh how it gets executed, uh even if you're using loops, you're using conditional statements and all of those.</p>
<p><b>7. Learners have basic knowledge about programming before coming to school or university, do you agree with this?</b></p>	<p>Uh some, majority of them I would think not. Uh it will depend on the country. If you go to in America still has problem in high school,</p>

	<p>some of them already start companies ‘because they can use I, to develop applications mainly for the, the iPhone there is, the tools are free there so conceptually they already know but in South Africa I tend to think that maybe it’s the opposite of that students don’t get exposed to what Programming is, when I was in school many years ago nobody talked about computer programming.</p>
<p><b>(So if you had said that students have a basic knowledge)</b></p>	
<p><b>8. How do you approach your lessons in order to build on this knowledge?</b></p>	<p>Uh use things that they, they identify with to begin with, in this day and age with social media uh for example if I use Facebook ‘because Facebook uses graph theory underneath to hook up all of these different things, if I were to say uh on your Facebook how many friends do you have? You’ll say uh well I have a thousand friends uh how are they actually connected most of them tell you “I don’t know” but if you say that there are links that link this student to that, to that, to that, to that so that by the time they look at graph theories they’ll say “oh okay” so graph theory just is slap in your face right in front of you for doing something useful, so for them to introduce them in such a way that use things they already know.</p>

<p>(Thank you, you've answered the second question, do you think learning programming will be meaningful if it is not built on previous knowledge, I think you used Facebook as an example, so we'll go onto the next one which is similar)</p>	<p>(Sure, sure)</p>
<p><b>9. How do your learners respond to programming lessons when you are using real teaching material from the environment?</b></p>	
<p>(Your example was Facebook; I'll take it as that)</p>	<p>(Sure, sure)</p>
<p><b>10. Reality is one of the important aspects of teaching programming. If you agree with this, can you tell me how you make your lessons real?</b></p>	
<p>(I'll consider the Facebook example as well)</p> <p><b>11. What are some of the strategies?</b></p>	<p>(Sure, sure)</p>
<p>(Uh if I'm okay to answer that you've used Facebook as an example where the students are with real life thinking</p>	<p>(Yeah)</p>
<p><b>12. Which Programming language do you normally use when teaching Programming and why?</b></p>	<p>It depends, if I deal with third year students I tend to use C Sharp because I know a lot of different Programming languages going way back to Assembly on the 360 uh on the main frame computers which was very difficult all the way to modern languages like C Sharp, uh Java and then you go onto the web uh you look at JavaScript, Angular JS and the list</p>

just goes on and on. you have to use the right language for the environment, for example if I'm teaching students when I teach my students I tell them that majority of them if they go into Programming there are three different areas you could either work on the front end, user interfaces in the middle area where you're actually building business rules or applying business rules to solve problems or if you're from the back end which will be databases I'm just leaving the infrastructure like if you're looking at security or networking, majority of them will end up implementing business rules so the language there I would use I guess is C sharp because it's not fancy you're not worried about colors and all of that. You just want to say here is the algorithm that an insurance company uses to calculate the premium so you have to understand that I'm taking this variable and then I'm multiplying I'm doing all of these things I'm putting the data in in the database but if I'm teaching someone that's building user interfaces C sharp is not the best language to use for that because it's designed to create objects that are inherent with other things whereas on the front and even though if you look at Java Script there is objects already in programming but most of the time you spend at the layouts so the language that are very good at doing that, Angular for example may be bootstrap and the other libraries that are available now, then I go

	<p>back to the back end from the database then I won't use Java Script for example to do that, C sharp wouldn't be, you can depending on the platform 'cause I can write C sharp modules that I can, that use, uh if I'm using Microsoft sequel server uh to run those modules to process the data but then I would use SQL which is a language for that particular environment, to answer your question, I hope I am</p>
<p><b>(Yeah no that's lovely, lovely thank you)</b></p>	<p>It just depends on where the students are focused that I'm actually thinking.</p>
<p><b>13. How do you see Programming lessons when conducted in groups?</b></p>	<p>Uh it depends again, if I'm looking at first year students group work wouldn't work very well part of the problem is that we have different personalities, some students will be the A type of personalities where they'll just take over and the other reserved students who rarely tend to have an opinion then uh those students are going to be left behind because the ones with type A personalities will just run all over all of them, especially in our culture here we have a lot of that students tend to be reserved so if you put them in a group they will struggle but if I'm looking at third years or senior students I would assume that they have enough background information so if you put them in a group, again as a teacher I've done this is to look at them, everybody has to have a task, you do this, you do this, you do this, I want to say you guys do this build this and then I sit back</p>



	<p>and watch because again the type A personalities are going to go home and write this whole thing, come back the next day and say well it's done so the other people don't get a chance to actually participate so if you say group, everybody is responsible for a part of the entire project.</p>
<p><b>(Thank you)</b></p>	
<p><b>14. Which types of programming questions do you give your learners in order for them prepare for tests and/or examinations?</b></p>	<p>We do types of questions if I'm uh, oh if they are preparing to take a test because they could be two, I guess there could be different types I could give an, a type of a question that looks for an understanding of syntax, I don't like those, I see quite a bit of those, I want to give them the type of a question that forces them to think, given a real problem write a small code snippet not the entire thing, from maybe the entry point to where the program ends and we'll say write an algorithm to implement this. I'm giving them a statement let's say I want to sort numbers from blah to blah maybe in reverse order, all of those kinds of things you can write because, memorizing syntax is not good if they don't understand how to use it but by giving them a small problem I'm forcing them to think about all, here's how this and that I learnt of, how to do sorting, how to do our order numbers, how to manipulate I guess data in several ways so again to answer your question I would like to, I prefer to give them</p>

	<p>small challenging questions where they need to use their knowledge to do it rather than say if I “<math>8x = a</math>.a what’s the answer?” that’s not good.</p>
<p><b>15. Do you think programming can be learned in isolation without social interaction?</b></p>	<p>It’s hard to do it that way, it goes back to what I said earlier, if you relate it to something that they already know it’s easier for them, they’ll say “oh this is how I do it”. An example I want to say, send an email, they send emails and SMS messages all the time but how does that actually happen? How do you do that? As soon as I type in a text message to my friend and say send what happens on the other side? And then there’s programming because I have to get the data from this screen, package it, maybe created some type of an object if you use OO or create a package and then you use help to push it somewhere, so if you teach them from a perspective of what they already know then I want them to go behind you just, I guess the interfaces say what happened? I’m curious as to how that happens behind the scenes.</p>
<p><b>(Thank you)</b></p>	
<p><b>16. What kind of teaching support material do you normally use when teaching Programming?</b></p>	<p>Oh there’s a ton of them especially available of the internet there are videos, videos help a lot there are blogs also, when you use stack overflows, I like stack overflows because that’s where people post questions, they have problems and sometimes I have a problem</p>

	and then let me go see what some of the other people have actually said about that particular I guess issue but I can cover the material and then watch all of these videos and then also you could uhm maybe some of the free, how does it, sort of like a mooc but they don't have to, Udacity for example is one of the moocs they do provide free resources to augment some of the stuff that I would actually use.
<b>(Udacity?)</b>	<b>(U-DA, Udacity)</b>
<b>(Udacity, oh yes)</b>	Coursera and the rest of them.
<b>(Coursera yes)</b>	
<b>17. Do you invite resource personnel during some of your lessons?</b>	What do you mean resource personnel?
<b>(A person that's an expert in C++)</b>	Uhm I haven't done that
<b>(That's fine)</b>	I haven't done that
<b>18. How often do you allow your learners to discuss in groups during programming lessons?</b>	
<b>(I'm just touching on that group aspect)</b>	Oh all the time
<b>(All the time)</b>	They need to do that because one of the things that I tell my students because I was in the development field for many years and did academics I think when we talk before is that when you start working you never work

	<p>alone, it's not an isolated environment unless you're a researcher even researchers collaborated to some degree but in the workplace you always are going to work as part of a team so it's important for them to get used to that now before we actually send them out because if we don't do that they get hired, here's your desk, here's your computer oh lets go have a meeting we need to discuss this project, they'll just sit there and do like they do in the classroom just sit there and don't say anything but it is important to make them get used to that environment.</p>
<b>(You already touched on this)</b>	
<b>19. What are some of your observations during group discussion lessons?</b>	
<b>(I think you mentioned that)</b>	<p>Yep, type A personality types take over, the reserved ones get left behind.</p>
<b>20. Do you send your learners out during some of your programming lessons to explore real work related scenarios?</b>	<p>Yes uh case studies are very important uh you may, maybe watch a video and respond to what they saw or go online and read somebody's blog or maybe an introduction to something new which happens all the time read about it, again when I teach them to, to write code I don't want to teach them in isolation that okay it's me and the computer I write this code, you have to also interact because the code that you write you're writing it for other people actually use so if you look at how people use the applications</p>

	<p>that we actually build then it gives you a better sense because (oh) the other thing that's important, if I develop an application and put it out for somebody to use, I don't, I want my students to think about how the person is actually going to use that application, because if they struggle it tells me that I didn't code it correctly or if it generates an error, or it generates an error it looks very cryptic to the end user so that's why a fault, a problem, so I want them to go out there and actually observe uh, again I'm just belabouring the point, if you build something you put it out there, so I used to challenge them, so you, run this application that you just built for me and then I say do this, go in here and enter 12345, xxxx, kkkk, enter, see what happens the application crashes and then I say "Alright what do you have to say about that?" so you have to make them think.</p>
<p><b>(Yes)</b></p>	<p>Not just about the code itself, learning syntax again I'm just giving you all my answers, we will agree that programming languages all do the same thing they only differ in syntax, all of them they do exactly the same therefore you can pick up a book learn syntax and still do the same thing, if you think file processing you open a file, read, you write, it updates it closes. It doesn't matter whether it's Java, codes or anything, they all do the same thing just different syntax.</p>

<p><b>21. Give me some of the topics you discussed with them this year which compelled you to send them outside the classroom?</b></p>	<p>Uh here I haven't since I've, I'm new but what I did uh when I was dealing with projects I designed five projects to select from and then what I wanted them to do is go out there and let's say if I'm writing, uh I want them to develop an application for a small medical practice, go to a doctors, find a doctor locally and these people are always busy, explain to them that you're trying to build this application blah blah blah would they be interested, would have time to sit down so you can actually ask them questions, because I'm saying when you build an application you don't build it for yourself you build it for other people so you need to get input for them as to what do you want, what would you like to see because I remember years ago IBM used to be notorious for telling you that you want what we build for you, we're not interested in what you want.</p>
<p><b>(What you want, yes)</b></p>	<p>And that actually wasn't good, so for my students I want them, there was a group that built a fleet management system they actually went to a trucking company and interacted and asked all of these questions because for me when I was creating projects for them I said innovations will come from you I'm not going to tell you, I won't say step one do this, I'm all I did was say here's a big problem find a solution and how you are</p>

	<p>going to code it uh I just want components but I do want you to use three-tier architecture but your implication is up to your imagination but then again to answer your question send them out there to talk to people who are going to be reading, using what they are going to build.</p>
<p><b>(Okay thank you, so you've answered this question, what do you like about the lessons which normally make you send your learners outside the classroom, you touched on that thank you. It's like you know what I'm going to ask next... alright)</b></p>	
<p><b>22. Describe one of your interesting lessons and what makes it so interesting?</b></p>	<p>Uh which one can I think of, I tend to have uh</p>
<p><b>(Something that sticks out of your mind)</b></p>	<p>Uh, which one, I tend to, uh okay one of the interesting ones in Programming it was, it pertains to ethics and security but it's part of programming you see I want them to think about that, what I said was uh you, oh you go to uhm, this case I use came up a long time ago and then I use it in different courses, yes okay you have a person that goes to see a doctor and uh it's a man in this case, it's not what do I say, it's no embankment of men but we men tend to maybe let's say men in general don't like going to doctors so the guy goes to do a physical and then they do all kinds of tests and finds out that he has, well</p>

	<p>in this case let's say well he's HIV positive but he's afraid to tell his wife because of lots of different implications and then the wife suspects that uh oh well she asks him what did the doctor say, no the doctor said everything is fine I just need to lose a little more weight, what he always tells us to do, and then uhm, anything else no no, but then she suspects something is not, he's not telling me the truth so she goes to the medical practice and then she says uh my husband uh blah blah blah he came in to you but I have a feeling that he's not telling the truth, can you locate his medical charts and tell me uh I guess tell me what's there and then you as a nurse you and look but you know you see the status and then you know the implications of him hiding that from her but then there's a dilemma should you tell her or should you not and then I throw it out there to students and then I say see that's a problem because when you're coding you should think of sensitive information, how do you protect sensitive information, who should have access to it, encryption you can encrypt all of the data or some cases you may have multiple levels of access in other words you don't have one person that can just go and uh log in just like you're signing cheques that involve millions you may have multiple signatures in order to say now you have access to the money when it comes to programming you have to think about the</p>
--	---



	<p>security of the data, how would you write the code to actually make sure that no one person has to have uh two log-ins or three log-ins before you can actually get to that, so it becomes a Programming exercise to them but the, the, the social impact of making sure that you can do programming that way is a story that I tell because it's still there in their mind, because otherwise if I say well "Should you have one or multiple log-ins before you can see data?" "Uh no no".</p>
<b>(Thank you, that's fine)</b>	
<b>23. How do you help your learners to maintain a positive attitude towards programming?</b>	Make it fun.
<b>(Make it fun?)</b>	Make it fun, give them things just like I'm doing now I'm thinking of how do, if I'm introducing IOT (Internet Of Things) to them it has to be these little gadgets that you see.
<b>(Internet Of Things)</b>	Okay
<b>(Sorry)</b>	No, no that's fine
	It has an operating system that runs it, you say okay let's get its open source, go tweak it then and upload it and make it do different things because all you want to do is generate an interest in students, say computers by themselves are not good until there's software that actually drives them if you can

	<p>make a student make or maybe create a nice, pretty, animated something, whatever that may be it may be saying “oh wow that’s interesting” and then they begin to think about things beyond just that interesting exercise that they did in the classroom.</p>
<p><b>24. What are the concepts that you consider difficult to teach in Programming?</b></p>	<p>Uh the module uh Programming can be uh one of them or the, uh how, uh several of them with loops they are usually not, I would say maybe the, the, the control flow logic it tends to be, because students sometimes they think in a top-down approach because when you get that data you have to make decisions now we’ll have to write codes that breaks here and here and here and some of them have a, a, a difficult uh time understanding how do I structure the how the code actually flows from top to bottom, years ago people used to do what we call this thing top-down code right what was there they start from the top and then they do a go-to bridge around this, and then do this and go somewhere and then in the end they don’t get a clear view of what the picture looks like the other thing in order to alleviate that what we tend to do is to teach them uh what we call flow charting, draw these little boxes that’ll give you a clear view of what your program will look like so that when you go start coding then you’ll know I’m here, I’m here, I’m here, so I would say that control flow tends to be a problem for some.</p>

<b>25. Which level of programming do you prefer to teach?</b>	
<b>(First, second or third year)</b>	I prefer all of them well I'll say this...
<b>(If you had to narrow it down)</b>	Yeah, well most of the time I tend to deal with senior students
<b>(Senior? Okay)</b>	And then I would say you know if I had studied with these students maybe where I'm seeing them struggling I would have actually addressed that when I started with them at first year level, that's why I say I tend to...
<b>(Okay)</b>	
<b>26. What can you say about the learner's performance?</b>	
<b>(Well you already said they struggle even at senior level)</b>	Yeah even at senior level, there were part of it which is what I'm trying to think about how to address in here is that I haven't been long here enough to maybe interact with uh I guess lecturers that teach programming so that I can look at, are they paying attention to where things are going in the industry not just what's in the textbook for example if you look at C sharp, C sharp is up to version number eight in terms of the syntax in the newer ones if you go most of the textbooks are maybe stuck at C sharp version four but we are at eight. Eight, if you look at all of the different syntaxes being introduced uh who's

	<p>one of the designers I tend to, he has a blog he gets videos where he, they sit in a room and say okay what can we do to improve the language, maybe uh reduce complexity but they end up introducing more complex things but then if you look at the industry at work programmers, because I worked as a programmer, we tended to look at what's the latest uh in terms of the language, uh I think with six version six they introduced new things uh then we start to use it immediately because we are experienced but what I'm saying is if you look at the university the lecturers looking at the new things that are designed to make the language a little bit easier to use or are we still teaching them version 1.0</p>
<p><b>(Okay)</b></p>	<p>Because if we are then that's bad when they come up they are going to be way below in terms of where they should be.</p>
<p><b>(So you've already answered the next, what are possible causes for the learners' performance, again you pre-empt what I'm saying)</b></p>	<p>(No)</p>
<p><b>(That's fine, so if I could just summarize the possible causes would be that we are not in touch with the version of software that we are using)</b></p>	<p>Yeah, look at the research</p>

(Yes)	<p>'Cause languages evolve all the time so if your textbook unfortunately can't keep up because we're always introducing new things as well if we do that, that textbook is going to be out of date, so I would say that I can pick up any textbook and I can tell you that most of them now discuss C sharp seven not even eight.</p>
(Okay)	
<p><b>27. How do you prepare yourself before any Programming lesson?</b></p>	<p>Uh I need to look at what the chapter is that I'm talking about, think of examples use tons and tons of examples, deliver to the students and then you need to draw them because again Programming is not just about learning a syntax you have to put things in context where would I even use, let's say I'm looking at the IF statement where would I use IF statement because then I would introduce them in such a way to students, give them a short narrative then I'd say if I'm giving different choices where I could choose choice A, or choice A, B, C or D. How do I actually write that? Why would I want to do that? I'll say I'm giving them an example I'm writing a code for insurance somebody's going to come in they want to uh get an insurance policy, there are different policies that we actually sell. It could be auto car insurance, could be health insurance, could be Pat insurance, could be all of these other things so in my code I'm going to say if they</p>

	want this do this, if they want to do this, do that.
<b>(Okay)</b>	
<b>28. Do you prepare lesson plan every day, weekly or monthly?</b>	No I prepare them maybe just weekly because I've been doing this...
<b>(For so long?)</b>	Yeah a very long time.
<b>29. Are you comfortable with every topic in the Syllabus?</b>	Uh yes I am.
<b>(Yes)</b>	
<b>30. What strategies do you use during Programming lessons?</b>	
<b>(I think you've answered that, you use examples, you prepare beforehand, you use blogs uh videos and uhm resources from the internet.)</b>	Yep
<b>31. How do you teach a new concept?</b>	
<b>(Again that was answered, thank you)</b>	(Agrees)
<b>32. What are your questioning styles?</b>	Are you referring to verbally or...?
<b>(You tell me, you're in a classroom and you need to expand a concept, how would you question your students?)</b>	Uh yeah I like to always give context for example I don't like questions like I said that tends to want to understand if somebody knows the syntax or not, if I say what is a variable, I wouldn't ask a question like that because it serves no purpose I may ask

	<p>maybe if you say I want to get the same answer, but I'll say in the computer program uh you have to store, store data somewhere within the program itself that you refer to, update, or what would you call where you actually store that, right. See I give it a little bit of a concept or the other way to get it out of a student I would ask them to give a description of a process so that I want to know if they understand all the things that would need to occur for that code to I guess to be accomplished and then in terms of programming I always like to give them uh maybe I could give them a short program and then base the questions on that so they would need to, sort of run the application, the program in their head in order for them to be able to answer the questions. I think that's uh...</p>
<b>(No, that's fine)</b>	That tends to be effective.
<b>33. How often do you give tasks to your learners?</b>	As much as I can.
<b>(Okay)</b>	Uh constantly, constantly.
<b>34. Are you satisfied with your learners' performance after giving them tasks? If no, how do you make them improve it?</b>	Uh as usual those who grab things pretty quickly not a problem, one, two, three they are done the, the, the problem tends to the people who either they don't or didn't understand uh what they were supposed to do, some of them are just not going to do it so there isn't much you can do in those

	<p>except maybe to try and understand as to why are they not even interested in it sometimes it could be maybe social issues, family issues, all of those things that may not rather than focus on school or focus on I guess society who've got problems even those, the thing that, even though it does take time is to put a name, come to my office let's just chat and see I'm a little bit concerned about your performance in class, you didn't do the homework, why didn't you do the homework or you didn't do very well is it that you don't understand some of these concepts if you don't I can spend extra time explaining to you, if not maybe you can get somebody to, who can refer you to a tutoring services where you can actually seek help again the ones that know fly through, sometimes I even use them other students within the course itself, I'll say okay can you help so and so partner with them, don't do their work just make sure that they understand what it is that gets done so I would say there's different ways you can actually deal with but the problem here in South Africa is our classes are huge we have three hundred students in a classroom it's going to be virtually impossible if maybe fifty percent of them are underperforming but then uh part of it maybe is more of a University issue is we have to reduce class sizes to where they are manageable therefore the interaction is going to be much higher than with a big class.</p>
--	--



<b>(Okay)</b>	
<b>35. How often do you attend in-service training /workshop in programming?</b>	Oh I do, there's con uh I attend conferences and I follow certain uh authors that I know in the field I'm always interested to see what it is that they are actually doing when they do uh at a conference they present either a paper or a workshop I always want to go there and participate because then you can ask them certain questions.
<b>36. Do you give feedback to your learners regularly?</b>	
<b>(I think you already touched on this, where you check on them and you call them in.)</b>	Yeah you have to.
<b>37. How many forms of assessments do you conduct?</b>	
<b>(I'm talking like theory, practical, oral)</b>	Uh usually it's the practical...
<b>(Practical)</b>	that I rely on, again I don't like to give them what we call toy programs because you have to make them think, you have to challenge them, you can give them maybe, some, the ones, if I wanted to force them a little bit further I would say here for extra credit, here are the harder problems to actually challenge uh some of them. The problem with that idea is that the ones that are doing well they'll always do the extra points ones it's the ones

	that are struggling they won't even attempt that.
<b>38. How many questions would you give in a practical assessment?</b>	Not a lot but each one, let's say for example if I give them five, again I don't actually do those little ones it's going to be a problem maybe it's five it may include you do six or seven things that you need addressing within problem one, problem two, problem three and problem four
<b>(Thank you, I think you've addressed the next one, do you prefer practical/theory assessments and why, you just said that it's practical)</b>	Practical, yeah because at work nobody is going to quiz your theory they want you to do, show it to me actually one of the things that uh we were doing, most companies in the US now because I went at a (blank) when potential programmers came in we didn't ask them questions we gave them a problem to solve, they would say here's the room, here's the computer's it's got all the software that you can ever want if there's something you want we don't have it we'll go load it but here's the problem that we want you to solve and then you'll have to go and design and build and do that and then you'll come show it to us so nobody's going to start to ask you uh theoretical questions tell me what a loop does, no we don't do that anymore.
<b>39. What support have you received from fellow academics, management in your dept. and from surrounding ToTs?</b>	Uh I'm relatively new here so I haven't interacted with a lot of people yet.
<b>(That's fine, fine, no problem.)</b>	

<b>40. Do you use pair programming?</b>	No.
<b>(No?)</b>	No.
<b>41. Are you currently experiencing any further challenges?</b>	
<b>(As an academic teaching programming)</b>	<p>Uh yeah the challenges are environmental if I may say the problem, I try to think of how do you solve this problem because many years when I was a student in here I behaved exactly like students behave today even though it's some decades later. I, the challenge is students tend to come in the classroom and just sit and listen to you, there is no interaction in the classroom it may just be my experience, I went to this workshop yesterday where this professor from Germany he was complaining about the same things "Why are students so timid?" so reserved, I should say so reserved they don't want to say anything in the classroom, because part of it is to engage them we have to dialogue, it's programming, there's a lot that we can actually talk about you know, but if I had to do it this way, this way there it would be coming from a student, what impact would it have on the program? Would it run fast or run slower? Our students in here no then you point at them they say, no, no, no not me so that is a challenge because if you look at it, in America students are not like this they are very vocal in the classroom, not that they are disrespectful, they will engage</p>

	<p>you so when you go into the classroom there you better be prepared because they won't just sit there and listen to you, they will go back and forth, back and forth so for us I wish there was a way we can get our students that thing to say you are a teacher, I am a student I'm just listening to you, you don't have to leave here from me I do want to hear from them.</p>
<p><b>(Alright, you actually answered the next one, is there a difference with regard to learners' attentiveness/abilities/background I etc., so thank you, you have)</b></p>	<p>Yes.</p>
<p><b>42. In your opinion when the learners are able to interact with their peers, does this affect their learning?</b></p>	<p>That's what we want.</p>
<p><b>(I think you've touched on this, not just with their peers with their staff)</b></p>	<p>I don't want them to look as though I'm some sort of authority figure I'm your teacher so ask me questions.</p>
<p><b>43. In your opinion how did programming impact on learners' interest, enthusiasm and motivation, especially the lower achieving ones?</b></p>	
<p><b>(You touched a little on this if you want to expand...)</b></p>	<p>Yeah I think if you do something exciting where they say "oh wow!" or maybe teach them I don't know like somebody said I'm going to be using this digital assistance where you can talk to a device if you don't want to talk there's Alexa over there, Amazon and Echo. I can write, I want to</p>

	<p>teach them how to write the Amazon skills, let's call it skills, short snippet of code you'll upload and then I can talk to the thing, back and forth, back and forth, if they can say oh wow that's how I can make this thing do something that's when I can say, well how do I use this, end users in a house where somebody is maybe is disabled because this thing can be hooked up to ten with the lights are on, I can say let's turn the lights on, poof!</p>
(Yes)	<p>And then when they do this they say, "wow I can do this, I want to do this Programming thing".</p>
(Okay thank you, you've answered the next one, in what ways if any does programming address the following modalities of learning: visual and auditory, you gave the example of Amazon and Alexa)	<p>Yeah.</p>
(The last question)	
<b>44. What advice/recommendations would you give to a teacher who is a novice "Programmer"?</b>	
(Just starting of)	<p>You starting of uhm, the advice be prepared to begin with, at the same time be prepared and don't try to sugar on everything to students especially let's say first years they are coming in they have never done Programming before so it's going to be</p>

	<p>difficult to let's say okay today maybe in the syllabus it say's we are teaching chapters one, two and three and then you want to go one, two and three and you can tell when you are getting through to students and when you are not so rather say well I don't want to fall behind because I do this all the time if I don't feel like I need to spend more time on this chapter I know I have other chapters to do it's fine I can catch up on the other ones but I want to make sure I get through on this one, so for a novice teacher is that you need to be patient, be prepared and don't sugar on things. Get a level of their engagement, are they getting, ask questions again which is a challenge for everybody, if I'm asking questions in a classroom nobody wants to raise their hand and say of so if you can try and find a way to make the students feels comfortable talking to you uh you make them comfortable even coming after well after a classroom I usually do pack up and leave I'll stay there because I'm sure there's students who want to ask the questions but they don't want to do this in front of the whole class but if I'm there and then they'll come and have this question okay sometimes I may say later I'll be in my office just come and talk to me later, if you can build that ability for students to trust and uh don't have a problem approaching you then you have a better chance of being successful when you go into the classroom because they are no longer</p>
--	---

	<p>looking at who are you but they are want you to hear more from you about certain things, the other thing that I do in the classroom I try not to make the material so dry that it's about this chapter and nothing else because you'll lose them what I would do is put in some adopted stories around these things like an example I used, 'cause for me when I drive around I look at things and say "oh wow that'll be interesting to students because it's a problem". The other day I was talking to Dr. Singh so even this IT I was at a gas station I see them using these dipsticks to take readings I guess so they know how much gasoline is in the tanks underground and I said that is old nobody does that anymore, I take my camera I take a picture I said this would be interesting to point out to students so I that if I go into the classroom I said, I would say put it up and say "what's wrong with that?" "What are these people doing?" maybe I would assume they have seen it or I would say maybe management at the end of the day wants to see how much gasoline is in the tank so they use a dipstick and do all of this I'll say no, there's sensors over here that can tell you all of that you don't need to do that so then for them I'll say now, let's go solve this problem so they say management at that gas station calls and says I know you're studying this IT, it means people are still dipping these things, how can you modernize this? What can you do? Then</p>
--	--

	<p>I let them go build this machine I won't dictate them but for them it's like it's not interesting now I was letting this in class but now I see the problem out there, if you look you see problems all over the place, as an IT person you will always find problems you may not have the solutions but take your phone you see something interesting I do this all the time, I sit back and say that does make a difference if you still do things that way so if you can get students interested again by showing them these things around them because we're not learning programming because we just like computer programming, we use, we are building a skill that allows us to build applications that solve problems that are out there.</p>
<p><b>(Okay thank you Progress, that concludes the interview)</b></p>	<p>Not a problem.</p>



## Appendix D: Request for Gatekeepers Permission

---

### REQUEST FOR GATEKEEPER'S PERMISSION

6 August 2016

Professor S Moyo  
Interim DVC – Engagement/Director – Research and Postgraduate Support  
Durban University of Technology

Dear Professor Moyo

### RE: PERMISSION TO CONDUCT RESEARCH AT THE DUT

Permission is kindly requested from the DUT to conduct my research study. I am currently studying towards a PhD degree in the School of Education at the University of KwaZulu-Natal (UKZN). As part of the requirements of this degree, I am required to complete a research thesis. This study focuses on teaching and learning programming. This research study is entitled "*An Exploration of the teaching and learning of Information Technology Programming in two higher education institutions in KwaZulu-Natal*".

The research target group is first year introductory programming students & academics teaching programming to participate in this research. I would be very grateful if you would consent to these students participating in this study. All discussions, interviews and dialogues with participants will be audio recorded using a Dictaphone, and thereafter transcribed verbatim to produce transcriptions. This research information (data) is required for the analysis of data and completion of the actual write up of the thesis. Collecting research information for this study will take approximately four to six weeks. All group discussions, in depth individual interviews and group and individual activities will take place at the DUT, IT Department.

I also wish to bring to your attention that I was one of the DUT/SANTRUST Pre-doctoral candidates to successfully complete the programme in 2014. To this end I am aware that one of the conditions of the programme is to complete the doctoral degree within a prescribed period. Gatekeeper's permission is kindly sought to enable me to fulfill that condition. The granting of the Gatekeeper's Permission will also enable me to apply for ethical clearance at UKZN as the Ethics Committee will only review my application once I have the necessary Gatekeeper's Permission.

I may be contacted at:  
Email address [prinaving@dut.ac.za](mailto:prinaving@dut.ac.za)  
Tel: 0845743807

My supervisor's contact details are:  
Email address Dr Vimolan Mudaly  
Tel: 031 2603697

If you would like any further information or if you are unclear about anything, please feel free to contact me at any time. Your co-operation and consent will be greatly appreciated.

Warm regards  
Yours Sincerely



T.P Govender (Prinavin)  
PhD Candidate  
School of Education  
University of Kwa-Zulu Natal

## Appendix E: Letter of Permission



*Directorate for Research and Postgraduate Support  
Durban University of Technology  
Tromso Annexe, Steve Biko Campus  
P.O. Box 1334, Durban 4000  
Tel.: 031-3732576/7  
Fax: 031-3732946  
E-mail: [moyos@dut.ac.za](mailto:moyos@dut.ac.za)*

30<sup>th</sup> August 2016

Mr Thamocharan Prinavin Govender  
c/o School of Education  
University of KwaZulu-Natal

Dear Mr Govender

### **PERMISSION TO CONDUCT RESEARCH AT THE DUT**

Your email correspondence in respect of the above refers.

I am pleased to inform you that the Institutional Research Committee (IRC) has granted provisional permission for you to conduct your research "An Exploration of the teaching and learning of Information Technology Programming in two higher education institutions in KwaZulu-Natal" at the Durban University of Technology.

Kindly note, that the committee requires you to provide proof of full ethical clearance prior to you commencing with your research at the DUT.

We would be grateful if a summary of your key research findings can be submitted to the IRC on completion of your studies.

Kindest regards.  
Yours sincerely

A handwritten signature in black ink, appearing to read 'S. Moyo', with a small flourish at the end.

**PROF. S. MOYO**  
**DIRECTOR: RESEARCH AND POSTGRADUATE SUPPORT**

## Appendix F: Ethical Clearance – MUT



02 November, 2016

Mr T.P. Govender

Durban University of Technology

Dear Mr Govender

It is my pleasure to inform you that permission to conduct project titled: "An exploration of the teaching and learning of information Technology (IT) programming in two higher education institutions in KwaZulu Natal (KZN)." has been granted.

Permission to conduct the project is granted on the condition that any changes to the project must be brought to the attention of the MUT Research Ethics Committee as soon as possible.

Good luck with your research.

Yours faithfully,



Dr. Anette Mienie

Director: Research

031 9077354/7450

[anette@mut.ac.za](mailto:anette@mut.ac.za)

## Appendix G: Informed Consent to Student



UNIVERSITY OF  
KWAZULU-NATAL  
INYUVESI  
YAKWAZULU-NATALI

### Informed Consent to Student

Dear

Ms/Mrs/Mr .....

Name of UoT .....

#### **Re: Permission to conduct a research study in your UoT**

I am requesting your permission to conduct a research study in your University of Technology (UoT). This research study is entitled:

An Exploration of the teaching and learning of Information Technology Programming in two higher education institutions in KwaZulu-Natal

My name is T.P Govender (Prinavin) and I am currently studying towards a PhD degree at the University of KwaZulu-Natal (UKZN). As part of the requirements of this degree, I am required to complete a research thesis. This study focuses on teaching and learning programming.

I would be very grateful if you would consent to these students participating in this study. They will be selected from your UoT

If you agree to this, they will be invited to respond to questions in an in depth individual interview.

All discussions, interviews and dialogues with participants will be audio recorded using a dictaphone, and thereafter transcribed verbatim to produce transcriptions. This research information (data) is required for the analysis of data and completion of the actual write up of the thesis. Collecting research information for this study will take approximately four to six weeks. All group discussions, in depth individual interviews and group and individual activities will take place at your UoT, IT department, with your permission. Times and dates will be discussed and arranged with you and the participants at a later stage. I will try to ensure that this takes place during their lunch breaks and free periods, in an attempt to avoid any disruptions during lessons. Participants will also be encouraged to eat their lunch during discussions, interviews and activities, as well as make use of the school toilet should the need arise. I will not deprive them of these opportunities, especially since I intend to use some of their free time in order to collect sufficient data for my study.

**Please note:**

- \* Times and dates of this data generation process will be at your sole discretion. I have merely presented you with an outline of what I intend to do, however you are free to make any changes and suggestions, if necessary.
- \* Participation is completely voluntary and participants have the right to withdraw from this study at any time. They will not be penalised if they choose to do so.
- \* Confidentiality and anonymity will be maintained at all times. The identity of your institution and all participants will not be revealed at any time, as pseudonyms (different names) will be used to protect everyone's right to privacy.
- \* Any information provided by the participants will not be used against them, or against the school, and will be used for purposes of this research only.
- \* Participation in this study will not result in any cost to your institution or the participants.
- \* Neither the participants nor your institution will receive financial remuneration. However costs incurred by participants as a result of their involvement in this project will be covered.
- \* This study does not intend to harm the participants in any way.
- \* Participants will be handed letters of consent which they will have to carefully read and sign, before I begin data collection.

I may be contacted at:

Email address [prinaving@dut.ac.za](mailto:prinaving@dut.ac.za)

Tel: 0845743807

My supervisor's contact details are:

Email address Dr Vimolan Mudaly

Tel: 031 2603697

If you would like any further information or if you are unclear about anything, please feel free to contact me at any time. Your co-operation and consent will be greatly appreciated. If you grant permission to conduct this research at your UoT, please complete the form below and return to me.

Warm regards

T.P Govender (Prinavin)

**DECLARATION**

I ..... (full name/s of student) of  
..... (name of UoT) hereby confirm that I understand  
the contents of this document and the nature of this research project, and I consent to  
participating in this research project.

*Additional consent*

I understand that interviews will be audio-recorded and I grant permission for this.

YES/NO

I understand that the students and the institution/UoT are free to withdraw from the research  
project at any time

YES/NO

**SIGNATURE OF STUDENT**

**DATE**

.....

.....

## Appendix H: Informed Consent to Academic

---



UNIVERSITY OF  
KWAZULU-NATAL  
INYUVESI  
YAKWAZULU-NATALI

---

### Informed Consent Letter to Academic

Dear

Ms/Mrs/Mr/Dr.....

Name of UoT .....

#### **Re: Permission to conduct a research study in your UoT**

I am writing to request your permission to conduct a research study in your University of Technology (UoT). This research study is entitled:

An Exploration of the teaching and learning of Information Technology Programming in two higher education institutions in KwaZulu-Natal

My name is T.P Govender (Prinavin) and I am currently studying towards a PHd degree at the University of KwaZulu-Natal (UKZN). As part of the requirements of this degree, I am required to complete a research thesis. This study focuses on teaching and learning programming.

I require first year introductory programming students to participate in this research. I would be very grateful if you would consent to these students participating in this study. They will be selected from your UoT

If you agree to this, they will be invited to respond to questions in an in depth individual interview. They will also be given the opportunity to participate in group and individual activities, which include programming tasks.



All discussions, interviews and dialogues with participants will be audio recorded using a dictaphone, and thereafter transcribed verbatim to produce transcriptions. This research information (data) is required for the analysis of data and completion of the actual write up of the thesis. Collecting research information for this study will take approximately four to six weeks. All group discussions, in depth individual interviews and group and individual activities will take place at your UoT, IT department, with your permission. Times and dates will be discussed and arranged with you and the participants at a later stage. I will try to ensure that this takes place during their lunch breaks and free periods, in an attempt to avoid any disruptions during lessons. Participants will also be encouraged to eat their lunch during discussions, interviews and activities, as well as make use of the school toilet should the need arise. I will not deprive them of these opportunities, especially since I intend to use some of their free time in order to collect sufficient data for my study.

**Please note:**

- \* Times and dates of this data generation process will be at your sole discretion. I have merely presented you with an outline of what I intend to do, however you are free to make any changes and suggestions, if necessary.
  - \* Participation is completely voluntary and participants have the right to withdraw from this study at any time. They will not be penalised if they choose to do so.
  - \* Confidentiality and anonymity will be maintained at all times. The identity of your institution and all participants will not be revealed at any time, as pseudonyms (different names) will be used to protect everyone's right to privacy.
  - \* Any information provided by the participants will not be used against them, or against the school, and will be used for purposes of this research only.
  - \* Participation in this study will not result in any cost to your institution or the participants.
  - \* Neither the participants nor your institution will receive financial remuneration. However costs incurred by participants as a result of their involvement in this project will be covered.
  - \* This study does not intend to harm the participants in any way.
  - \* Participants will be handed letters of consent which they will have to carefully read and sign, before I begin data collection.
-

I may be contacted at:

Email address [prinaving@dut.ac.za](mailto:prinaving@dut.ac.za)

Tel: 0845743807

My supervisor's contact details are:

Email address Dr Vimolan Mudaly

Tel: 031 2603697

If you would like any further information or if you are unclear about anything, please feel free to contact me at any time. Your co-operation and consent will be greatly appreciated. If you grant permission to conduct this research at your UoT, please complete the form below and return to me.

Warm regards

T.P Govender (Prinavin)

**DECLARATION**

I ..... (full name/s of academic) of  
..... (name of UoT) hereby confirm that I understand  
the contents of this document and the nature of this research project, and I consent to the  
students participating in this research project. I also grant permission for my UoT to be used  
as the research site.

*Additional consent*

I understand that interviews will be audio-recorded and I grant permission for this.

YES/NO

I understand that the students and the institution/UoT are free to withdraw from the research  
project at any time

YES/NO

**SIGNATURE OF ACADEMIC**


**DATE**

.....

.....

## Appendix I: CS50 online course @Harvard extension school

### Congratulations and welcome to the CS50 family

 You forwarded this message on Thu 7/13/2017 10:35 PM



Kasey Champion <Kaseyc@exchange.microsoft.com>  
Tue 1/26/2016 10:11 PM



**Congratulations!** You have been selected for the CS50 Educators Scholarship

We are so happy that you are interested in learning more about Harvard's CS50 program for High Schools, and are thrilled to offer you a full scholarship to cover the cost of both semesters of the extension school course. Here is the link to the course:

<https://www.extension.harvard.edu/academics/courses/intensive-introduction-computer-science-i/24640> I am working with the Harvard Extension School to finalize the details, so you will likely hear from them shortly on next steps.

Thanks and let me know if you have any questions,  
Kasey Champion



**HARVARD**

John A. Paulson  
School of Engineering  
and Applied Sciences

**Douglas R. Lloyd, Jr.**

Senior Preceptor in Computer Science  
Harvard University Division of Continuing Education  
lloyd@cs50.harvard.edu

20 July 2017

To Whom it May concern:

In Spring 2016 and Fall 2016, respectively, Prinavin Govender was a student in CSCI E-50a: Intensive Introduction to Computer Science I and CSCI E-50b: Intensive Introduction to Computer Science II, courses offered through Harvard University's Division of Continuing Education (Harvard Extension School). As we advise students, "[m]ore than just teach you how to program, this course teaches you how to think more methodically and how to solve problems more effectively. As such, its lessons are applicable well beyond the boundaries of computer science itself."

Among the two courses' expectations were 10 problem sets (programming assignments that typically involve 10 or more hours of work per week outside of class), 4 quizzes, and a final project. Mr. Govender took the course in conjunction with a program designed to prepare teachers to teach material covered in the course to younger students, and he performed very well in the two courses. Indeed, despite participating remotely from South Africa he was among the most active and engaged students both semesters. It was a pleasure to have him join us, and we wish him well in all his future endeavors.

Sincerely,

/s/ Douglas Lloyd

Douglas Lloyd

Course Manager, CSCI E-50a/b

---

33 Oxford Street, Cambridge, MA 02138

## Appendix J: Language Editing Certificate

**Trevor French**

Professional Editing, Proofreading  
& Copy writing

---

tregf369@gmail.com

084 326 9251

---

30 October 2019

---

### LANGUAGE EDITING CERTIFICATE

---

To whom it may concern:

I have language-edited and proofread the thesis by Thamotharan Prinavin Govender entitled:

**“An exploration of the teaching and learning of Information Technology (IT)  
programming in two higher education institutions in KwaZulu-Natal (KZN)”**

To the best of my knowledge, this work is the author’s own, and is free of spelling, grammatical, structural and stylistic errors to meet the requirements for submission to the University of KwaZulu-Natal.

With gratitude.



---

T. G. French (Mr)