UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

# Mathematical thinking skills needed by first-year programming students

by

**Carla Coetzee**

**A dissertation submitted in partial fulfilment of the requirements for the degree of MEd**

**Faculty of Education**
**Department of Science, Mathematics and Technology Education**
**University of Pretoria**

**Supervisor: Prof G Stols**

**August 2016**



© University of Pretoria

## Declaration of Originality

I declare that the dissertation, which I hereby submit for the degree MEd (SMTE) at the University of Pretoria, is my own work and has not been submitted by me for a degree at this or any other tertiary institution.



© University of Pretoria

The author, whose name appears on the title page of this thesis, has obtained, for the research described in this work, the applicable research ethics approval. The author declares that she has observed the ethical standards required in terms of the University of Pretoria's "*Code of ethics for researchers and the Policy guidelines for responsible research".*

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

RESEARCH ETHICS COMMITTEE

| | |
|---|---|
| **CLEARANCE CERTIFICATE** | CLEARANCE NUMBER: SM 15/10/01 |
| **DEGREE AND PROJECT** | MEd<br>Mathematical thinking skills needed by first year programming students |
| **INVESTIGATORS** | Carla Coetzee |
| **DEPARTMENT** | Science, Mathematics and Technology Education |
| **APPROVAL TO COMMENCE STUDY** | 08 March 2016 |
| **DATE OF CLEARANCE CERTIFICATE** | 17 August 2016 |

Please note:

For Master's application, Ethics Clearance is valid for 2 years
For PhD application, Ethics Clearance is valid for 3 years

**CHAIRPERSON OF ETHICS COMMITTEE:** Prof Liesel Ebersöhn

**CC** Bronwynne Swarts
Gerrit Stols

This Ethics Clearance Certificate if issued subject to the following conditions:

1. A signed personal declaration of responsibility
2. If the research question changes significantly so as to alter the nature of the study, a new application of ethical clearance must be submitted
3. It remains the student's responsibility to ensure that all the necessary forms for informed consent are kept for future queries

Please quote the clearance number in all enquiries



© University of Pretoria

Tshwane University of Technology
*We empower people*

## Research Ethics Committee

*The TUT Research Ethics Committee is a registered Institutional Review Board (IRB 00005968) with the US Office for Human Research Protections (IORG# 0004997) (Expires 9 Jan 2017). Also, it has Federal Wide Assurance for the Protection of Human Subjects for International Institutions (FWA 00011501) (Expires 22 Jan 2019). In South Africa it is registered with the National Health Research Ethics Council (REC-160509-21).*

April 27, 2016

Ref #: REC/2016/04/002
Name: Coetzee C
Student #: 14320917 (UP)

Ms Coetzee C
C/o Prof G Stols
Department of Science, Mathematics and Technology Education (SMTE)
Faculty of Education
University of Pretoria

Dear Ms Coetzee,

**Decision: Final Approval**

---

**Name:** C Coetzee
**Project title:** *Mathematical thinking skills needed by first year programming students*
**Qualification:** M Ed, University of Pretoria
**Supervisor:** Prof G Stols

---

Thank you for submitting the revised project documents for ethics clearance by the Research Ethics Committee (REC), Tshwane University of Technology (TUT). In reviewing the documents, the comments and notes below are tabled for your consideration, attention and notification:

- **Permission Letter**
  - **Head of Department.** The permission letter to access previous exam papers from Dr M Lall (Head of Department: Computer Science; dated 25 Apr 2016) is in order and duly noted.
- **Proposal**
  - **Document Analysis (Section 9.3.1).** The clarification that TUT will be the only university targeted for the document analysis phase of the study is in order and duly noted.





© University of Pretoria

- **Information Leaflet and Informed Consent (Assessment & Interviews), Moderators and Examiners**
  - **TUT Logo.** The replacement of the TUT logo with the University of Pretoria logo is in order and duly noted.
  - **Examiners and Moderators.** The REC took note that all the target moderators are TUT staff members (i.e. internal moderators).
  - **Declaration: Conflict of Interest.** The revised Conflict of Interest Declaration is in order and duly noted.
  - **"Has the study received ethical approval?".** The addition of a statement regarding TUT Research Ethics Committee approval is in order and duly noted.

The Chairperson of the Research Ethics Committee, Tshwane University of Technology (TUT), reviewed the revised project document on Apr 27, 2016. **Final approval** is granted to the project.

The proposed research project may now continue with the proviso that:

1) The researcher/s will conduct the study according to the procedures and methods indicated in the approved proposal, particularly in terms of any undertakings and/or assurances made regarding the confidentiality of the collected data.
2) The proposal will again be submitted to the Committee for prospective ethical clearance if there are any substantial changes from the approved proposal.
3) The researcher/s will act within the parameters of any applicable national legislation, professional codes of conduct, institutional guidelines and scientific standards relevant to the specific field of study. Strict adherence to the following South African legislation, where applicable, is especially important: Protection of Personal Information Act (Act 4 of 2013), Children's Act (Act 38 of 2005) and the National Health Act (Act 61 of 2003).
4) The current ethics approval expiry date for this project is **June 30, 2018**. No research activities may continue after the ethics approval expiry date. Submission of a duly completed Research Ethics Progress Report (available at: http://www.tut.ac.za/Other/rninew/ResearchEthicsCommittees/Pages/default.aspx) will constitute an application for renewal of REC ethics approval.

*Note:*
*The reference number [top right corner of this communiqué] should be clearly indicated on all forms of communication [e.g. Webmail, E-mail messages, letters] with the intended research participants.*

Yours sincerely,

WA HOFFMANN (Dr)
Chairperson: Research Ethics Committee
[Ref#2016=04=002=CoetzeeC]

*We empower people*



© University of Pretoria

# Acknowledgements

Without the following people, this journey would not have been possible and I gratefully acknowledge them:

- My supervisor Prof Gerrit Stols for motivation, guidance, patience and academic support.
- My mother-in-law, Dr Miemie Fourie, for sharing your years of experience in supervising qualitative research with me. You were a wonderful soundboard when I needed inspiration.
- My father, Dr PJ van Zyl, thank you for your example of dedication and hard work in academia. Thank you for all the unconditional love and support you and my mothers (late Lena) and Heyla gave us through the years.
- My husband, Braam Coetzee, for your unconditional love and support and for taking the household duties off my hands to have time for my studies. Thank you for proofreading my dissertation again and again.
- My sons, Wicus and Ruan Coetzee, for all the coffee, tea and water that you carried to my study. Without your love and support I would not have been able to study. You make me proud!
- Thank you to all our friends and family who supported me and my family.
- Thank you to my colleagues at TUT who took part in interviews and were willing to share their experience as well as examination papers with me for analysis.
- I would also like to thank my employer, TUT, for supporting me with study leave and allowing me to do the data collection for my research at TUT.



© University of Pretoria

# Abstract

The aim of this qualitative study is to explore and describe the mathematical thinking skills that students require for a first level programming subject that forms part of the National Diploma in Information Communication Technology (ICT) at a University of Technology (UoT).

Mathematics is an entry requirement for many tertiary programmes, including ICT courses, unfortunately the poor quality of schooling in South Africa limits learners' access to higher education. From the literature, it is evident that students lack fluency in fundamental mathematical and problem-solving skills when they enter higher education.

In this study, the concept of programming thinking skills is explored, described and linked to mathematical thinking skills. An instrument (Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessment) has been developed and used to analyse examination papers of a first-year programming subject (at TUT) in order to identify mathematical skills as these appear in programming assessments. Semi-structures interviews were conducted with first-year programming lecturers, examiners and moderators. The literature as well and the results of the analysed data indicated and confirmed that mathematical thinking skills are extremely important when learning to program. The results of the study indicate a strong relationship between mathematical thinking skills and programming thinking skills.

The outcome of this study is therefore a set of mathematical thinking skills that needs to be developed when compiling a mathematics curriculum for first level programming students studying towards a National Diploma in ICT.



© University of Pretoria

## Table of Contents

**Chapter 1: Introduction and Contextualisation ..... 1**

1.1 Introduction ..... 1

1.2 Rationale ..... 2

1.3 Problem Statement ..... 3

1.4 Purpose of the Study ..... 4

1.5 Research Questions ..... 4

1.5.1 Primary Research Question ..... 4

1.5.2 Secondary research question ..... 4

1.6 Methodological Considerations ..... 5

1.7 Concept Clarification ..... 6

1.8 Possible Contributions of the Study ..... 6

1.9 The Structure of the Dissertation ..... 7

**Chapter 2: Literature Review and Conceptual Framework ..... 8**

2.1 Introduction ..... 8

2.2 Thinking skills ..... 8

2.3 What are Mathematical Thinking Skills? ..... 13

2.3.1 Development of Mathematical Thinking Skills ..... 14

2.3.2 The Relationship between Problem-Solving and Mathematical Thinking Skills ..... 15

2.3.3 Elementary vs Advanced Mathematical Thinking Skills ..... 16

2.4 Thinking Skills for Programming ..... 17

2.5 Linking Mathematical Thinking, Mathematical Proficiency and Programming Thinking ..... 24

2.5.1 Manipulation of formulae ..... 26

2.5.2 Making use of mathematics to represent realities ..... 27

2.5.3 Behaviour of function values ..... 29

2.5.4 Breaking down and simplifying problems ..... 30

2.5.5 Dealing with infinity ..... 31



© University of Pretoria

2.5.6 Generalisation of concepts/problems ....32
2.5.7 Abstract reasoning ....33
2.5.8 Information structures ....34
2.5.9 Algorithmic thinking ....35
2.6 Conceptual Framework ....36
2.6.1 Introduction ....36
2.6.2 Problem-solving, reasoning skills, algorithmic thinking in programming ....36
2.7 Summary ....39
**Chapter 3: Research Methodology .... 41**
3.1 Introduction ....41
3.2 Paradigmatic Perspectives ....41
3.3 Interpretive Paradigm ....42
3.4 Research Design ....43
3.4.1 Selection of Participants and Sampling Procedures ....44
3.4.2 Data Collection and Documentation ....46
3.4.3 Document Analysis and Content Analysis ....46
3.4.4 Interviews ....48
3.5 Data Analysis and Interpretation ....50
3.5.1 Trustworthiness and Credibility ....51
3.5.2 Inter-rater reliability ....51
3.6 Ethical Considerations ....53
3.7 Data Collection: Document Analysis and Content Analysis ....53
3.8 Data Collection: Interviews ....53
3.9 Summary ....54
**Chapter 4: Results and findings .... 55**
4.1 Introduction ....55
4.2 Analysis of Question Papers ....55
4.2.1 Question 1: True or False ....57



© University of Pretoria

4.2.2 Question 2: Multiple Choice .....60

4.2.3 Question 3: Predict the Output.....63

4.2.4 Question 4: Predict the Output.....68

4.2.5 Question 4: Write a complete algorithm .....71

4.2.6 Questions 5 and 6: Practical (coding in VB.Net or C++).....76

4.2.7 Results of Analysis of Question Papers .....79

4.3 Analysis of Semi-structured Interviews .....82

4.4 Summary .....97

**Chapter 5: Conclusions and Implications ..... 99**

5.1 Introduction.....99

5.2 Summary of Chapters.....99

5.3 Discussion of the Research Questions .....100

5.4 Concluding Remarks .....105

5.5 Limitations of the study .....105

5.6 Recommendations for future research .....106

5.7 Implications.....107

5.8 Reflections.....107

References .....109

Appendix A: Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments .....125

Appendix B: Solution of Question 4 of Paper 4.....126

Appendix C: Solution of Question 5 of Paper 4.....130

Appendix D: Results of Analysis of Question Papers.....136

Appendix E: Permission letter.....137

Appendix F: Information leaflet and Informed Consent (Assessments).....138

Appendix G: Information leaflet and Informed Consent (Participation in Interviews) .....142



© University of Pretoria

# List of Abbreviations

| | |
|---|---|
| **AMT** | Advanced Mathematical Thinking |
| **ANA** | Annual National Assessments |
| **DBE** | Department of Basic Education |
| **DHET** | Department of Higher Education and Training |
| **DSO** | Development Software |
| **EMT** | Elementary Mathematical Thinking |
| **ICT** | Information Communication Technology |
| **TIMMS** | Trends in Mathematics and Science Study |
| **TUT** | Tshwane University of Technology |
| **UoT** | University of Technology |
| **UP** | University of Pretoria |
| **VB.Net** | Visual Basic.Net |



© University of Pretoria

# List of Appendices


| | | |
|---|---|---|
| **A** | Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments | 124 |
| **B** | Solution of Question 4 of Paper 4 | 125 |
| **C** | Solution of Question 5 of Paper 4 | 129 |
| **D** | Results of Analysis of Question Papers | 135 |
| **E** | Permission Letter | 136 |
| **F** | Information leaflet and Informed Consent (Assessments) | 137 |
| **G** | Information leaflet and Informed Consent (Interviews) | 141 |




© University of Pretoria

# List of Tables

| | | |
|---|---|---|
| Table 1.1 | Selected paradigm | 5 |
| Table 2.1 | ICT Competencies | 21 |
| Table 3.1 | Selected paradigm | 42 |
| Table 3.2 | Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments | 47 |
| Table 3.3 | Kvale's seven stages of an interview inquiry were used for this study | 50 |
| Table 3.4 | Degree of Agreement | 53 |
| Table 4.1 | Example of Question 1.8 (Paper 1) | 57 |
| Table 4.2 | Results: Analysis of Question 1 (Paper 1) | 59 |
| Table 4.3 | Example of Question 2.8 (Paper 1) | 60 |
| Table 4.4 | Trace table to demonstrate processing of the algorithm in Table 4.3 | 61 |
| Table 4.5 | Results: Analysis of Question 2 (Paper 1) | 63 |
| Table 4.6 | Example of Question 3 (Paper 3) | 63 |
| Table 4.7 | Results: Analysis of Question 3 (Paper 3) | 67 |
| Table 4.8 | Example of Question 4 (Paper 2) | 68 |
| Table 4.9 | Trace table to demonstrate processing of the algorithm in Table 4.8 | 69 |
| Table 4.10 | Results: Analysis of Question 4 (Paper 2) | 72 |



© University of Pretoria

| | | |
|---|---|---|
| Table 4.11 | Example of Question 4 (Paper 4) | 72 |
| Table 4.12 | Results: Analysis of Questions 4 and 5 (Paper 4) | 75 |
| Table 4.13 | Example of Question 5 (Paper 4): Code the following design into C++ / VB | 76 |
| Table 4.14 | Summary of analysis of examination papers | 82 |
| Table 4.15 | Interview questions | 83 |
| Table 4.16 | Participants' roles in terms of first-year programming | 83 |
| Table 4.17 | Question 1: How long have lectured first-year programming (DSO) at TUT? | 84 |
| Table 4.18 | Question 3: Do the first years struggle with programming? If yes, why? | 84 |
| Table 4.19 | Question 4: What mathematical thinking skills do you think first-year programming students need? | 86 |
| Table 4.20 | Question 5: Does a lack of mathematical thinking skills limit their understanding of programming concepts? If yes, why? | 87 |
| Table 4.21 | Question 6: What do you think is the most important aspect of mathematics that they need for first-year programming? In other words, knowledge of which content do they need? | 88 |
| Table 4.22 | Question 7: Do you think the first-year programming students need to be able to | 90 |
| Table 5.1 | Mathematical Thinking Skills: Ordered Results of the Analyses of Four Question Papers | 101 |



© University of Pretoria

Table 5.2 Ordered Results of Analyses of Question Papers: Linking Mathematical Thinking Skills with Programming Thinking Skills 103



© University of Pretoria

# List of Figures


| | | |
|---|---|---|
| Figure 2.1 | Three Interactive Elements of Thinking | 9 |
| Figure 2.2 | Thinking and Reasoning Skills | 11 |
| Figure 2.3 | Matrix of Thinking Skills | 11 |
| Figure 2.4 | Bloom's Structure of the Cognitive Process Dimension | 12 |
| Figure 2.5 | Knowledge Dimensions of Bloom's Revised Taxonomy | 12 |
| Figure 2.6 | Mathematical Activities | 16 |
| Figure 2.7 | Mathematical Activities | 19 |
| Figure 2.8 | Programming Thinking Skills | 21 |
| Figure 2.9 | Relationships between components of ICT competencies and programming thinking skills | 23 |
| Figure 2.10 | Programming Framework | 24 |
| Figure 2.11 | Mathematical Thinking | 26 |
| Figure 2.12 | Data processing | 27 |
| Figure 2.13 | Pseudo-code example: Making use of mathematics to represent realities | 27 |
| Figure 2.14 | Mathematical function $\boldsymbol{f(x) = 2x + 3}$ | 30 |
| Figure 2.15 | Programming steps | 31 |
| Figure 2.16 | Infinite loop, repetition does not stop (error) | 31 |
| Figure 2.17 | Example of a matrix, representing data of Figure 2.18 | 34 |




© University of Pretoria

| | | |
|---|---|---|
| Figure 2.18 | Example of an array `studentArray` with student numbers and marks | 35 |
| Figure 2.19 | Algorithmic thinking competence | 36 |
| Figure 2.20 | Conceptual Framework: Relationship between mathematical and programming thinking skills | 38 |
| Figure 2.21 | Programming and Mathematical Thinking Skills | 40 |
| Figure 3.1 | Research process followed | 45 |
| Figure 3.2 | Data Collection | 46 |
| Figure 3.3 | Analysis of Question 1 of Paper 4 (Researcher) | 52 |
| Figure 3.4 | Analysis of Question 1 of Paper 4 (Participant) | 52 |
| Figure 4.1 | Example of Analysis in MS Excel | 57 |
| Figure 4.2 | Analysis of 96 questions: Mathematical Thinking Skills | 81 |
| Figure 4.3 | Analysis of 96 questions: Programming Thinking Skills | 81 |
| Figure 5.1 | Links between Programming Thinking Skills and mathematical Thinking Skills | 102 |



© University of Pretoria

# Chapter 1: Introduction and Contextualisation

"Mathematics is a living subject which seeks to understand patterns that permeate both the world around us and the mind within us" (Schoenfeld, 1992, p. 4).

## 1.1 Introduction

The aim of this study is to explore and describe the mathematical thinking skills that students require for a first level programming subject towards obtaining a National Diploma in Information Communication Technology (ICT).

Mathematics is an entry requirement for many tertiary courses, including ICT courses. However, Feza-Piyose (2012) points out that the majority of learners in South Africa demonstrate poor performance in mathematics. This situation affects many prospective students' entry into tertiary education, and therefore also limits their opportunities to enter into careers in programming. The South African education system has over time proved to produce learners lacking mathematical proficiency. In 2011, for example, South African learners performed second lowest overall of the 42 countries that participated in the Trends in Mathematics and Science Study (TIMSS) (Reddy *et al*., 2012). The South African Department of Basic Education (DBE) has also conducted the Annual National Assessments to determine learners' proficiency in mathematics. In 2013, the mathematics average of grade 9 learners in the country was 14% (DBE, 2014). The poor quality of schooling in South Africa therefore severely limits learners' opportunities in terms of further education and training (Spaull, 2013).

Locally, the South African Department of Higher Education and Training (DHET) has taken cognisance of the severity of this problem. Over the past few years, universities in South Africa have received funding aimed at presenting foundation and extended programmes. These programmes extend three-year degrees and diplomas to four years and should enable students to develop tertiary academic skills more gradually. The foundation and extended programmes are designed to improve the support and quality of teaching and learning that first-year students receive at higher education institutions (DHET, 2013). Tshwane University of Technology (TUT) is one of the universities that offer students the opportunity to enjoy the benefit of extended and foundation programmes.



© University of Pretoria

TUT offers a National Diploma in ICT, which is a three-year qualification on NQF level 6. The mathematics entry requirement for the National Diploma ICT is a 4 for Mathematics (50% - 59%). If students do not qualify for direct entry into the Diploma, they have the option to enter into the qualification by completing a foundation year before enrolling for the Diploma; this option extends their studies with a year. The mathematics entry requirement for the National Diploma ICT with Foundation Provision at TUT is a 3 for Mathematics (40% - 49%) or a 5 for Mathematical Literacy 5 (60% - 69%).

## 1.2 Rationale

Worldwide, the success rate at first-year level programming subjects is relatively low (Caspersen, 2007; Dehnadi & Bornat, 2006). Teaching novice programmers to program is a challenge, and attrition rates for first level programming courses can be as high as fifty percent (Beaubouef, Lucas & Howatt, 2001). Several authors have set out to determine the factors that influence students' success or failure in programming (Caspersen, 2007; Goold & Rimmer, 2000; Wileman, Konvalina & Stephens, 1981; Wilson & Shrock, 2001). One of the main influencing factors is that students lack the necessary fundamental mathematics skills when they enter post-school mathematic courses (Steen, 2003). In particular, the common theme identified by these authors is a lack of problem-solving skills - which could be attributed to poorly-developed mathematical knowledge and mathematical thinking skills among students entering higher education. The phenomenon of (mathematical) under-preparedness among programming students is also evident in other countries such as the USA, Australia, the UK and Ireland (Hourigan & O'Donoghue, 2007; Tall, 1995, Titus, 2010). Hourigan and O'Donoghue (2007) have identified the following problems amongst programming students in Higher Education Institutions:

- Lack of mathematical knowledge.
- Underdeveloped numeracy skills that are necessary to do basic calculations needed in their daily lives.
- Low competency levels in algebra.



© University of Pretoria

- An inability to apply mathematics in new contexts that are different from the ones they have encountered in previous exercises.
- Inadequate mathematical reasoning abilities, i.e. problem-solving skills.

The present study sets out to identify the role of mathematical knowledge and skills as demonstrated by the development of specific mathematical thinking skills needed for programming success.

Students in the ICT Foundation Course at TUT have proven to be under-prepared for the problem-solving nature of the ICT Diploma. In view of this state of affairs, TUT has introduced a new foundation subject called ICT Foundation Mathematical Skills. The objective of the subject is to bridge the gap between Grade 12 Mathematics/Mathematical Literacy in terms of the mathematical thinking skills, problem-solving skills and mathematical concepts that are required for programming. The aim of this module is to develop students' relevant mathematical thinking and problem-solving skills in preparation for the study of programming in the National Diploma. Identifying and teaching the much-needed mathematical thinking skills that are required for effective programming should benefit students and, hopefully, eventually contribute towards improved success rates in the diploma itself.

## 1.3 Problem Statement

The objective of the ICT Foundation programme at TUT is to improve students' readiness for tertiary education and also to develop their thinking and problem-solving skills. This programme constitutes an effort aimed at improving the success rate specifically of programming subjects in the ICT Faculty at the TUT.

In my experience, lecturing ICT Foundation students at TUT, students lack adequate mathematical thinking skills that are needed for them to develop into successful programmers. The assumption is therefore that developing relevant mathematical thinking skills will result in students being better equipped to complete their first-year programming subject successfully.



© University of Pretoria

## 1.4 Purpose of the Study

The purpose of the present study is to explore and describe the mathematical thinking skills that are imperative for students' success in first-year programming courses of a National Diploma in ICT.

This is a qualitative study that involves the exploration of mathematical thinking skills required for programming as set out in the available literature. What programming thinking skills entails, also forms part of this endeavour. The study aims to describe the specific thinking skills in mathematics that are relevant and essential for successful first-year programmers. In other words, the study aims to answer the question: which specific mathematical thinking skills do first-year programming students need to program? These skills will be determined in order to be able to remedy the lack of mathematical acumen and knowledge that have been identified as essential for first-year students enrolled for a first-year programming course as part of the National Diploma in ICT. The results of this study can be used to improve the mathematics modules for foundation and first-year students studying ICT at TUT.

## 1.5 Research Questions

### 1.5.1 Primary Research Question

Taking into consideration the particular set of thinking skills needed for programming, the present study is guided by the following primary question: *Which mathematical thinking skills do first-year students, studying towards a National Diploma in ICT, need?*

### 1.5.2 Secondary research question

In order to fully explore the primary research question, the following secondary question needs to be addressed:

- *Which mathematical thinking skills are related to programming thinking skills?*



© University of Pretoria

## 1.6 Methodological Considerations

In an effort to justify the choices made for the study, a brief summary of the methodological considerations is presented in Table 1.1. The research methodology is discussed in greater detail in Chapter 3.

In order to identify the relevance of mathematical thinking skills needed for programming, first-year programming question papers from TUT were selected and analysed according to the skills that have been identified. The examiners and moderators of these papers were interviewed in semi-structured interviews. The results obtained from these interviews as well as the analysis of the examination papers confirmed the identified mathematical thinking skills that are needed for programming.

Table 1.1: Selected paradigm (adapted from Wagner, Kawulich & Garner, 2012)

| **Interpretivist paradigm** | |
|---|---|
| **Why is the research conducted?** | To understand the nature and set of mathematical thinking skills that is relevant to programming. |
| **Philosophical paradigm** | Phenomenological: identify mathematical thinking skills based on analysis of data obtained. |
| **Ontological assumptions** | Many sets of mathematical thinking skills are described in the literature. Therefore, there is more than one reality (or truth). |
| **Nature of knowledge (epistemology)** | The study relies on the researcher's subjective understanding of the gathered knowledge. The interpretation of the findings may be biased by the researcher's own experience and background knowledge of mathematics and programming. |
| **Methodology** | Qualitative |



© University of Pretoria

| | |
|---|---|
| **Data collection techniques** | The researcher gathered most of the data by making use of interviews (semi-structured), document analysis and an extensive literature study. |
| **Research design** | Case study (phenomenological): identify mathematical thinking skills based on experiences of researcher and ICT lecturers. A case study can be used to gain contextual understanding of the researched phenomenon (Creswell, Hanson, Plano Clark & Morales, 2007). |
| **Data analysis** | Content analysis: identify themes and categories of mathematical thinking skills to derive the relevance of the skills that overlap between mathematics and programming. |

## 1.7 Concept Clarification

The following concepts were used in this study:

- *Programming* is the process of converting real-world problems into computer code by making use of problem-solving. Real-world situations (problems) are represented as programming code in program implementation to deliver solutions to the stated problem.
- A *novice programmer* is a person who has no previous experience or knowledge of programming.

## 1.8 Possible Contributions of the Study

This study hopes to contribute towards a better understanding of those mathematical thinking skills that should be developed when compiling a mathematics curriculum for first-year programming students studying towards a National Diploma in ICT. The actual development of such a curriculum is, however, outside the scope of the present study.



© University of Pretoria

## 1.9 The Structure of the Dissertation

This Chapter introduces the study and presents the rationale, the problem statement, the purpose of the study as well as the research questions. Concepts are clarified, methodological considerations are explored and possible contributions of the study are mentioned.

Chapter 2 presents an extensive literature review in which the thinking skills used in mathematics and programming are investigated. The investigation includes thinking skills in general as well as mathematical and programming skills. Overlaps and links between mathematical thinking and programming thinking are explored and are then situated in terms of the research question to indicate the links between mathematical thinking skills and programming thinking skills. The definitive work of Knuth (1985), as well as the work of Kilpatrick, Swafford and Findell (2001), is used to develop the conceptual framework towards establishing the importance of mathematical thinking skills in first-year programming subjects.

In Chapter 3, the methodological considerations are expanded into the detailed research methodology followed for this study. Paradigmatic perspectives and the characteristics of the interpretive paradigm are discussed. The research design chosen determined the sampling procedures for data collection through document and content analysis as well as for the semi-structured interviews.

The data as collected is analysed and the results discussed in Chapter 4. Chapter 5 provides the findings and conclusions with reference to the possibility of future research regarding the links between mathematical and programming thinking skills.



© University of Pretoria

# Chapter 2: Literature Review and Conceptual Framework

"The function of education is to teach one to think intensively and to think critically"
(Martin Luther King Jr.)

The aim of this study is to identify the mathematical thinking skills that first-year programming students need at a University of Technology (UoT). The literature review provides a critical synthesis of published research related to the mathematical thinking skills needed for programming.

## 2.1 Introduction

Beyer (1984) propounded that thinking includes a range of skills consisting of several cognitive steps which can also be described as a process of reasoning. As a result of the cognitive ability to think, one can therefore form opinions, judge, make decisions, contemplate about issues and solve problems (Beyer, 1984). It is important to add that a skill can be described as the ability to do something at a certain level of competency after undergoing training and gaining experience. It is also evident that different tasks require different skill sets, for example, driving a car requires a different set of skills than riding a bicycle (Smith, 2002a; Smith, 2002b).

## 2.2 Thinking skills

It is a challenge to define thinking skills since this set of skills is part and parcel of several aspects of our daily lives (Glevey, 2006, p. 292). Beyer (1984, p. 486) described thinking skills as "mental techniques or abilities" and (2002a; 2002b, p 662) suggested that "thinking skills are high-level, consciously controlled, mental activities". Tall (2013, p. 6) concurs by stating that thinking is an ongoing "reconstruction of mental connections that evolve to build increasingly sophisticated knowledge structures over time". Smith (2002a; 2002b) is of the opinion that thinking skills are applied when performing tasks that require thinking; he further suggested that people need procedural knowledge to perform specific thinking tasks. Smith (2002a; 2002b) also pointed out that thinking skills are not simply a collection of mental activities, but that the different thinking skills are organised into an intelligible whole. He argued that



© University of Pretoria

thinking skills are special abilities, extending beyond the conventional capacity to think and that they "lie between intuition and algorithms" (Smith, 2002b, p. 663). Smith (2002b, p. 663) then concluded with the following formal definition of a thinking skill: "A thinking skill is a teachable, partially proceduralised, mental activity that reaches beyond normal cognitive capacities and can be exercised as well".

Without thinking, learning is not possible. In this regard, Marzano and Hutchins (1985, p. 3) identified "three interactive elements" namely content thinking, reasoning and learning-to-learn, see Figure 2.1. These authors also acknowledged the fact that teaching students how to think should form part of every educator's objectives (Marzano & Hutchins, 1985).

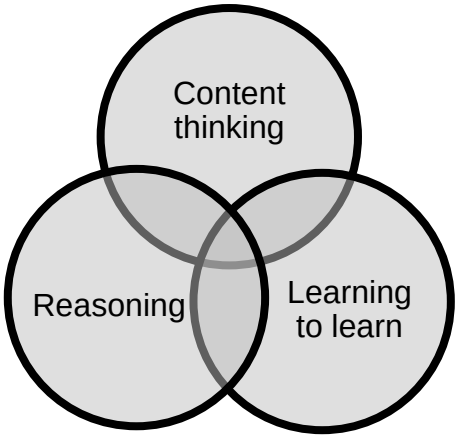



Figure 2.1: Three Interactive Elements of Thinking (Marzano & Hutchins, 1985)

Coffman (2013, p. iii) emphasises the importance of thinking skills in education and concludes that "thinking skills have long been regarded as an essential outcome of the educational process". Several other authors confirm that a range of thinking skills is necessary for effective thinking and learning (Assaf, 2009; Daly, 1995; Glevey, 2006; Mayer, 2002; Moore, 2004; Richard, 1985; Smith, 2002a; Smith, 2002b).

A considerable body of literature has been published on thinking skills and critical thinking skills. In the 1980s and 1990s, a well-documented debate between McPeck (1981, 1984, and 1990) and Ennis (1962, 1989, and 1990) ensued with opposing views regarding the generality and specificity of thinking skills (Ennis, 1989; Ennis, 1990; McPeck, 1981; McPeck 1984, McPeck, 1990). Both Ennis (1962, 1989, and 1990) and McPeck (1981, 1984, and 1990) used the terms critical thinking and thinking as



© University of Pretoria

synonyms (Ennis, 1962; Ennis, 1989; Ennis, 1990; McPeck, 1981; McPeck 1984; McPeck, 1990). Ennis (1990) reasoned that thinking skills are more general in the sense that when a thinking skill is applied in various fields, it becomes a general thinking skill. On the other hand, McPeck (1981, p.3) argued that "thinking is always thinking about something" which refers to the subject specificity of thinking skills. He concluded that critical thinking is "careful and precise thinking that is used to resolve [specific] problems" (McPeck, 1981, p. 2) and that it always "manifests itself in connection with some identifiable activity or subject area and never in isolation" (McPeck, 1981, p. 5). His view on Ennis's publications is that to teach general problem-solving (critical thinking) skills, is "like giving people a language with a syntax and no semantic", which is in effect meaningless (McPeck, 1984, p. 39).

Other authors have also joined the ongoing debate about the generality or specificity of thinking skills (Moore, 2004; Smith, 2002a; Smith, 2002b). Smith (2002) criticised McPeck's (1981, p. 665) viewpoint by distinguishing between the scope (narrow or wide) and the generality of thinking skills. He used the example of being able to count and do addition, and referred to counting and addition as "narrow scope" skills; however, used in numerous scenarios over different types of content areas, these skills can be described as general skills. This version of the matter, however, is oversimplified and Smith (2002b) stated that if a thinking skill is general it cannot necessarily be generalised, in the sense that one will not be able to apply it effectively in various fields. Moore (2004) also debated the issue by questioning if university students must be taught critical thinking as a separate subject or whether it must be taught in conjunction with subjects taken by the students. After careful analysis of both Ennis's and McPeck's work, his conclusion was that there is no definitive answer and that the debate needs to be kept alive (Moore, 2004).

Although Marzano (1998) did not specifically refer to the work of Ennis and McPeck, he and fellow researchers conducted a study at the Mid-Continent Regional Educational Lab (McREL, Aurora) in which all [American] national standards documents of all [school] subject areas were analysed in order to identify general thinking and reasoning skills across all the subject areas. Six general thinking and reasoning skill areas were identified across these subject areas. These are listed in figure 2.2.



© University of Pretoria

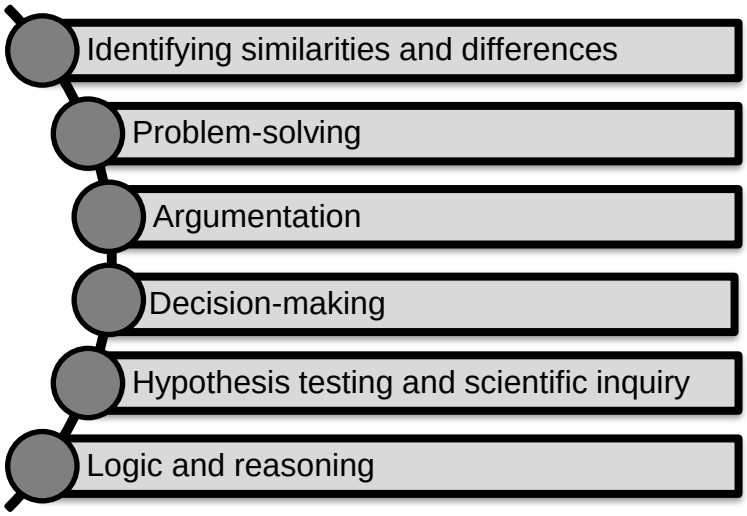



Figure 2.2: Thinking and Reasoning Skills (Marzano, 1998, p. 270).

In an unpublished work, Gubbins (1985) compiled a matrix of (general) thinking skills. The matrix was compiled by making use of the work of several well-known authors, including Bloom, Bruner, Dewey, Ennis, Feuerstein, Lipman, Sternberg, and others. Gubbins (1985) also made use of several standardised tests on critical thinking when compiling the matrix. The six main categories are listed in figure 2.3, in which the subcategories define the main categories in greater detail.

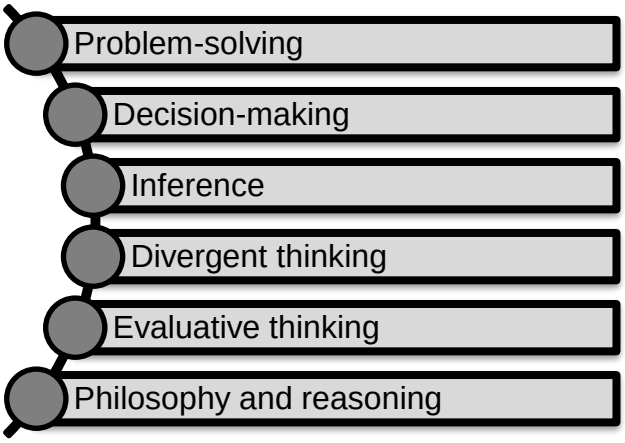



Figure 2.3: Matrix of Thinking Skills (Gubbins, 1985).

A third list of (general) thinking skills can be extracted from the Taxonomy of Educational Objectives, which was developed from 1949 until it was published in 1956. This taxonomy is referred to as Bloom's Taxonomy (Aviles, 2000; Krathwohl, 2002).



© University of Pretoria

This taxonomy was revised 45 years later and is now known as the Revised Taxonomy. For the purpose of the present study, the revised taxonomy is referred to as Bloom's Taxonomy. Although Bloom's (original and revised) Taxonomy was not developed with a view to define critical thinking, it includes six cognitive process dimensions (Krathwohl, 2002) which are used to make sense of information and knowledge (Vrchota, 2004). These cognitive processes (thinking skills) are listed in Figure 2.4.

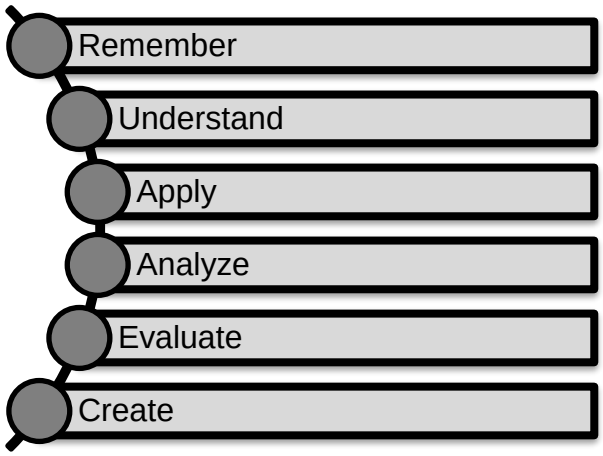



Figure 2.4: Bloom's Structure of the Cognitive Process Dimension (Krathwohl, 2002).

Aviles (2000, p. 4) argued that the above processes "constitute the construct of critical thinking". It is imperative to mention that these cognitive processes, or thinking skills, can be used and applied across different subject or knowledge areas (Krathwohl, 2002). Bloom's Taxonomy further includes four categories of knowledge in which the six thinking skills can be applied. These are listed in Figure 2.5 (Krathwohl, 2002).

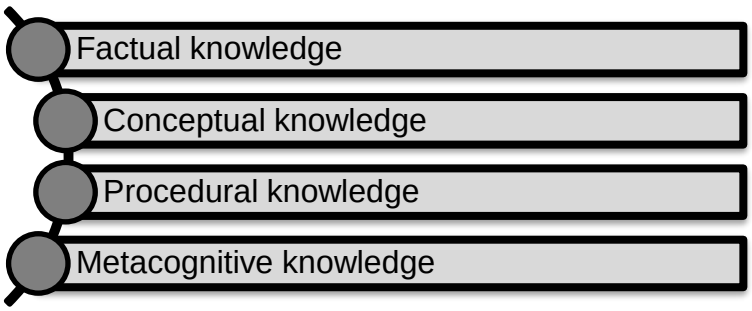



Figure 2.5: Knowledge Dimensions of Bloom's Revised Taxonomy (Krathwohl, 2002).

It is evident from the above discussion that different authors have divergent views on the specific elements and areas that form part of thinking skills. However, a common



© University of Pretoria

theme is that problem-solving and reasoning skills are crucial elements that are embedded in, and inseparable from thinking skills in general. Marzano (1988) identified six thinking and reasoning skill areas, of which problem-solving and logic and reasoning are two areas. However, in order to solve problems and reason in a logical way, the other four areas (or skills) are incorporated and applied seamlessly (see Figure 2.2 for other areas). The matrix compiled by Gubbins (1985) also included problem-solving and (philosophy and) reasoning as categories that define thinking skills, while the cognitive processes of Bloom's taxonomy emphasise the elements involved in thinking and solving problems over an array of knowledge dimensions.

Mathematical thinking is explored below as a thinking skill that includes a range of general thinking skills.

## 2.3 What are Mathematical Thinking Skills?

> "Mathematical thinking uses the same mental resources that are available for thinking in general." (Tall, 2013, p.6)

Katagiri (2004) considered the ability to think mathematically as the most important outcome of teaching, but also mentioned that in reality it does not really happen. Several attempts have been made to define the notion of mathematical thinking, and a considerable body of literature has been published on mathematical thinking skills. In his book *Textbooks, testing, training: How we discourage thinking*, Willoughby (2014, p. 1) states that "thinking is a basic skill." Willoughby then asks the question: "What good does it do us to be able to do arithmetic [mathematics] if we can't decide when arithmetic [mathematics] is desirable or decide what the results of the arithmetic [mathematics] mean?". The author argues that learning the basic skills of mathematics should not exclude thinking, but should rather encourage intelligent thinking (Willoughby, 2014).

In one of many papers by Tall (2008, p. 5) the author propounds that long-term human learning is built from "genetic structures that we all share and [to] develop more sophisticated individual knowledge based on personal experiences". Tall focuses on the notion that the mathematical thinking skills involved in university mathematics are significantly different from the thinking involved in school mathematics. A transition in



© University of Pretoria

thinking takes place here, giving rise to a "rich variety of ways of thinking mathematically" (Tall, 2008, p. 5). Mathematical thinking is therefore not a single concept, but a combination of different ways of thinking as well as thinking skills and knowledge structures (Knuth, 1985; Tall, 2008).

"Large numbers of students enter college with little intuitive feeling for the logical principles that are necessary to solve mathematical problems", according to Epp (1994, p. 263). In Gordon's view (2011), first-year students are apprehensive when it comes to mathematical thinking and participation in the mathematics classroom. This phenomenon presents itself especially when students are challenged with problems that require them to think about problems that are different from the ones that were demonstrated or explained. His observation is that mathematically successful students "have a propensity for a number of mental actions, including to look for patterns; tinker; visualize; take things apart; conjecture; simplify the problem; and change representations" (Gordon, 2011, p. 461).

For the purpose of the present study, mathematical thinking skills are a diverse combination of thinking skills that are used in the cognitive process to solve mathematical problems.

### 2.3.1 Development of Mathematical Thinking Skills

> "Mathematics is the science of patterns, and relevant mathematical activities – looking to perceive structure, seeing connections, capturing patterns symbolically, conjecturing and proving, and abstracting and generalizing – all are valued." (Schoenfeld, 1994, p. 68)

Schoenfeld (1992, p.3) stated that the tools of mathematics are "abstraction, symbolic representation, and symbolic manipulations". He reasoned that using these tools does not necessarily result in mathematical thinking, and suggested that in order to think mathematically, the student must develop:

- a mathematical viewpoint that allows them to mathematise, abstract and apply mathematical processes;
- a level of proficiency with the mathematical tools to his/her disposal in order to make mathematical sense.



© University of Pretoria

Cuoco, Goldenberg and Mark (1996) advocated changes in the mathematics curriculum. They mentioned that when asking learners [students] what mathematics is, the answer should not be limited to content (like fractions or percentages), but it must refer specifically to the problem-solving nature of mathematics. The authors warned against the tendency to teach by drilling methods like "The Five Steps for Solving a Problem". Instead, they want high school learners [students] to develop a set of "useful general habits of mind" as well as worthwhile mathematical thinking approaches (Cuoco, Goldenberg & Mark, 1996, p. 3). The learners' or students' mathematical thinking must therefore include geometric and algebraic habits of mind that should complement each other (Cuoco, Goldenberg & Mark, 1996). Once again, this view illustrates that a key aspect of mathematical thinking is that it cannot be seen in isolation.

### 2.3.2 The Relationship between Problem-Solving and Mathematical Thinking Skills

"In today's world, critical thinking and problem solving are among the primary skills that need to be held by individuals" (Özyurt, 2015, p. 353).

Halmos (1980) stated that mathematics exists in order to solve problems, and that mathematics consists of problems and solutions. Mathematical thinking includes problem-solving skills, and students solve problems while doing mathematics after learning mathematical concepts and skills, both routine and non-routine problems (Stanic & Kilpatrick, 1989). Harel and Sowder (2005) described three interrelated mathematical activities that can also be seen as indicative of the problem-solving nature of mathematics and mathematical thinking. These activities are represented in Figure 2.6. The authors concluded that the three activities can easily be combined under an umbrella activity, namely problem-solving.

The relationship between mathematical thinking and problem-solving was also confirmed by Schoenfeld (1992). He argued that when thinking mathematically, students must be able to integrate mathematical concepts and skills (insights) when confronted with new, unknown problems and scenarios; they must therefore be able to:



© University of Pretoria

- Find solutions;
- Discover patterns; and
- Formulate assumptions.

Problem-solving skills are therefore part of, and inseparable from, mathematical thinking skills.

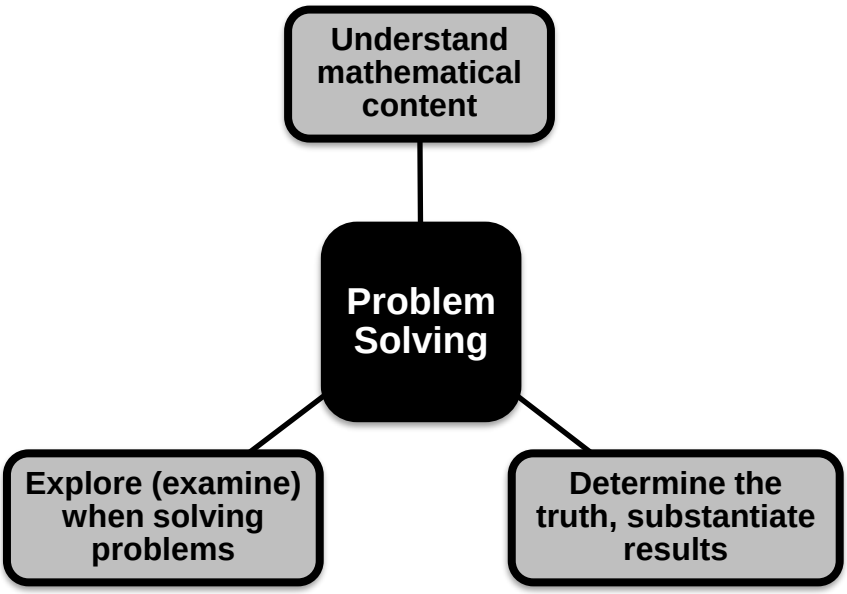



Figure 2.6: Mathematical Activities (Harel & Sowder, 2005)

### 2.3.3 Elementary vs Advanced Mathematical Thinking Skills

Several authors distinguish between elementary mathematical thinking skills (EMT) and advanced mathematical thinking skills (AMT) (Edwards, Dubinsky & McDonald, 2005; Seldan & Seldan, 2005; Tall, 1995; Zazkis & Applebaum, 2007). A broad definition of advanced mathematical thinking is that it includes mathematical thinking that is advanced for a students' age (Edwards *et al*., 2005). Harel and Sowder (2005) suggested that advanced mathematical thinking is possible even at primary school level, and concluded that the development of advanced mathematical thinking skills is a process that can evolve over time. A more comprehensive definition is that advanced mathematical thinking "requires deductive and rigorous reasoning about mathematical notions that are not entirely accessible to us through our five senses" (Edwards *et al*.,



© University of Pretoria

2005, p. 15). Tall (1995, p. 61) suggested that "elementary mathematics begins with perceptions of and actions on objects in the external world". The "change in status" of mathematical objects as well as mathematical proof evolves, and elementary mathematical thinking develops into advanced mathematical thinking (Tall, 1995, p. 13).

Learners at school level largely apply mathematical methods while university students work with formal definitions and theorems which involve deductive reasoning (Engelbrecht, 2010). At this level, their mathematical thinking must therefore evolve from describing (EMT) to defining (AMT) and from convincing (EMT) to proving (AMT) (Tall, 1995). This transition from elementary mathematical thinking to advanced mathematical thinking is usually forced when students enter tertiary mathematics and has proven to be traumatic (Engelbrecht, 2010; Tall, 2008; Watson, 2010).

For the purpose of the present study, the domain of elementary mathematical thinking is more relevant as the majority of the students studying towards a National Diploma in ICT at TUT will not be doing formal mathematics on tertiary level during their studies.

## 2.4 Thinking Skills for Programming

Programming requires thinking and learning to program is difficult
(Canast, Bajo & Gonzalvo, 1994; Schrage, 1997).

It is generally assumed that learning to program is challenging to most novice programmers (at universities) world-wide (Allan & Kolesar, 1996; Dehnadi & Bornat, 2006; Hu, 2006; Kurvinen, Hellgren, Kaila, Laakso, & Salakoski, 2016; Robins, Rountree & Rountree, 2003; Rogalski & Samurçay, 1990; Tan, Ting, & Ling, 2009). Students starting with programming in their first year usually do not struggle with the basics taught during the first few weeks of programming courses. However, due to the challenging nature and complexity of programming, not many students reach their third year of Computer Science or Information Technology courses (Bergin & Reilly, 2005).

Kurland, Pea, Clement and Mawby (1986) observed that novice programmers are inclined to focus on a specific programming language, whereas experienced programmers are able to apply and focus on general concepts and solution strategies. This observation was confirmed by Robins, Rountree and Rountree (2003, p. 151)



© University of Pretoria

when they pointed out that novice programmers “lack the knowledge and skills of experts”. It is evident that later in their programming courses, novice programmers require ongoing, guided learning (Mohamed Shuhidan, Hamilton & De Souza, 2011).

In a study exploring the development of programming ability and thinking skills among high school students, it was highlighted that teaching children to program will improve their thinking and problem-solving skills. When adult programmers were observed, they demonstrated “problem-solving strategies” such as breaking up problems in smaller parts (modules) and the use of “analogical reasoning”. Successful programmers are therefore able to plan, code and debug programs with relative ease. In this vein, Kurland, Pea, Clement and Mawbay (1986, pp. 429-430) mentioned that computer programming requires “complex cognitive skills”.

Rogalski and Samurçay (1990) developed a framework for the analysis of programming and concluded that programming involves cognitive activities of which representation and processing are the most fundamental cognitive components of programming (see Figure 2.7). Real-world problems are solved in the known context from where these are translated into a program or a programming concept that can be applied in a real-world problem-solving scenario (Rogalski & Samurçay, 1990).

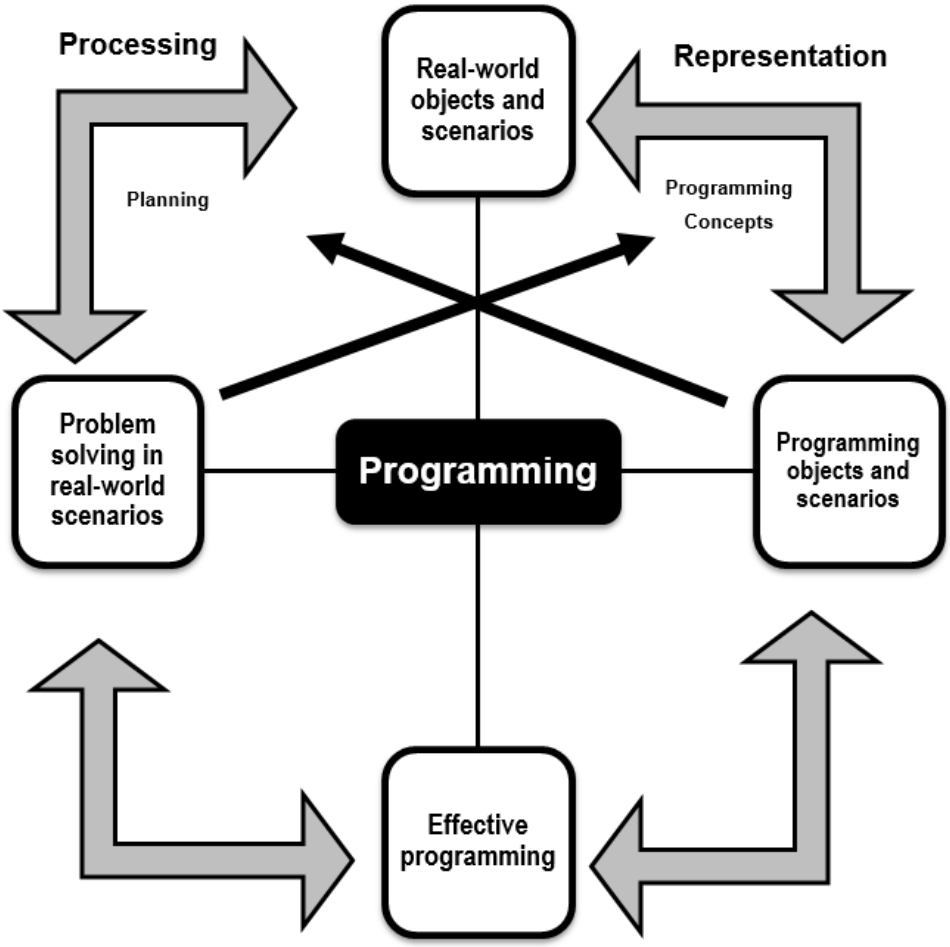
Programming thinking involves not only extensive reasoning skills, but a significant amount of mathematical thinking skills (Knuth, 1985). Knuth (1985, pp. 171-172) also reasoned that “computer science is primarily the study of algorithms”. Knuth’s (1985, p. 170) belief is that algorithms include a “whole range of concepts dealing with well-defined processes, including the structure of data that is being acted upon as well as the structure of the sequence of operations being performed”. Pretorius and Erasmus (2013) describe an algorithm as the sequential steps that need to be followed in order to solve a problem. Zsakó and Szlávi (2012) expand this definition of an algorithm and add that algorithms can be implemented (as a computer program); the authors also highlight the fact that a problem (or procedure) is divided into steps that need to be executed in a specific order. Writing algorithms is the first step in programming, and after writing an algorithm the problem can then be translated into a computer language of choice. Programming therefore teaches the concept of solving problems algorithmically (Kafai & Burke, 2015).



© University of Pretoria

Eckerdal and Berglund (2005) conducted a phenomenographical study where first-year students were interviewed and asked what it means to learn to program. Many of the students who were interviewed described programming as a "different way of thinking" (Eckerdal & Berglund, 2005, p. 135). Five different categories describing the students' understanding of what it means to learn to program were identified. The two most important categories are:

- "Learning [to program] is a way of thinking, which is experienced as difficult to capture, and which is understood to be aligned with the programming language (Eckerdal & Berglund, 2005, p. 137);
- Learning [to program] is a way of thinking, which enables problem-solving, and which is experienced as a "method" of thinking" (Eckerdal & Berglund, 2005, p. 139).






© University of Pretoria

Figure 2.7: Cognitive activities involved in programming (Adapted from Rogalski & Samurçay, 1990, p. 160)

It is important to note that the above discussion indicates that learning to program also involves problem-solving as a way of thinking. Pretorius and Erasmus (2013) focus on teaching basic programming principles and emphasise the fact that programming involves problem-solving skills.

A study based on questionnaires conducted by Wilson and Shrock (2001) set out to determine possible factors that might promote success in an introductory course in Computer Science. The outcome of the study was that the two most important factors that predict success are “comfort level” and “math background” (Wilson & Shrock, 2001, p. 187). This finding confirms Knuth’s belief (1985) that programming involves a fair amount of mathematical thinking.

According to Gomes, Santos and Mendes (2012), several factors influence students’ success or failure in programming. Two notable factors are:

- Attitudes towards programming
- Lack of problem-solving and mathematical skills.

The above-mentioned factors were also identified in the research by other authors (Goold & Rimmer, 2000; Knuth, 1985; Wilson & Shrock, 2001). The level of comfort that Wilson and Shrock (2001) referred to can be closely linked to the attitudes that students demonstrate toward programming (Gomes, Santos & Mendes, 2012) and students lacking motivation also tend to fail or drop out of programming courses (Corney, Teague, & Thomas, 2010; Kinnunen and Malmi 2006). Goold and Rimmer (2000, pp. 42-43) found that a “dislike of programming” influenced the students’ performance in programming negatively.

Programming is difficult to learn, especially for novice programmers (Grandell, Peltomäki, Back, Salakoski, 2006; Lahtinen, Ala-Mutka, & Järvinen, 2005; Milne & Rowe, 2002). Factors found to have a bearing on programming success are mainly the lack of mathematical thinking skills, lack of problem-solving skills, and the inherent difficulties of programming (Gomes, Carmo, Bigotte & Mendes, 2006).



© University of Pretoria

When thinking like a programmer, students need to be able to solve problems, reason about solutions, and then write algorithms to solve the problems. These skills are summarised and represented in Figure 2.8.

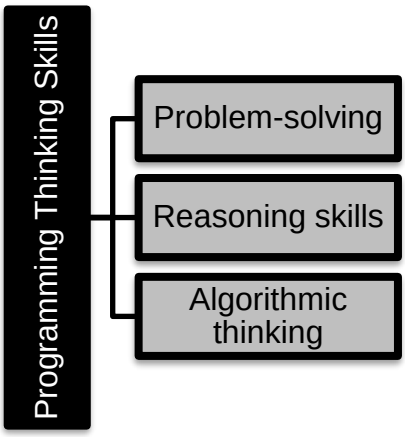



Figure 2.8: Programming Thinking Skills

Zsakó and Szlávi (2012, p. 49) argue that if a person is competent in ICT, he or she will be able to "apply basic ICT tools and methods for learning and problem solving in everyday life". Making use of Hungarian National Curriculum (ICT), Zsakó and Szlávi (2012) identify ten components that indicate whether a person is competent in ICT. These competencies are listed and briefly described in terms of their relevance to programming thinking in see Table 2.1, and an adapted version of these components are represented in Figure 2.9.

Table 2.1: ICT Competencies (adapted from Zsakó and Szlávi (2012))

| ICT Competency | Description |
|---|---|
| Algorithmic thinking | Algorithmic thinking is "problem solving, the skill of understanding, implementing, analysing and designing algorithms" (Zsakó and Szlávi, 2012, p.57). |
| Problem-solving | Problems are identified, analysed and solved and the possible use of ICT tools are evaluated throughout the process. |
| Data modelling | Real world objects are described/represented with data structures when programming. |



© University of Pretoria

| | |
|---|---|
| Modelling the real world | Models of real world phenomena are used to predict possible results/outcomes in different scenarios. |
| Orientation and information skills | Society relies on information and the ICT professional should organise information in such a way that the relevant information is accessible to the public. |
| Systemic thinking | People are part of systems in the real world, and therefore insight into computer systems forces the user to "examine, handle, develop and create" connections to utilise the system (Zsakó and Szlávi, 2012, p. 51). |
| Team work Collaboration Interoperability | Most (larger) ICT developments are done in teams and people need to develop the relevant skills to collaborate within this environment. |
| Communication skills | ICT has introduced many channels of communication and the users need to take cognisance of the benefits and dangers of these channels. |
| Application skills | Knowledge of a variety of ICT tools is essential in order to reach the most effective solution to a problem. |
| Creative skills | ICT tools are utilised in creative processes on a daily basis, for example graphic design, architecture etc. |

ICT competencies are therefore also part of the indispensable skills set of a programmer; furthermore, there are evident overlaps with basic ICT literacy as required by an information-rich society. Robins, Rountree and Rountree (2003) developed a programming framework that confirms the work of Zsakó and Szlávi (2012), as seen in Figure 2.10.

The competencies of algorithmic thinking and problem-solving are relevant to programming thinking, because writing algorithms is essentially the first step in programming and therefore teaches the concept of solving problems algorithmically (Kafai & Burke, 2015). Data modelling and modelling the real world are essential for computer programs to be effective. Data modelling refers to a method of identifying, classifying and defining data, specifically in a computer program (Klein & Hirschheim, 1987). "Programs are usually written for a purpose – with respect to some task, problem or specification", according to Robins, Rountree and Rountree (2003, p. 149). These tasks, problems and specification are therefore defined and modelled so that the programs deliver results that can be used for modelling real-world phenomena.



© University of Pretoria

Teams of programmers and other ICT professionals develop systems (involving systemic thinking) and when working together in teams, people learn different skills through collaboration and interaction with members with different skill sets (Gwo-Dong, Liang-Yi, Chin-Yea, 2012; Lippert & Granger, 1997). The digital society in which we live demands that we understand the importance of having basic knowledge of Information and Communication Technology (ICT) (Zylka, Christoph, Kroehne, Hartig, & Goldhammer, 2015). This knowledge is regarded as a fundamental educational goal for the 21st century (Wilson, Scalise, & Gochyyev, 2015). Programmers make use of computers and should therefore be ICT/computer literate to be able to utilise applications when programming and designing.

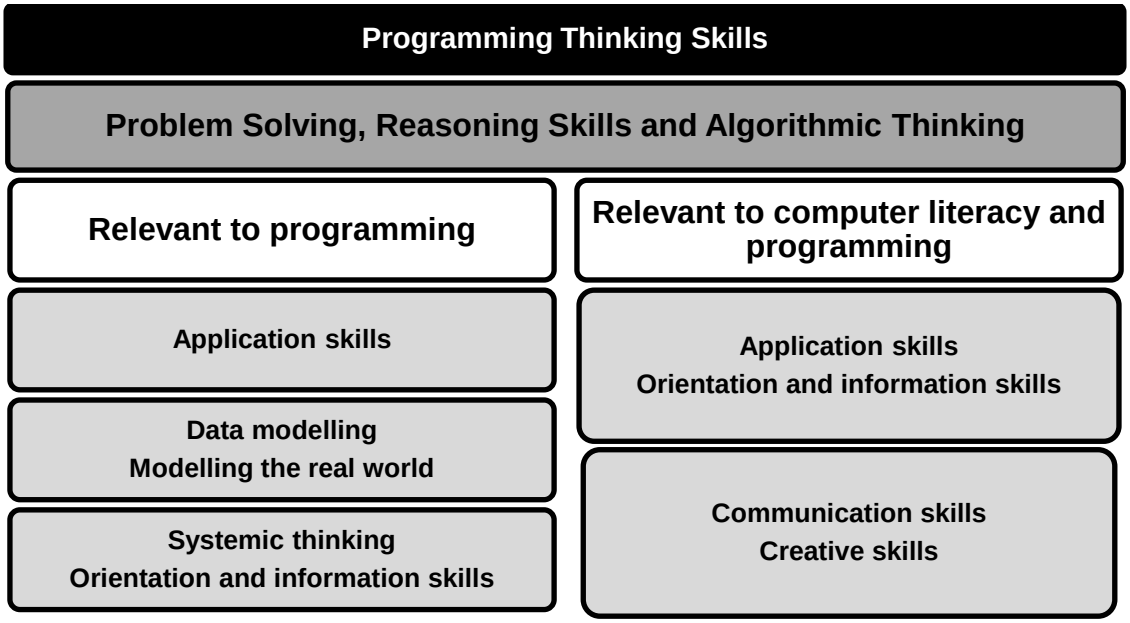



Figure 2.9: Relationships between components of ICT competencies and programming thinking skills (adapted from Zsakó & Szlávi (2012))

While it is imperative that programmers must be computer literate, it is not true that all computer literate people are able to program. This is due to the specific thinking skills required for programming. Robins, Rountree and Rountree (2003) concluded that programming (thinking) requires the abilities to obtain knowledge, develop strategies and model problems; in this process, the programmer must design, generate and evaluate the knowledge, strategies and models to reach the best possible solutions (Robins, Rountree & Rountree, 2003).



© University of Pretoria

Although these and other authors add several (and different) elements to their models, the common theme among them is that programming (thinking) requires algorithmic thinking, reasoning and problem-solving skills.

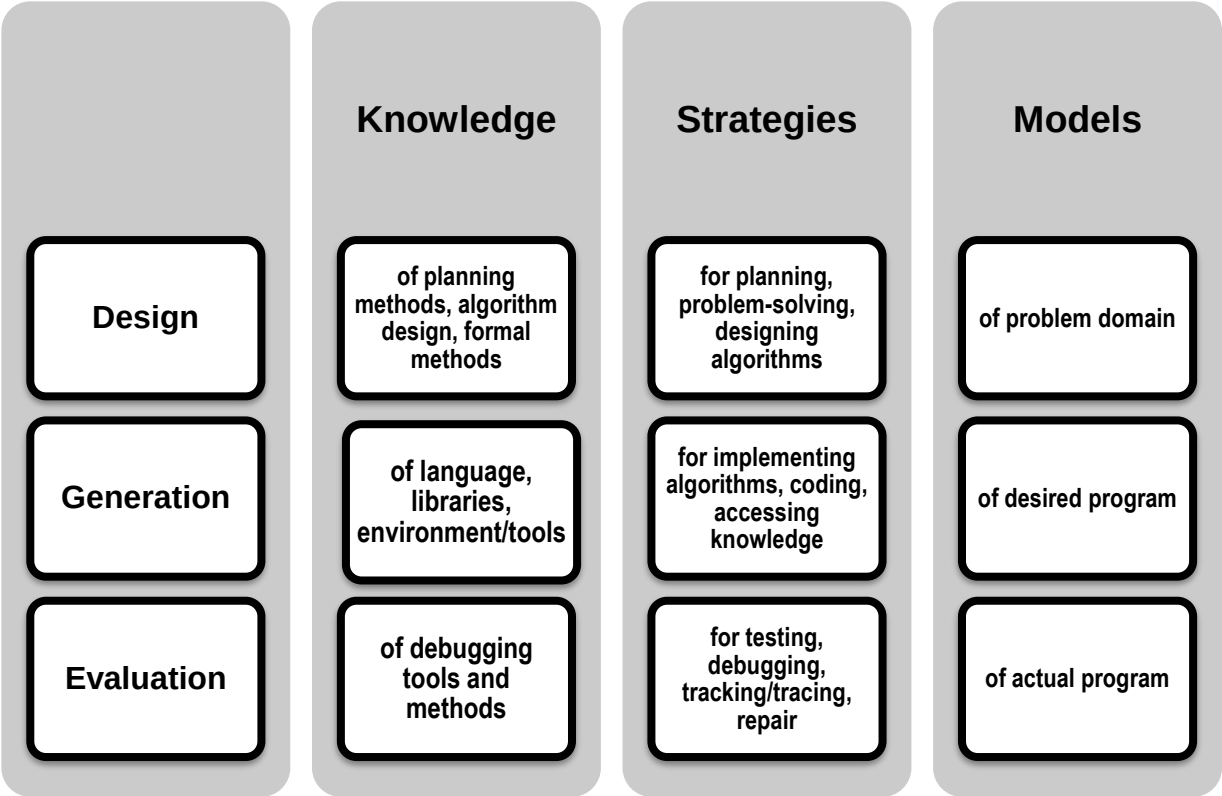



Figure 2.10: Programming Framework (Robins, Rountree & Rountree, 2003, p. 164)

## 2.5 Linking Mathematical Thinking, Mathematical Proficiency and Programming Thinking

"In the most general interpretation, every problem-solving activity is an application of mathematical reasoning." (Henderson, 2003, p. 48)

Several authors refer to either mathematical thinking skills or programming skills in isolation. Hu (2006) proposed that mathematical problem-solving abilities may play a role in the demonstration of programming abilities, and that certain mathematical skills are also linked to programming language constructs. Knuth (1985, p. 170) investigated the role of algorithms in mathematical thinking and argued that computer science



© University of Pretoria

[programming] is essentially the "study of algorithms". Hu (2006) also concluded that problem-solving, in the context of programming, is based on the use of algorithms. Knuth (1985) referred to algorithms in a broad sense and reasoned that writing algorithms include a variety of concepts that enable understanding of processes and structures used to solve problems. Algorithms can further be described as a sequential set of instructions that are written to solve problems (Pretorius & Erasmus, 2013). The main theme is therefore that algorithms are used in problem-solving when programming.

In his pursuit to determine the relationship between mathematical thinking and programming (algorithmic thinking), Knuth (1985) inquired whether mathematicians' thinking is different from programmers' thinking. In an effort to partially answer the question and to define the concept of mathematical thinking, Knuth (1985) randomly selected nine mathematics textbooks from his shelves and analysed the mathematics concepts on page 100 of each book. In his analysis of the mathematics concepts, Knuth identified nine possible categories of mathematical thinking (reasoning) skills, also referred to as "modes of thought" (Knuth, 1985, p. 172). The author (1985) highlighted the fact that combinations of these modes of thinking (reasoning) are possible, especially when referring to the thinking processes of computer scientists (algorithmic thinking). In his analysis of the mathematics in the books he selected, Knuth (1985) identified eight types of mathematical thinking and then added a ninth, namely algorithmic thinking. The skills that Knuth identified are listed in Figure 2.11.

The ninth category of "Algorithmic thinking" is an interesting addition to the above list and suggests a strong link between mathematical thinking and programming. Knuth (1985) highlighted the fact that combinations of these types of thinking are possible, especially when referring to the thinking processes of computer scientists (algorithmic thinking). It is therefore suggested that lecturers should teach students to think in an algorithmic way, and that students must take cognisance of the difficulties of this type of thinking (Gal-Ezer & Harel, 1998). In an effort to determine the possible relationship between mathematical thinking and algorithmic (programming) thinking, the concepts in Figure 2.11 are briefly discussed.



© University of Pretoria

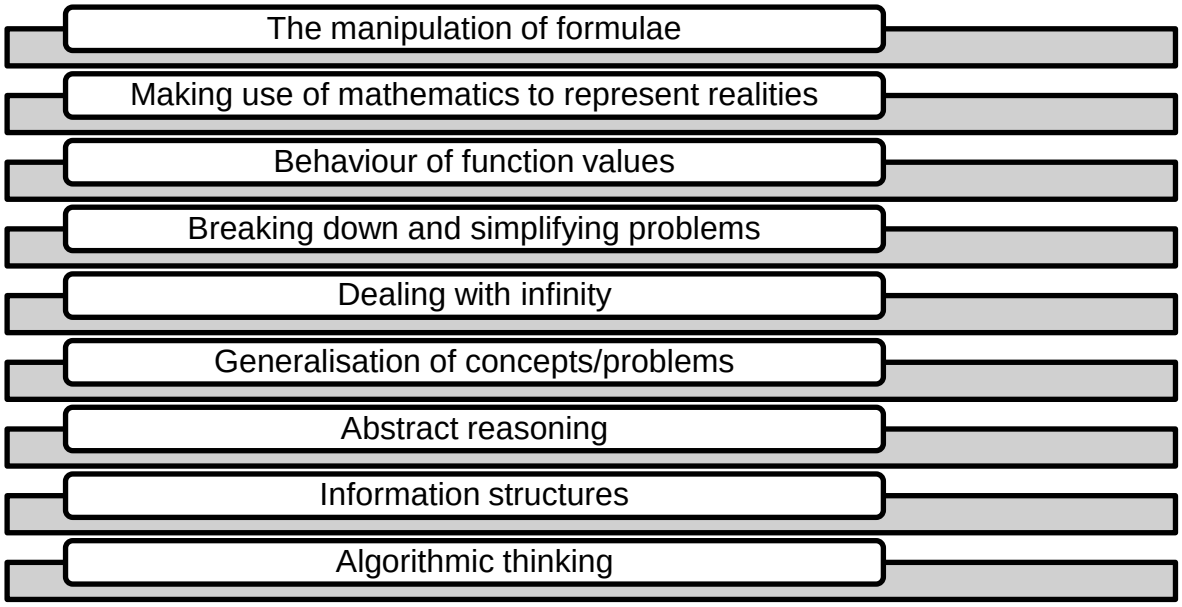



Figure 2.11: Mathematical Thinking (Knuth, 1985)

### 2.5.1 Manipulation of formulae

"Current and long-standing practice places the mastery of computational and symbolic manipulation skills in a featured role in the mathematics classroom" (Heid, 1989, p.414). In a study by Landy and Goldstone (2007) that aimed to explore the role of physical (visual) layout of algebraic equations, the authors mention that algebra is based on the manipulation of symbols used to perform numerical calculations and to interpret abstract problems. Their findings include that symbolic manipulation (reasoning) is more visual than previously thought. Similarly, Marshall (2003) argued that students must be taught mathematical understanding (mathematical thinking); this author also stressed the importance of teaching students (learners) to understand, use and apply formulae appropriately and specifically highlighted that that skills must be taught in mathematics. Knuth (1985) pointed out that formulas are manipulated (executed) in mathematics and systematic procedures are followed to determine outcomes or results. This is also the case in programming where the final execution of the program is done on a computer to deliver outcomes/results. When programming, data is processed to produce meaningful information; the processing phase makes use



© University of Pretoria

of formulae that manipulate input data to render useful information as output. This process is represented in Figure 2.12 (Pretorius & Erasmus, 2013; Farrell, 2002).

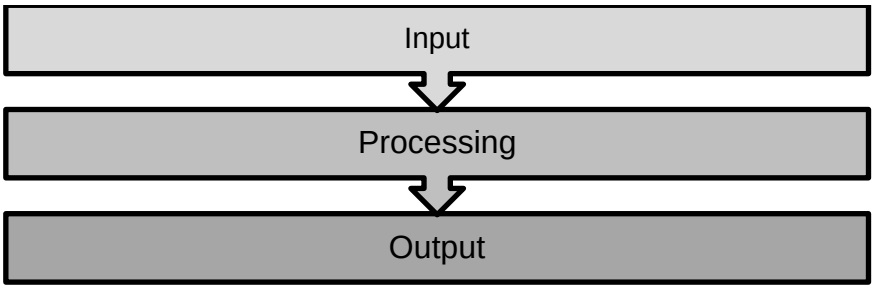



Figure 2.12: Data processing (adapted from Farrell, 2002; Pretorius & Erasmus, 2012)

### 2.5.2 Making use of mathematics to represent realities

> "Common objectives of real mathematical activity are problem solving, pattern finding, theorem production, and quantitative analysis of real-life situations" (Heid, 1989, p. 411).

Programming usually involves the construction of mathematical models of reality that can be implemented as computer programs (Bruce, Drysdale, Keleman, & Tucker, 2003). In this regard, one can consider figure 2.13 where input values are changed to determine different outcomes and therefore different views of reality. If the problem is to determine whether a student's result is pass or fail, marks are used as input, calculations are done and the answer determines the student's result of pass or fail. This concept is therefore also applied in conjunction with the manipulation of formulae.

**PROBLEM:** Enter 5 marks (percentages) and calculate the average mark (percentage).

**PROCESSING** (Making use of a formula to calculate an average percentage)

```
CalculateAverage
   Display "Enter your 5 marks as percentages"
   Enter mark1, mark2, mark3, mark4, mark5
   average = (mark1 + mark2 + mark3 + mark4 + mark5)/5
   Display "Your average is", average, "%"
   If average >= 50 then
     Display "Congratulations, you have passed the module!"
   else
     Display "You have failed the module."
   endif
end
```



© University of Pretoria

**INPUT** (Unprocessed data, as seen on computer screen)

| Enter your marks as percentages | Mark 1 | Mark 2 | Mark 3 | Mark 4 | Mark 5 |
|---|---|---|---|---|---|
| | 45 | 67 | 84 | 27 | 67 |

**OUTPUT** (Result, meaningful information, as seen on computer screen)

```
Your average is 58%
Congratulations, you have passed the module!
```

Figure 2.13: Pseudo-code example: Making use of mathematics to represent realities

Marr (2001) observed that when students are challenged with real-life scenarios and objects in mathematical activities, this enhances their understanding of mathematical formulae and symbols. In a similar vein, Schoenfeld (1992, p. 4) argued that learning mathematics is “empowering” and that students who are mathematically proficient are also “quantitatively literate”. Worldwide, studies have shown that most students leave high school with inadequate mathematical and quantitative literacy skills and are therefore ill-prepared for the real world (Ball, Lubienski & Mewborn, 2001; Hourigan & O'Donoghue, 2007; Kantner, 2008; Parnell & Statham, 2007). “Unfortunately, little numeracy is taught anywhere except in mathematics classes, and not as much as one might expect, is taught (much less learned) even there.” (Steen, 1999, p. 8).

The term ‘quantitative literacy’ is typically used in the USA while the British and Australians prefer the term ‘adult numeracy’. In the South African context ‘mathematical literacy’ is used, and this term also refers to a subject offered from grade 10 to 12 as an alternative to Mathematics (Frith & Prince, 2009). Quantitative literacy has become a major role-player in a society filled with numerical data, graphs and numerous other quantitative pieces of information that need to be interpreted. These skills form part of adults’ (and students’) daily lives when they perform essential tasks such as shopping for groceries, using recipes, working out budgets, or doing home improvements. In other words, adults (and students) need basic quantitative literacy.

In an attempt to achieve a better understanding of the concepts of quantitative literacy and adult numeracy, a number of authors’ perspectives are examined in the discussion that follows.



© University of Pretoria

Frith and Prince (2009, p. 85) argue that quantitative literacy is a concept that manifests as "a social practice in which people manage situations or solve problems involving quantitative information". Lloyd and Frith (2013, p. 1) propound that "Quantitative literacy is the ability to manage situations or solve problems in practice, and involves responding to quantitative (mathematical and statistical) information that may be presented verbally, graphically, in tabular or symbolic form; it requires the activation of a range of enabling knowledge, behaviours and processes and it can be observed when it is expressed in the form of a communication, in written, oral or visual mode".

The common theme seen above is that quantitative literacy is demonstrated by the capability of an adult (i.e. a student) to manage real-life situations and solve problems when confronted with quantitative scenarios. It is also important to mention that "the relationship between numeracy and mathematics is problematic"; in other words, being quantitative literate is not an automatic outcome of years of formal mathematics as taught in schools (O'Donoghue, 2002, p. 37). O'Donoghue (2002) also argued that adult mathematics education should involve not only pure, formal mathematics but also quantitative literacy, because knowledge of the one does not necessarily imply knowledge of the other.

### 2.5.3 Behaviour of function values

Transformation of functions is, and has always been an indispensable element of algebra (Ninness *et al*., 2006), Croft, Croft and Davison (2008) developed that a mathematical function uses inputs on which a rule is applied to produce output. Waisman, Leikin, Shaul and Leikin (2014) point out that through the study of functions, links are created between symbolic, graphical, verbal and numerical representations. Functions therefore allow multiple representations of real-life problems (Waisman, Leikin, Shaul & Leikin, 2014). The manipulation of formulas lead to the possible understanding of the behaviour of functions, and formulas applied in functions are used to do calculations. When the values of variables (input data) are changed, the output (results) must also change (see Figure 2.14). Analysing these results assists towards decision-making and making possible adjustments to the algorithmic process. Writing algorithms and then translating them into programs require all the above skills to solve real-life problems.



© University of Pretoria

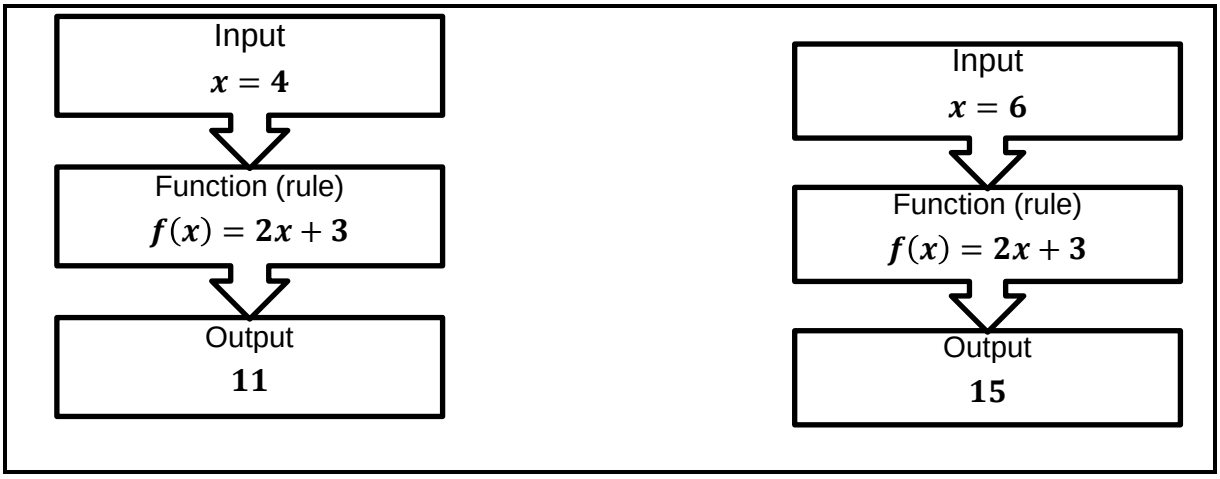



Figure 2.14: Mathematical function $f(x) = 2x + 3$

### 2.5.4 Breaking down and simplifying problems

Breaking problems into smaller parts as well as generalisation as described by Knuth (1985) requires students to be procedurally fluent (Kilpatrick, Swafford & Findell, 2001). Most programming languages allow programmers to split complex algorithms into smaller modules (functions or procedures) which are easier to understand, and these modules can then be re-used multiple times in a computer program (Farrell, 2002). The concept of breaking up problems is an essential skill for programmers (Barnes, Fincher & Thompson, 1997; Shrage, 1997). Lecturers should teach these concepts in such a way that it advances the students' "planning skills through the use of algorithms and modular programming" (Overbaugh, 1993, p. 4). Modular programming refers to the process when programs and/or algorithms are written in modular form and broken down into smaller procedures or functions that each solves a specific portion of the specified problem (Barnes, Fincher & Thompson, 1997; Shrage, 1997). When utilising functions in programs, parameters (values) are passed between functions, but novice students find the concept of passing parameters challenging (Madison & Gifford, 2002). However, this knowledge and skills are imperative for programming students when learning to program.

Several steps need to be followed when breaking up bigger programs in to modular units. The steps as suggested by McCracken *et al.* (2001) can be seen in Figure 2.15. Mathematical problems are also solved by simplifying a problem and identifying



© University of Pretoria

possible patterns, and these patterns can in turn be generalised to define and clarify formal concepts (Knuth, 1985; Polya, 1957; Putnam, 1987). In this process, the problem is broken down into several smaller, easier to solve problems.

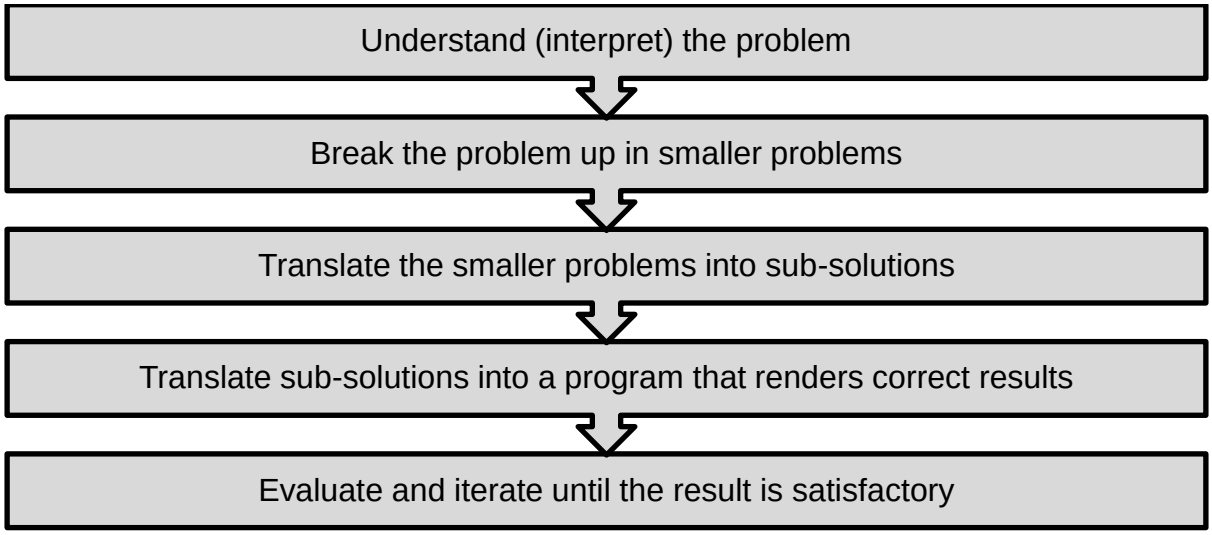



Figure 2.15: Programming steps (adapted from McCracken *et al*., 2001).

### 2.5.5 Dealing with infinity

It is often necessary to make use of repetition in programming. The structures used for repetition are called loops (Pretorius & Erasmus, 2013). If loops are not used correctly, it can lead to programs running indefinitely without end (see Figure 2.16). The programmer must therefore be aware of the correct number of executions of the sequence of steps. The concept of infinity is also demonstrated and applied as regression, which is too advanced for first-year programming students and was therefore excluded for the purposes of the present study.

**PROBLEM:** Enter 5 marks (percentages) for any number of students each and calculate the average mark of each student (percentage).

**PROCESSING** (Making use of a formula to calculate an average percentage)

```
CalculateAverage
   Display "How many students are in the class?
   Enter noStudents
   repeats = 0
   do while repeats <= noStudents
      Display "Enter your 5 marks as percentages"
      Enter mark1, mark2, mark3, mark4, mark5
      average = (mark1 + mark2 + mark3 + mark4 + mark5)/5
      Display "Your average is", average, "%"
```



© University of Pretoria

```
        If average >= 50 then
            Display "Congratulations, you have passed the module!"
        else
            Display "You have failed the module."
        endif
    loop
end
```

**INPUT** (Unprocessed data, as seen on computer screen)

| | | | | | |
|---|---|---|---|---|---|
| Enter your marks as percentages | Mark 1 | Mark 2 | Mark 3 | Mark 4 | Mark 5 |
| | 45 | 67 | 84 | 27 | 67 |
| Enter your marks as percentages | Mark 1 | Mark 2 | Mark 3 | Mark 4 | Mark 5 |
| | 26 | 56 | 46 | 78 | 47 |
| This will be repeated **infinite** times, this results in a run-time error | | | | | |

**OUTPUT** (Result, meaningful information, as seen on computer screen)

```
Your average is 58%
Congratulations, you have passed the module!

Your average is 50.6%
Congratulations, you have passed the module!
```

[This will be repeated infinite times because the line that changes the value of **`repeats`** is omitted]

Figure 2.16 Infinite loop, repetition does not stop (error)

### 2.5.6 Generalisation of concepts/problems

"When we cannot use the compass of mathematics or the torch of experience…it is certain we cannot take a single step forward." (Voltaire, http://www.azquotes.com/quote/1313601)

Once students develop the ability to transfer their knowledge and skills from quantitative scenarios to general rules that can model situations of different types, they are able to generalise the scenarios and create links between the mathematical content (Lannin, 2003). "The objects needed for mathematical [thinking] are mental models, and the [problems] are conceptual resources that are used in constructing models" (Greeno, 1994, p. 272). Mathematics is therefore used to model, generalise and solve real-world problems. In this regard, it has been observed that students who



© University of Pretoria

are successful in mathematical modelling are able to grasp the connections between the real world and the mathematical world with greater skill (Crouch & Haines, 2004).

Programming is also used to model, generalise and solve real-world problems and, as is the case with in mathematics, programs make use of variables to represent (changing) values. Variables can also be referred to as a position in memory where a value is stored and the value of variables are changed when processing takes place (Pretorius & Erasmus, 2013). Becker and Riviera (2005) proposed that students regard variables as more than placeholders, and they also interpret and use them in the context of the problem statement. Variables are used in order to generalise the problem, while the input values determine the output values, thus making use of a generalised formula or function. In a study by Becker and Riviera (2005, pp. 4-128), the authors found that "students who fail to generalize tend to start out with numerical strategies" rather than utilising varying generalisation approaches to solve a problem.

### 2.5.7 Abstract reasoning

"Where the senses fail us, reason must step in."

(Galileo Galilei, http://www.brainyquote.com/quotes/quotes/g/galileogal377935.html)

"Epistemologically, reasoning is the foundation of mathematics" (Steen, 1999, p. 271). Cerbin (1988, p. 1) defined reasoning as an intellectual ability that is demonstrated when "solving problems, making judgements and decisions". The author emphasised the fact that students need reasoning skills to ensure success in higher education and also in life in general (Cerbin, 1988). Vos (1978) conducted a study on learning styles and plausible reasoning and found that students who were successful problem-solvers could reason well without following a set of rules; the author found that such students therefore had a natural ability to reason and solve problems.

Problem-solving requires the ability to reason about appropriate solutions and finding alternative solutions to problems (Kilpatrick, Swafford & Findell, 2001). Knuth (1985) referred to abstract reasoning which is similar to adaptive reasoning; both mathematical thinking and programming thinking are dependent on the ability to reason in an abstract way (Kilpatrick, Swafford & Findell, 2001; Overbaugh, 1993; Robins, Rountree & Rountree, 2003; Schoenfeld, 1992; Tall, 2013). Programs are



© University of Pretoria

written in order to solve problems, and therefore programmers must interpret the problem and then design a solution. It follows that programmers put abstract reasoning into practice. It is imperative to note, in this regard, that knowledge about the structure and syntax of a programming language does not guarantee a working solution (Madison & Gifford, 2002; Robins, Rountree & Rountree, 2003).

### 2.5.8 Information structures

When programming and writing algorithms, data structures are used to represent and organise data. Mathematical representations of complex objects used in proofs are similar to data structures used when programming. One such example is arrays. Arrays are structures that are often used in programming, and are similar to matrices as used in mathematics (See Figure 2.17). An array can be defined as a list of variables (similar to a table) which has a single (variable) name. Arrays can have one or more dimensions (multiple rows and columns) and elements are referred to by making use of indices that indicate their position in the array (Farrel, 2002; Pretorius & Erasmus, 2013), see figure 2.18.

$$\begin{bmatrix} 21348713 & 23 & 55 & 56 & 38 & 66 \\ 21348714 & 45 & 83 & 76 & 89 & 78 \\ 21348715 & 66 & 47 & 56 & 73 & 65 \\ 21348716 & 73 & 83 & 87 & 91 & 77 \\ 21348717 & 76 & 77 & 65 & 59 & 66 \\ 21348718 & 88 & 52 & 66 & 48 & 58 \\ 21348719 & 28 & 78 & 65 & 76 & 47 \end{bmatrix}$$

Figure 2.17: Example of a matrix, representing data of Figure 2.18

| | | | | | |
|---|---|---|---|---|---|
| 21348713 | 23 | 55 | 56 | 38 | 66 |
| 21348714 | 45 | 83 | 76 | 89 | 78 |
| 21348715 | 66 | 47 | 56 | 73 | 65 |
| 21348716 | 73 | 83 | 87 | 91 | 77 |
| 21348717 | 76 | 77 | 65 | 59 | 66 |
| 21348718 | 88 | 52 | 66 | 48 | 58 |
| 21348719 | 28 | 78 | 65 | 76 | 47 |

Figure 2.18: Example of an array `studentArray` with student numbers and marks



© University of Pretoria

When referring to the fourth mark of the second student in Figure 2.18, it will be referred to as `studentArray(2,5)`. Both the above examples could also be represented as multiple one-dimensional arrays or matrices, or as multi-dimensional arrays or matrices as demonstrated above.

### 2.5.9 Algorithmic thinking

"Mathematics is central to designing and analysing algorithms" (Bruce *et al*., 2003, p. 41).

As previously mentioned, an algorithm is used to solve problems; it is a set of sequential instructions written in pseudo-code that can be translated into a programming language of choice. Zsakó and Szlávi (2012) state that the implementation of an algorithm requires a specific way of thinking, i.e. algorithmic thinking. Algorithmic thinking requires the thinker (programmer/problem solver) to implement algorithms that solve problems accurately (Zsakó & Szlávi, 2012). Algorithmic thinking is fundamental to being able to program (Snyder *et al*., 1999). Zsakó and Szlávi (2012) argue that a person (programmer) is competent in terms of algorithmic thinking, and therefore programming, if he/she demonstrates certain abilities. The abilities are represented in Figure 2.19.

Algorithmic thinking includes several concepts used in programming such as breaking problems into smaller parts, repetition, basic data organisations, generalisation and the top-down design and refinement of programs (Snyder *et al*., 1999). Snyder *et al*. (1999, p. 31) proposed that programming is the medium used to apply algorithmic thinking and that "algorithmic thinking is key to applying information technology" to real-life scenarios.

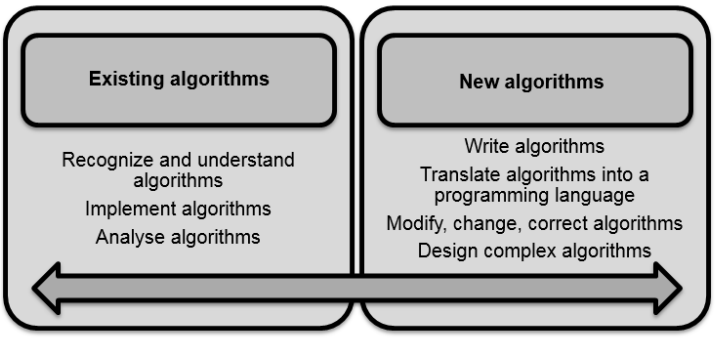



Figure 2.19: Algorithmic thinking competence (adapted from Zsakó and Szlávi (2012)).



© University of Pretoria

## 2.6 Conceptual Framework

### 2.6.1 Introduction

For the present study, a conceptual framework is needed to "situate the research" (Maree & Van der Westhuizen, 2013, p. 30). The conceptual framework of this study can provide an "overview of perspectives" (Ferreira, 2012, p. 34). The purpose and focus of the present study is to identify the necessary mathematical thinking skills that are required by novice programmers. This study sets out, in short, to determine the mathematical thinking skills required when programming. The mathematical thinking skills highlighted by Knuth (1985) and the components of mathematical proficiency (Kilpatrick, Swafford & Findell, 2001) demonstrate similar sets of thinking skills that can be used to determine the relationship between mathematical thinking and programming thinking. This relationship is summarised in Figure 2.20.

It is imperative to recognise that problem-solving and reasoning skills have also been identified as thinking skills by several authors as discussed above (Gubbins, 1985; Kratwohl, 2002; Marzano, 1998; Vrchota, 2004). The above summary and discussions clearly indicate the application potential and practicality of the overlaps between mathematical thinking skills and programming thinking skills (Kilpatrick, Swafford & Findell, 2001; Knuth, 1985).

### 2.6.2 Problem-solving, reasoning skills and algorithmic thinking in programming

In a publication by the National Research Council (USA), "five strands of mathematical proficiency" were identified by the Mathematics Learning Study Committee with a view to determine what is involved when mathematics is learnt successfully (Kilpatrick, Swafford & Findell, 2001, p. 116). These five interdependent components/strands of mathematical proficiency provide a structure for the identification of knowledge, skills and abilities required for a person to be mathematical proficient. These strands are positioned in conjunction with the work of Knuth (1985) in the next section. Kilpatrick, Swafford & Findell, (2001) referred to conceptual understanding as being central to mathematical proficiency. In order to reach this point of (conceptual and strategic) understanding in mathematics, students need to be guided (taught) through the use of models, emphasis on relevant concepts and extensive exercises (Gordon, 2011).



© University of Pretoria

When programming, students must be able to apply programming concepts to different scenarios, and in doing so, they are required to demonstrate conceptual and strategic understanding. Without conceptual and strategic understanding, problem-solving is impossible. Indeed, Gordon (2011) emphasises the fact that students need problem-solving skills to solve new, unfamiliar problems.

Programming thinking requires problem-solving skills, and therefore the same principles apply to mathematics and programming. A study conducted at a Malaysian university, to explore possible teaching strategies for programming subjects, investigated students' perceptions why programming is difficult (Tan, Ting & Ling, 2009). The results of the study confirmed the fact that students tend to lack the necessary understanding and reasoning of how programming works, and they fail to understand the abstract nature of how programs are executed by the computer (Tan, Ting & Ling, 2009). Lacking these skills prevents students from becoming efficient programmers (Woodley & Kamin, 2007).

Kilpatrick, Swafford & Findell (2001) argue that adaptive reasoning entails the ability to think in a logical way and to understand the relationships between concepts and situations. Problem-solving requires the ability to reason about the appropriate solutions and finding alternative solutions to problems (Kilpatrick, Swafford & Findell, 2001). In this regard, Knuth (1985) refers to abstract reasoning which is related to adaptive reasoning.

As indicated above, programming is challenging, especially to novice programmers - and this is evidenced by high drop-out rates in programming courses (Grandell, Peltomäki, Back, Salakoski, 2006; Lahtinen, Ala-Mutka, & Järvinen, 2005; Milne & Rowe, 2002). Lahtinen, Ala-Mutka, and Järvinen (2005) conducted a study using a web-based questionnaire at six universities in Germany, Finland, Iceland, Romania and Latvia, in which they investigated the difficulties that novice that programmers experience. The study had three focus areas:

- General information of respondents;
- Investigating difficulties in learning course content, i.e. program construction and programming concepts;



© University of Pretoria

- What type of resources students perceive to be effective when learning to program (Lahtinen, Ala-Mutka, & Järvinen, 2005).

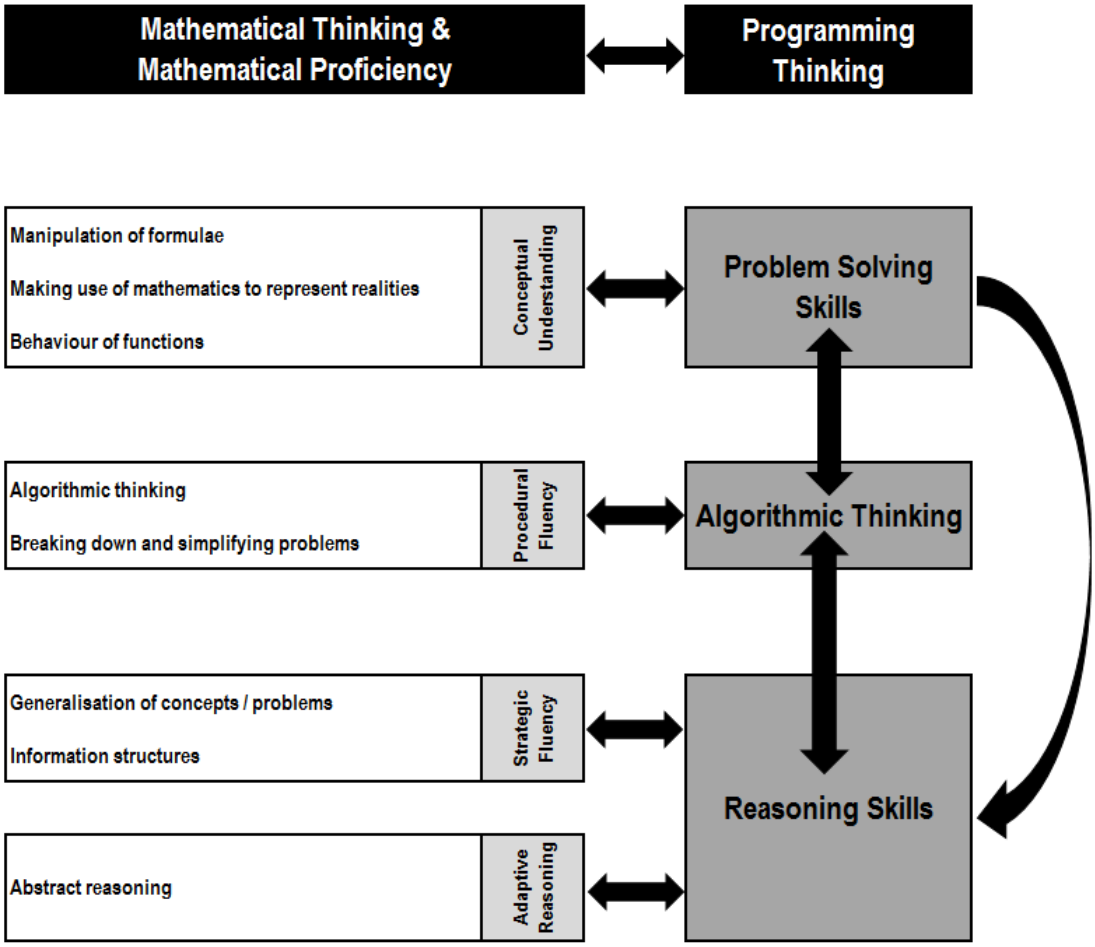



Figure 2.20: Conceptual Framework: Relationship between mathematical and programming thinking skills

For the purpose of the present study, I want to emphasise the results of the second focus area (investigating difficulties in learning course content, i.e. program construction and programming concepts). The results clearly indicated that students struggle the most with the following:

- How to solve a specific problem and to program it (problem-solving and reasoning);
- Breaking up problems in smaller parts (algorithmic thinking);
- De-bugging their programs (Lahtinen, Ala-Mutka, & Järvinen, 2005).



© University of Pretoria

Gal-Ezer and Zeldes (2000) suggested that programming students should focus on more than the syntax of a programming language and rather develop the skills to apply concepts for reuse in multiple scenarios, different data structures and programming languages. The authors further argued that when students (programmers) are able to organise their knowledge, this leads to improved retention and the ability to reuse (generalise) the knowledge and programming concepts efficiently (Gal-Ezer & Zeldes, 2000).

Algorithmic thinking requires procedural competence (fluency) when solving non-routine mathematical problems and also when writing algorithms when programming (Knuth, 1985; Kilpatrick, Swafford & Findell, 2001). A range of different information structures (data structures) is used to determine the appropriate outcome or result when solving problems and writing algorithms (programs). When students are procedurally fluent in mathematics, they are also able to re-use procedures in familiar and unfamiliar scenarios in appropriate, flexible, accurate and efficient ways (Kilpatrick, Swafford & Findell, 2001). The same applies for programming where breaking problems into smaller parts as well as generalisation as described by Knuth (1985) requires students to be procedurally fluent. Students who are able to use different strategies to solve the same problem (in mathematics), are strategically and procedurally competent (fluent) and can formulate, represent and solve non-routine problems (Kilpatrick, Swafford & Findell, 2001). The same competencies are relevant in the context of programming.

## 2.7 Summary

In summary, when learning to think like programmers, first-year students need to be able to solve problems, reason about solutions and then write algorithms to solve the problems, while making use of mathematical thinking skills. This summary is represented in Figure 2.21.



© University of Pretoria

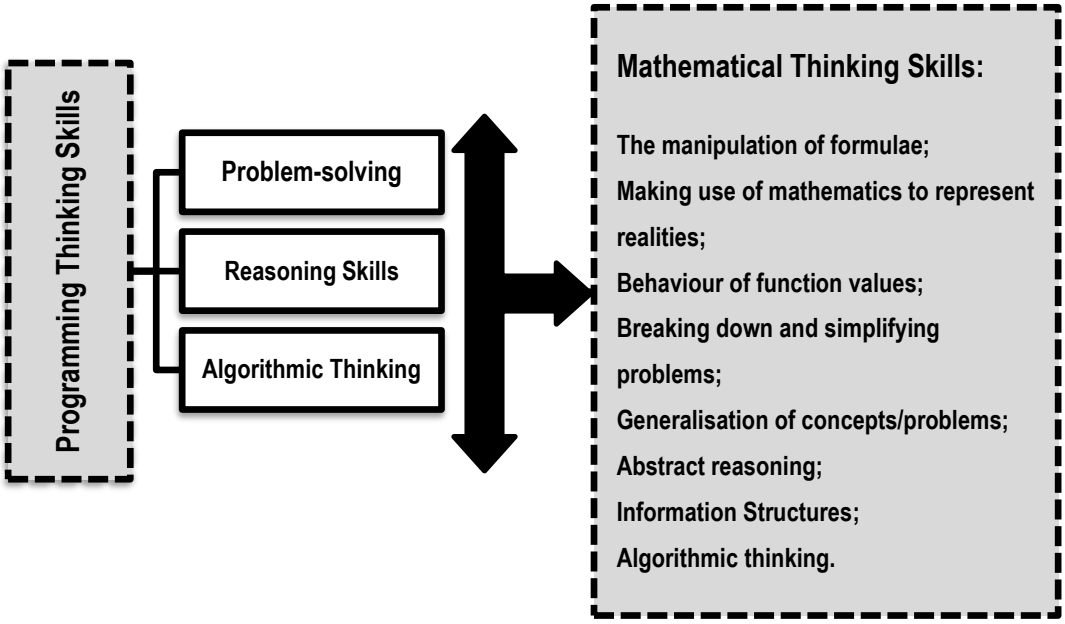



Figure 2.21 Programming and Mathematical Thinking Skills



© University of Pretoria

# Chapter 3: Research Methodology

"Qualitative researchers are interested in accessing experiences, interactions and documents in their natural context and in a way that gives room to the particularities of them and the materials in which they are studied." (Kvale, 2011. P. xi)

## 3.1 Introduction

The present study was conducted following a qualitative methodology. When conducting qualitative research the researcher is emerged in, and part of the research process, and his/her presence and experience in the field of the study therefore empowers him/her to participate and reflect with substantial understanding of the research topic (Cohen, Manion & Morrison, 2011; Kvale, 2011).

## 3.2 Paradigmatic Perspectives

A paradigm can be described as a person's view of the world and is represented in his/her assumptions of reality and the knowledge thereof. Therefore, all research must be conducted within the researcher's paradigm or understanding of the world. It could be said that this view determines a person's principles, which in turn guide his/her actions and ideas (Maree & Van der Westhuizen, 2013; Nieuwenhuis, 2013). Due to the nature of the present study, critical realism was chosen as the ontological paradigm, and anti-positivism as the epistemological paradigm. Both of these are part of the broader spectrum of interpretivism.

When identifying mathematical thinking skills, these were classified and contextualised and then linked to the relevant mathematical thinking skills that are needed for programming (Barron, 2013; Denzin & Lincoln, 2005). This process did not only rely on the researcher's own view and experiences of mathematics and programming thinking skills; literature and other experts were also consulted in the process.



© University of Pretoria

## 3.3 Interpretive Paradigm

An interpretive paradigm (see Table 3.1) refers to one's subjective view and understanding of the world we live in (Cohen, Manion & Morrison, 2011). From a particular interpretive view, a person attempts to understand the world (phenomena) by examining the different meanings that different people assign to the same phenomena (Nieuwenhuis, 2013). Interpretivists deduce their findings and theories from individuals' interpretations of their environment and situations (Bales, 2002; Cohen, Manion & Morrison, 2011). Bales (2002) mentioned that interpretive researchers believe that humans make certain choices based on their interpretations of the social environment they are in, and Wagner, Kawulich and Garner (2012) point out that reality is constructed socially and cannot be generalised.

In order to understand human experiences in a subjective way, the researcher is immersed or involved in the research (Cohen, Manion & Morrison, 2011; Nieuwenhuis, 2013). In this regard, critical realism is one way of describing and explaining why the world and its phenomena are prone to errors and different categories; relationships between phenomena are therefore not absolute and are thus open for variable interpretations (Scott, 2005). It is relevant to point out that anti-positivism gives rise to subjective interpretation of research as demonstrated by the involvement of the researcher in the environment in which a study is conducted.

Table 3.1: Selected paradigm (adapted from Wagner, Kawulich & Garner, 2012)

| **Interpretivist Paradigm** | |
|---|---|
| **Why is the research done?** | To understand the nature and set of mathematical thinking skills relevant to programming. |
| **Philosophical paradigm** | Phenomenological: identify mathematical thinking skills based on analysis of data obtained. |
| **Ontological Assumptions** | Many sets of mathematical thinking skills are described in the literature. Therefore, there is more than one reality (or truth). |



© University of Pretoria

| | |
|---|---|
| **Nature of knowledge (epistemology)** | The study relies on the researcher's subjective understanding of the gathered knowledge. The researcher interprets the findings, and this may be biased by the researcher's own experience and background knowledge of mathematics and programming. |
| **Methodology** | Qualitative |
| **Data Collection Techniques** | The researcher gathers most of the data by making use of interviews (semi-structured), document analysis and an extensive literature study. |
| **Research Design** | Case study (phenomenological): identify mathematical thinking skills based on experiences of researcher and ICT programming lecturers. A case study can be used to gain contextual understanding of the researched phenomenon (Creswell, Hanson, Plano Clark & Morales, 2007). |
| **Data Analysis** | Content analysis: Identify themes and categories of mathematical and programming thinking skills to derive the relevance of thinking skills that are shared by mathematics and programming. |

## 3.4 Research Design

Nieuwenhuis (2013, p. 70) refers to research design as "a plan or strategy". In an attempt to answer the primary research question, the researcher collected, analysed and reported on text-based data obtained from an extensive literature review, documents (assessments) and semi-structured interview with first-year programming lecturers, examiners and moderators.

As regards case studies, these "can include people, events institutions or even programmes" (Seabi, 2012, p. 83). In the present case study, the researcher selected



© University of Pretoria

an area of focus with the hope that this would improve her insight and understanding of the programme (phenomenon) that is being researched (Nieuwenhuis, 2013). A case study must lead towards an understanding of the relationships and interactions between participants when studying a phenomenon. This type of understanding positions a case study within the interpretivist paradigm. Nieuwenhuis (2013) also adds that the term case study is used in many different ways; it was found apposite to use a case study research design for the present study.

When using a case study design, it is imperative to make a decision regarding the "unit of analysis" (Nieuwenhuis, 2013, p. 75). Cohen, Manion and Morrison (2011, p. 296) concur that case study research is the observation of the "characteristics of an individual unit". Generalisations regarding the unit as part of a system can therefore be made, and these units can also be referred to as phenomena. For the purpose of the present case study design, the focus is on determining those mathematical thinking skills that are necessary for programming, and the unit of analysis (phenomenon) is therefore a set of mathematical thinking skills (Nieuwenhuis, 2013).

### 3.4.1 Selection of Participants and Sampling Procedures

Sampling is defined as the "process used to select a portion of the population" that needs to be studied (Nieuwenhuis, 2013, p. 79). The main purpose of sampling is to collect the best (richest) data that will answer the research question(s) (Nieuwenhuis, 2013). Smaller samples are adequate when conducting qualitative research and are easier to adjust, adapt or change until all the relevant data has been obtained; in other words, until data saturation has been reached (Nieuwenhuis, 2013). In order to identify the mathematical thinking skills needed for programming, convenience sampling is used in the present study. Entry level (first-year) programming assessments (examinations papers) from Tshwane University of Technology (TUT) over two years were selected and analysed. The examiners and moderators of these question papers, as well as lecturers who lecture or who have lectured, this subject for two years or more were interviewed. The sample was selected in such a way that a better understanding of the necessary skills could be attained.

The outcome of this process entailed a selection of the most critical mathematical thinking and problem-solving skills that need to be developed when compiling a



© University of Pretoria

mathematics curriculum for first-year programming students studying towards a National Diploma in ICT. However, the development of such a curriculum itself is outside the scope of the present study. The research process followed in this study is represented in Figure 3.1.

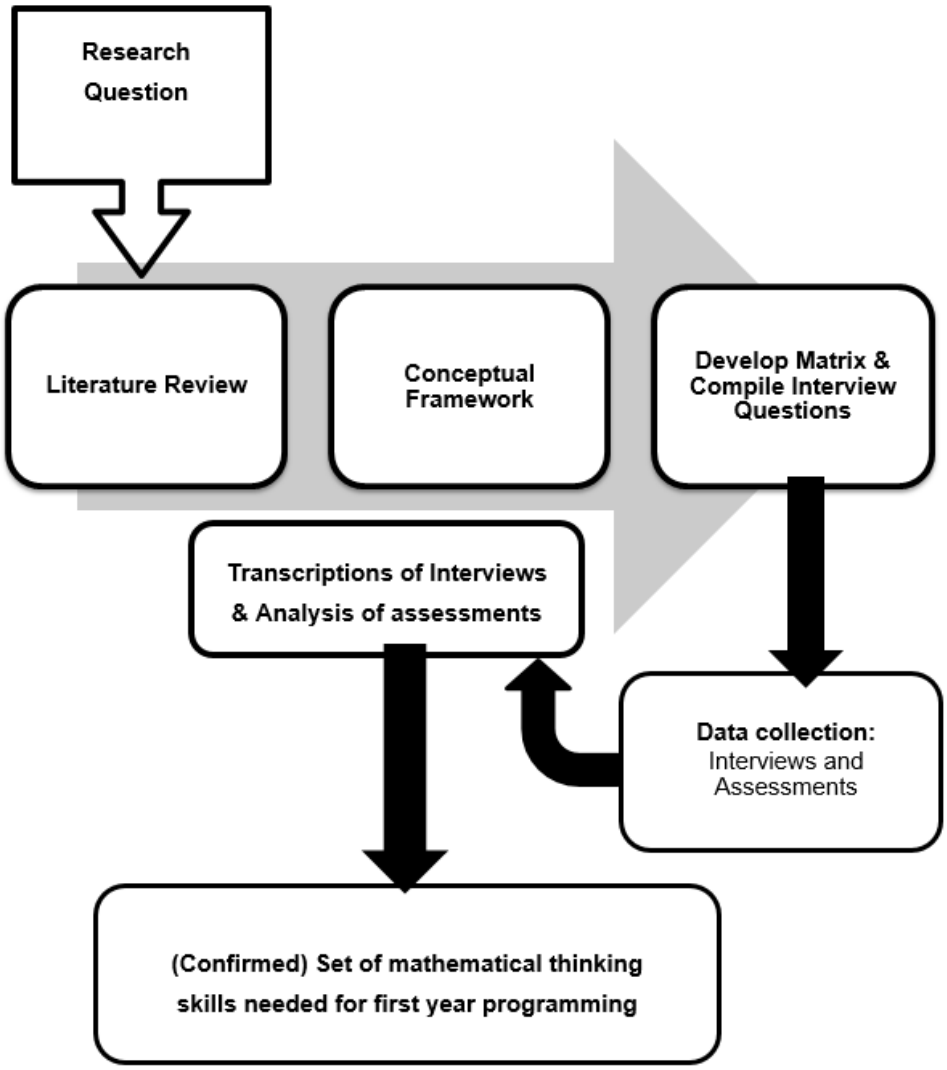



Figure 3.1: Research process followed



© University of Pretoria

### 3.4.2 Data Collection and Documentation

Considering the nature of the study, the data was mostly text-based (see Figure 3.2).

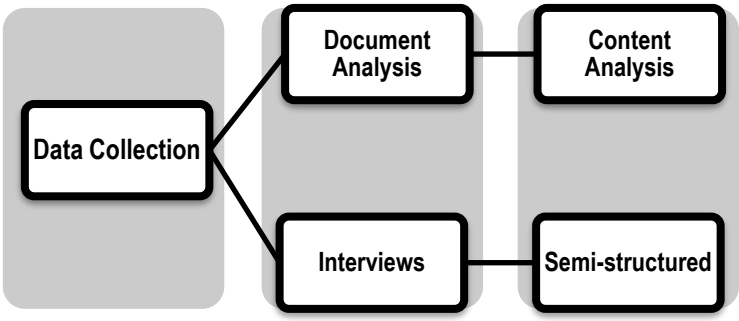



Figure 3.2: Data Collection

### 3.4.3 Document Analysis and Content Analysis

Document analysis is "part of the larger domain of documentary research methods" (Wagner *et al*., 2012, p. 140). The content of the documents (examination papers) of the first-year programming subject at TUT, Development Software 1 which is offered in the National Diploma in ICT, was examined. Content analysis entails "an inductive and iterative process" that is used to identify similarities and variances in text that would substantiate, or even disprove, a proposed theory (Nieuwenhuis, 2013, p. 101). However, the researcher used a deductive approach to confirm the conceptual framework as derived from a literature review. Content analysis therefore assisted the researcher to peruse the data from different points of view in order to identify key elements that could lead to improved interpretation of the data (Nieuwenhuis, 2013).

Examination papers of the first-year programming subject, Development Software, were obtained from the examiner. The subject consists of two modules, module A and module B, and both modules are offered in every semester due to the large number of students repeating the subject. The subject is a prerequisite for second-year programming subjects, and failing it extends students' studies considerably. Examination papers from both modules A and B for 2014 and 2015 were obtained. The policy at TUT (since 2015) is that examiners must compile three examination papers per subject (or module) per examination period. Therefore, 12 papers from 2015, and



© University of Pretoria

four (main and supplementary) from 2014 examination papers were available for analysis. In order to guarantee fair assessment practices, the structure of the examination papers for a subject (or module) for a specific examination period, are similar but not identical. Due to the similar structure of the examination papers, only two papers of each module were analysed; papers of the A-module were selected from 2014 and papers of the B-module were selected from 2015.

After an extensive literature review and careful consideration, a matrix was compiled to identify the particular mathematical and programming thinking skills when analysing programming assessments (see Table 3.2). It is believed that analysis of the content of these documents will lead towards a better understanding of the mathematical and programming thinking skills required of first-year programming students to succeed. The skills listed in the matrix resulted from an extensive literature review and are aligned with the conceptual framework. Examination papers were also analysed to identify the necessary mathematical and programming thinking skills.

Table 3.2: Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments

| **Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments** | | | | | |
|---|---|---|---|---|---|
| **Mathematical Thinking Skills** | **Questions** | | | | **Programming Thinking Skills** |
| | **1** | **2** | **3** | **4** | |
| **The manipulation of formulae** | | | | | **Problem-solving skills** |
| **Making use of mathematics to represent realities** | | | | | |
| **Behaviour of functions** | | | | | |
| **Breaking down and simplifying problems** | | | | | **Algorithmic thinking** |
| **Algorithmic thinking** | | | | | |
| **Generalisation of concepts/problems** | | | | | |



© University of Pretoria

| | | | | | |
|---|---|---|---|---|---|
| **Information structures** | | | | | **Reasoning skills** |
| **Abstract reasoning** | | | | | |

### 3.4.4 Interviews

In the present study, the researcher made use of interviews. In this regard, "Qualitative interviewing is a flexible and powerful tool to capture the voices and the ways people make meaning of their experience." (Rabionet, 2011, p. 563). Interviews have the advantage that the researcher can engage with the interviewee in person and be in control of the research situation. One of the disadvantages of interviews is the fact that it can be time-consuming and costly (Wagner *et al*., 2011). For the purpose of the present study, the interviewees (sample) were chosen within close proximity to the researcher in order to control possible cost and time constraints.

Thematising, as described by Kvale (2011), see Table 3.1, was applied when interviews were conducted in order to identify the mathematical thinking skills needed for first-year programming. Participants for these interviews were selected from the lecturers involved in the teaching of ICT modules at TUT. This convenience purposive sampling included four lecturers (excluding the examiners and moderators) with at least two years' lecturing experience of first-year programming subjects.

The semi-structured interviews consisted of a set of open-ended questions related to possible mathematical thinking skills used in programming. During the design stage of the interviews, the questions were carefully selected and planned so that the interviewer could lead the participants and therefore try to identify other lines of inquiry to be examined at a later stage (Kvale, 2011; Nieuwenhuis, 2013).

Invitations to participate in interviews were sent via e-mail to the examiners, moderators and lecturers of the subject Development Software I. Invitations were also extended to lecturers who have previously lectured the subject for many years. Two examiners, two moderators and two of the previous lecturers promptly replied and appointments were made, but only two of the current first-year lecturers responded.



© University of Pretoria

These two lecturers also failed to keep their appointments. Invitations were then extended to two second- and third-year programming lecturers as well as to two lecturers who have lectured first-year programming in the past. These lecturers responded and appointments were made. Interviews were scheduled in time slots chosen by the participants during the period between 16 May and 2 June 2016.

The interviews were recorded and the researcher made field notes while recording the interviews. Hereafter, the interviews were transcribed verbatim and stored digitally as well as in print-form to compare with the audio recordings and the notes made. Although the interviews lasted between 10 and 15 minutes each, one-hour time-slots were booked so that the researcher could first make casual conversation to ensure the participants felt relaxed and did not feel threatened in any way. The interviewer (researcher) compiled and followed an interview guide as suggested by Kvale (2011) to ensure that the exact same questions were asked to all the participants.

The researcher ensured that she was properly prepared for each interview. While interviews were recorded, the interviewer did not only rely on recordings, and also made field notes while conducting the interviews. The researcher ensured that the recording device was tested and in working order before commencing each interview. The researcher followed Kvale's (2011) recommendations and briefed the interviewees about the purpose and format of the interview to establish rapport with the interviewees. After the interview, she debriefed the interviewees by summarising what she had learned from the interviewee and asked if the interviewee had more to add (Kvale, 2011). Recordings and field notes were combined into a transcript (36 pages) of the interviews that was used for analysis, the reporting of the results follows in Chapter 4 of this dissertation. The last stage, namely verification of the results, was done as described in paragraph 3.5 (Kvale, 2011).



© University of Pretoria

Table 3.3: Kvale's seven stages of an interview inquiry were used for this study (Kvale, 2011, p. 35-36):

| | |
|---|---|
| **Thematising** | • **Identification of the mathematical and programming thinking skills.** |
| **Designing** | • **Planning the interview with lecturers, examiners and moderators.** |
| **Interview itself** | • **An interview guide was used for each interview, each interview lasted approximately 15 minutes.** |
| **Transcription** | • **All the interviews were transcribed verbatim, resulting in 36 pages of transcripts.** |
| **Analysis** | • **The need for mathematical thinking skills for first-year programming students, was confirmed.** |
| **Verification** | • **Reliability and validity checks were attempted, see paragraph 3.5.** |
| **Reporting** | • **The results are discussed in Chapter 4 of this dissertation.** |

## 3.5 Data Analysis and Interpretation

Malterud (2001) described qualitative research as a process of reflection that delivers knowledge that can be challenged, shared and transferred. Qualitative data analysis is therefore characterised by interpretation (Cohen, Manion & Morrison, 2011). Data analysis in qualitative research is traditionally an interpretive and inductive process. However, as mentioned before, the researcher followed a deductive process in an attempt to identify mathematical thinking skills related to programming thinking skills.



© University of Pretoria

### 3.5.1 Trustworthiness and Credibility

At the onset, the researcher's frame of reference and experience enabled a mutual meaning and understanding of the phenomena that were explored, and this helped both the interviewees and the researcher. Babbie and Mouton (2012) explain that trustworthiness refers to the findings being neutral; for data and findings to be trustworthy, these must also be credible. Triangulation is applied when different sources and different methods are used to determine credibility (Babbie & Mouton, 2012). Data was collected from interviews and also derived from document analysis to enable the researcher to triangulate, interpret and question findings in order to ensure the trustworthiness and credibility (Malterud, 2001).

Credibility was further achieved through prolonged engagement by conducting additional interviews (with more experienced lecturers) until data saturation was achieved. The researcher also made use of peer debriefing. She discussed the study with experienced colleagues (from another university) and in the process confirmed the challenges that first-year programming students and lecturers face in terms of mathematical thinking skills (Babbie & Mouton, 2012). Member checking was done by asking the participants to review the transcripts for accuracy (as suggested by Babbie & Mouton, 2012). Furthermore, referential adequacy was obtained by making digital recordings of the semi-structured interviews for future reference (Babbie & Mouton, 2012).

### 3.5.2 Inter-rater reliability

Reliability of an instrument is reached once the results yielded when using the instrument repeatedly are the same every time it is used (Babbie & Mouton, 2012). The matrix that was used in the present study was regarded as a reliable instrument when tested by the researcher and a colleague (with 34 years of experience) who both analysed the same question (Question 1) of the same paper (Paper 4) and the results were satisfactorily comparable. The researcher first explained the matrix to the participant (the colleague) to ensure that the analysis process was understood. This process was followed to determine whether consistency could be demonstrated when



© University of Pretoria

using the matrix in the analysis of question papers. The results of the analysis follow below (see Figure 3.3 [result of researcher] and Figure 3.4 [result of colleague]). The measure of agreement between the researcher and her colleague was then calculated as a percentage - see Table 3.4 (Cohen, Manion & Morrison, 2013). The fact that a score of 100% was obtained indicate that inter-rater reliability was achieved.

The researcher and her colleague's analysis yielded the exact same result and this suggested that the instrument (matrix) is reliable and that the researcher is consistent in using the matrix.

Figure 3.3: Analysis of Question 1 of Paper 4 (Researcher)

| Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mathematical Thinking Skills | Question 1: TRUE and FALSE (Paper 4) - Researcher | | | | | | | | | | Programming Thinking | TOTALS | TOTALS |
| | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 | 1.6 | 1.7 | 1.8 | 1.9 | 1.10 | | | |
| The manipulation of formulae | | | | | | | | 1 | | 1 | Problem solving skills | 2 | 6 |
| Making use of mathematics to represent realities | | | | | | | | | | | | 0 | |
| Behaviour of functions | | 1 | 1 | | | | 1 | 1 | | | | 4 | |
| Breaking down and simplifying problems | | | | | | | | 1 | | | Algorithmic thinking | 1 | 6 |
| Algorithmic thinking | 1 | | 1 | | | 1 | | 1 | | 1 | | 5 | |
| Generalisation of concepts/problems | | | | 1 | | | | 1 | | 1 | Reasoning skills | 3 | 12 |
| Information structures | | | | 1 | 1 | | | | 1 | 1 | | 4 | |
| Abstract reasoning | 1 | 1 | | | | 1 | | 1 | | 1 | | 5 | |

Figure 3.4: Analysis of Question 1 of Paper 4 (Participant)

| Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mathematical Thinking Skills | Question 1: TRUE and FALSE (Paper 4) - Participant | | | | | | | | | | Programming Thinking | TOTALS | TOTALS |
| | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 | 1.6 | 1.7 | 1.8 | 1.9 | 1.10 | | | |
| The manipulation of formulae | | | | | | | | 1 | | 1 | Problem solving skills | 2 | 6 |
| Making use of mathematics to represent realities | | | | | | | | | | | | 0 | |
| Behaviour of functions | | 1 | 1 | | | | 1 | 1 | | | | 4 | |
| Breaking down and simplifying problems | | | | | | | | 1 | | | Algorithmic thinking | 1 | 6 |
| Algorithmic thinking | 1 | | 1 | | | 1 | | 1 | | 1 | | 5 | |
| Generalisation of concepts/problems | | | | 1 | | | | 1 | | 1 | Reasoning skills | 3 | 12 |
| Information structures | | | | 1 | 1 | | | | 1 | 1 | | 4 | |
| Abstract reasoning | 1 | 1 | | | | 1 | | 1 | | 1 | | 5 | |



© University of Pretoria

Table 3.4: Degree of Agreement (Cohen, Manion & Morrison, 2013).

$$\boldsymbol{Degree\ of\ Agreement\ (MTS) = \frac{Number\ of\ agreements}{Number\ of\ opportunities} \times 100}$$

$$\boldsymbol{Degree\ of\ Agreement\ (MTS) = \frac{10}{10} \times 100 = 100\%}$$

## 3.6 Ethical Considerations

Ethical clearance was applied for and granted by the University of Pretoria (UP) and then an application was handed in at TUT. TUT also granted ethical clearance for the study. Information leaflets, explaining what the study is about, combined with consent forms were compiled and all the participants signed these documents. Participation was voluntary and participants were informed that they could withdraw at any stage of the research. Examples of all the relevant documentation can be found at the end of the dissertation (see Appendices F and G). Permission was also granted by the Head of the Computer Science Department, which is the custodian of the first-year programming subject of which the assessments were analysed in this study. The signed permission letter is attached (see Appendix E).

### 3.6.1 Data Collection: Document Analysis and Content Analysis

The documents in question (assessments, i.e. tests and examination papers) belong to the university (TUT). Although previous question papers are available to download from TUT's website, ethical clearance and permission was obtained to use these documents, and examiners and moderators signed consent forms that guaranteed them confidentiality and anonymity. No examiner or moderator was mentioned by name, and questions were selected without reference to the date of the paper or the name of the examiner and/or moderator.

### 3.6.2 Data Collection: Interviews

"An interview inquiry is a moral enterprise" (Kvale, 2011, p. 23). It is therefore pertinent to address all possible ethical concerns when planning and conducting interviews. Kvale (2011) states that ethical concerns may arise in any (or all) the seven stages of interview inquiry. The concerns specific to this study were addressed as follows:



© University of Pretoria

- The study was not only scientific, it also contributed to the field of study, namely first-year programming success.
- The study ensured confidentiality and anonymity, and made use of proper consent forms. Consequences were explained to the interviewees.
- Confidentiality of interviewees was protected and transcripts were true to the oral statements made. Pseudonyms were used in the reporting to ensure confidentiality and anonymity.
- The interviewees were invited to corroborate their statements before publication, and were invited to validate their inputs (member checking).

## 3.7 Summary

After identifying the research question, an extensive literature review was conducted from which a conceptual framework was compiled. A matrix (Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments) was developed and used to analyse assessments. This was done to confirm the validity of the conceptual framework. Semi-structured interviews were conducted to determine whether the identified mathematical thinking skills were also known and relevant to first-year programming lecturers, examiners and moderators.



© University of Pretoria

# Chapter 4: Results and findings

"Any time scientists disagree, it's because we have insufficient data. Then we can agree on what kind of data to get; we get the data; and the data solves the problem. Either I'm right, or you're right, or we're both wrong. And we move on."

(DeGrasse Tyson, N., URL: http://www.brainyquote.com/quotes/keywords/data.html)

## 4.1 Introduction

The primary research question under investigation is: *Which mathematical thinking skills do first year students, studying towards a National Diploma in ICT, need?* In order to fully explore the primary research question, the following secondary question is also addressed: *Which mathematical thinking skills are related to programming thinking skills?* An analytical matrix was developed from the conceptual framework (Figure 3.20) to analyse the question papers. During the analysis of the semi-structured interviews, an elemental approach was followed and codes were identified by utilising in vivo coding and themes were also developed (Saldaña, 2013). The next section explains this analysis process.

## 4.2 Analysis of Question Papers

The memorandum was used to identify the mathematical thinking skills in each question. Each question in the four selected question papers was analysed with a view to identify the mathematical thinking skills needed to answer the question. The structure of the 2014 (module A) question papers, that were analysed, included five theory questions (pseudo code) and one practical question (coding). The mark allocations and questions were as follows:

- Question 1: TRUE or FALSE questions (10)
- Question 2: Multiple Choice Questions (10)
- Question 3: Write, rewrite or change pseudo code segments of algorithms (5)
- Question 4: Predict the output of algorithms written in pseudo code (5)
- Question 5: Write complete algorithms (30)
- Question 6: Practical (VB.NET or C++): Code a complete program (50)



© University of Pretoria

The structure of the 2015 (module B) question papers that were analysed included four theoretical questions (pseudo code) and two practical questions (coding). The mark allocations and questions were as follows:

- Question 1: TRUE or FALSE questions (10)
- Question 2: Multiple Choice Questions (10)
- Question 3: Predict the output of algorithms written in pseudo code (2 and 7 marks)
- Question 4: Write complete algorithms (38 and 43 marks)
- Questions 5 and 6: Practical (VB.NET or C++): Code a complete program (40+10 = 50)

Every question type in the four papers was analysed by identifying and counting the number of times the mathematical and programming thinking skills were needed in order to answer the questions. The papers are referred to as Paper 1 to Paper 4, where Papers 1 and 2 were selected from 2014 (A module) and Papers 3 and 4 from 2015 (B module). The results, per paper, per type of question, were captured in MS Excel spreadsheets of which an example, of Question 1 of Paper 1, is displayed in Figure 4.1. An example of each question type is discussed below to demonstrate how the mathematical thinking skills were identified (see Tables 4.1 to 4.13).

It is important to take note of the following conventions and technical points in programming:

- The tilde (~) is used to indicate comments in an algorithm; all text after the ~ is not executed.
- The equal to sign (=) in programming is used to assign the calculations to the right of the equal to sign to the variable on the left of it.



© University of Pretoria

| Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments | | | | | | | | | | | | TOTALS | TOTALS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mathematical Thinking Skills | Question 1: TRUE and FALSE (Paper 1) | | | | | | | | | | Programming Thinking | | |
| | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 | 1.6 | 1.7 | 1.8 | 1.9 | 1.10 | | | |
| The manipulation of formulae | 1 | | 1 | | | 1 | | 1 | | | Problem solving skills | 4 | 9 |
| Making use of mathematics to represent realities | | | 1 | 1 | 1 | | | | | | | 3 | |
| Behaviour of functions | | | 1 | | 1 | | | | | | | 2 | |
| Breaking down and simplifying problems | 1 | | 1 | | 1 | | | 1 | | | Algorithmic thinking | 4 | 12 |
| Algorithmic thinking | 1 | | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | | 8 | |
| Generalisation of concepts/problems | 1 | | 1 | | 1 | | | 1 | | | Reasoning skills | 4 | 10 |
| Information structures | | | | | | | | | | | | 0 | |
| Abstract reasoning | 1 | | 1 | 1 | 1 | | 1 | 1 | | | | 6 | |

**Mathematical thinking skills**

**Programming thinking skills**

Figure 4.1: Example of Analysis in MS Excel:

All the examination papers were analysed as demonstrated in the following sections and the results were captured and summarised in MS Excel. Table 4.8 presents a summary of the results. A total 96 questions were analysed in the four selected question papers. The number of times a thinking skill was identified was tallied and the results were converted into percentages.

### 4.2.1 Question 1: True or False

In Question 1, the students had to evaluate statements in order to determine whether the statements were True or False. This section consists of 10 short questions. Table 4.1 explains how question 1.8 was analysed. The aim of this question is to test the students' insight into concepts and terminology that are closely related as well as the logical flow of code segments.



© University of Pretoria

Table 4.1: Example of Question 1.8 (Paper 1)

Indicate whether the following statement is TRUE or FALSE:

The following 2 loops are equivalent:

```
counter = 1
do while counter < 10
    display counter
    counter = counter + 1
loop
```

and

```
for counter = 1 to 10
     display counter
next counter
```

The correct answer is FALSE because the first loop will not execute for the counter value of 10; only from 1 to 9, and the second loop for a counter value from 1 to 10. In order to arrive at the solution, the following thinking is required:

- The manipulation of formulae, for example:

| | |
|---|---|
| `counter = 1` | ~The value 1 is assigned to the variable **counter** |
| `do while counter < 10` | ~The value of **counter** is tested, if it is TRUE, the loop ~will be executed, if not, it will execute the first statement ~after the end of the loop |
| `display counter` | ~The value of **counter** is displayed on the screen |
| `counter = counter + 1` | ~The value of **counter** is increased by 1 |
| `loop` | ~End of the loop |

- Generalisation of concepts/problems: variables are used to represent values (see the above example and explanation of changing values of variables). The concept of the Do-While loop can be generalised and applied in other scenarios.
- Breaking down and simplifying problems: these segments are part of a larger algorithm/solution. In order to solve (programming) problems, problems are broken down in smaller segments to simplify the solutions.



© University of Pretoria

- Abstract reasoning: it is expected of the student to compare the two segments of pseudo code and determine the output of both to decide whether these segments deliver the same result (or not). In the **`for`** loop, the value of counter changes without an assignment statement (=); this concept can also be regarded as abstract.
- Algorithmic thinking: the steps in the above pseudo code are in a specific sequence which will have a different result if the sequence is changed. For example, if the statement **`counter = 1`** is omitted from the first loop, the value of counter will be unknown and cannot be tested in the **`do while`** statement.

The results for the entire Question 1 (Paper 1) are summarised in Table 4.2. However, for students to answer Question 1.8 (True or False) specifically, they require five mathematical thinking skills (see below). These skills are also required in the other questions and are indicated in brackets.

- The manipulation of formulae (1.1, 1.3 and 1.6);
- Generalisation of concepts/problems (1.1, 1.3 and 1.5);
- Breaking down and simplifying problems (1.1, 1.3 and 1.5);
- Abstract reasoning (1.1, 1.3, 1.4, 1.5 and 1.7) and
- Algorithmic thinking (1.1, 1.3, 1.4, 1.5, 1.6, 1.9 and 1.10).

Additional mathematical thinking skills that are required for the other questions (excluding 1.8), are:

- Making use of mathematics to represent realities (1.3, 1.4 and 1.5);
- Behaviour of functions (1.3 and 1.5).



© University of Pretoria

Table 4.2: Results: Analysis of Question 1 (Paper 1)

| Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments | | | | | | | | | | | | TOTALS | TOTALS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mathematical Thinking Skills | Question 1: TRUE and FALSE (Paper 1) | | | | | | | | | | Programming Thinking | | |
| | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 | 1.6 | 1.7 | 1.8 | 1.9 | 1.10 | | | |
| The manipulation of formulae | 1 | | 1 | | | 1 | | 1 | | | Problem solving skills | 4 | 9 |
| Making use of mathematics to represent realities | | | 1 | 1 | 1 | | | | | | | 3 | |
| Behaviour of functions | | | 1 | | 1 | | | | | | | 2 | |
| Breaking down and simplifying problems | 1 | | 1 | | 1 | | | 1 | | | Algorithmic thinking | 4 | 12 |
| Algorithmic thinking | 1 | | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | | 8 | |
| Generalisation of concepts/problems | 1 | | 1 | | 1 | | | 1 | | | Reasoning skills | 4 | 10 |
| Information structures | | | | | | | | | | | | 0 | |
| Abstract reasoning | 1 | | 1 | 1 | 1 | | 1 | 1 | | | | 6 | |

In summary, the most important skills required for Question 1.8 specifically, are abstract reasoning and algorithmic thinking. These are demonstrated by the students' insight into concepts and terminology (generalisation) that are closely related (abstract reasoning) as well as the logical flow and execution of code segments (algorithmic thinking).

### 4.2.2 Question 2: Multiple Choice

Question 2 is made up of 10 short multiple choice questions. Table 4.3 explains how question 2.8 was analysed. As was the case with the True or False questions, the aim of this question is to test the students' insight into concepts and terminology that are closely related as well as the logical flow of code segments. The accurate execution of statements is also tested in this case.

Table 4.3: Example of Question 2.8 (Paper 1)

What will be the value of `total` after the following statements have been executed?

```
total = 0

for x = 1 to 15 step 3

     total = total + 2
     do while total < 12
           total = total + 4
     loop

next x
```



© University of Pretoria

```
total = total + x
```

a. 16
b. 22
c. 38
d. 40
e. None of the above

The correct answer is **c. 38** as demonstrated below in the trace table (Table 4.4):

Table 4.4: Trace table to demonstrate processing of the algorithm in Table 4.3

| **total** | **x** | **Statements** |
|---|---|---|
| **0** | | `total = 0` |
| | **1** | `for x = 1 to 15 step 3` |
| **2** | | `total = total + 2` |
| **6** | | `do while total < 12`<br>`total = total + 4`<br>`loop` |
| **10** | | |
| **14** | | |
| | **4** | `for x = 1 to 15 step 3` |
| **16** | | `total = total + 2` |
| | **7** | `for x = 1 to 15 step 3` |
| **18** | | `total = total + 2` |
| | **10** | `for x = 1 to 15 step 3` |
| **20** | | `total = total + 2` |
| | **13** | `for x = 1 to 15 step 3` |
| **22** | | `total = total + 2` |
| | **16** | `next x` |
| **38** | | `total = total + x` |



© University of Pretoria

In order to arrive at the correct answer, the following thinking skills are required:

- The manipulation of formulae (see explanation (above) of calculation of the correct answer); the value of **`total`** and **`x`** are changed when the algorithm is executed.
- Making use of mathematics to represent realities: the application of calculation of totals is widely used in programs written for accounting, statistics and many other fields.
- Generalisation of concepts/problems: variables are used to represent (changing) values and the concept of loops (For-loop and Do While loop) can be generalised and applied in other scenarios as well.
- Breaking down and simplifying problems: this segment forms part of a larger algorithm/solution. Keeping track of processing of the loops also requires breaking down the problem into smaller parts.
- Algorithmic thinking: the steps in the above pseudo code are in a specific sequence which will yield a different result if the sequence is changed.
- Abstract reasoning: it is expected of the student to understand the nested loops and the execution of these.

The results for the entire Question 2 are summarised in Table 4.5. However, for students to be able to answer Question 2.8 (Multiple Choice) specifically, five mathematical thinking skills are required (see below). These skills are also required in the other questions and are indicated in brackets.

- The manipulation of formulae (all, but 2.7);
- Making use of mathematics to represent realities (only 2.8);
- Generalisation of concepts/problems (all, but 2.7);
- Breaking down and simplifying problems (2.1, 2.2, 2.9 and 2.10);
- Abstract reasoning (all) and
- Algorithmic thinking (all).

The most important skills are, once again, the manipulation of formulae, generalisation of concepts and problems, abstract reasoning and algorithmic thinking which can be demonstrated by the students' insight into concepts (generalisation) that are closely



© University of Pretoria

related (abstract reasoning) as well as the logical flow and execution of code segments (algorithmic thinking). It is also important that the students should be able to calculate the correct result and therefore the manipulation of formulae and the use of variables are also salient.

Table 4.5: Results: Analysis of Question 2 (Paper 1)

| Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments | | | | | | | | | | | | TOTALS | TOTALS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mathematical Thinking Skills | Question 2: Multiple Choice Questions (Paper 1) | | | | | | | | | | Programming Thinking | | |
| | 2.1 | 2.2 | 2.3 | 2.4 | 2.5 | 2.6 | 2.7 | 2.8 | 2.9 | 2.10 | | | |
| The manipulation of formulae | 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | Problem solving skills | 9 | 10 |
| Making use of mathematics to represent realities | | | | | | | | 1 | | | | 1 | |
| Behaviour of functions | | | | | | | | | | | | 0 | |
| Breaking down and simplifying problems | 1 | 1 | | | | | | 1 | 1 | 1 | Algorithmic thinking | 5 | 15 |
| Algorithmic thinking | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 10 | |
| Generalisation of concepts/problems | 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | 1 | Reasoning skills | 9 | 19 |
| Information structures | | | | | | | | | | | | 0 | |
| Abstract reasoning | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | 10 | |

### 4.2.3 Question 3: Predict the Output

In Question 3, the students were given segments of pseudo code which should result in specific output. Students must follow the steps correctly in order to determine the exact output that will be displayed if the code is executed. Table 4.6 explains how Question 3 was analysed.

Table 4.6: Example of Question 3 (Paper 3)

Study the following array, **`integer`**, with values for 7 elements. Indicate what the values in the array would be after the following code has been executed.

```
FillArray
    num = 3
    for element = 7 to 1 step -1
        integer(element-1) = element
    next element
    for j = 1 to 6 step 2
        integer(j) = num \ 2
```



© University of Pretoria

```
            num = num + 1
            integer(num) = integer(j) mod 3
        next j
    end
```

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | |

The correct answer is:

| | | | | | | |
|---|---|---|---|---|---|---|
| **1** | **1** | **3** | **2** | **1** | **2** | **2** |

At first, there are no values in the array. The elements of the array are calculated by the first for-loop. Elements are added from the last to the first position and the value of the element is 1 less than the position:

```
for element = 7 to 1 step -1
     integer(element-1) = element
next element
```

After the first for-loop was executed (7 times), the array will look as follows:

| | | | | | | |
|---|---|---|---|---|---|---|
| **0** | **1** | **2** | **3** | **4** | **5** | **6** |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** |

**Index of element (position in array)**

In the second for-loop, the values of the elements are changed as follows:

An initial value of 3 was assigned to the variable `num`; after the first execution of the loop, the second element of the array has changed to 1



© University of Pretoria

```
for j = 1 to 6 step 2                    ~j = 1
    integer(1) = 3 \ 2                   ~ the second element (element in the second
                                         ~ position, index is 1) of the array changes
    num = 3 + 1                          ~ the value of num is increased by 1
    integer(4) = integer(1) mod 3        ~ The fifth element (position 4) changes to 1
next j                                   ~ End of the loop
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 3 | 4 | 1 | 6 | 7 |

```
for j = 1 to 6 step 2                    ~j = 3
    integer(3) = 4 \ 2                   ~ the third element of the array changes to 2
    num = 4 + 1                          ~ the value of num is increased by 1
    integer(5) = integer(3) mod 3        ~ The sixth element changes to 2
next j                                   ~ End of the loop
```

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 3 | 2 | 1 | 2 | 7 |

```
for j = 1 to 6 step 2                    ~j = 5
    integer(5) = 5 \ 2                   ~ the sixth element of the array  is changed
                                         ~  and becomes 2
    num = 5 + 1                          ~ the value of num is increased by 1
    integer(6) = integer(5) mod 3        ~ The sixth element changes to 2
next j                                   ~ End of the loop
```



© University of Pretoria

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 3 | 2 | 1 | 2 | 2 |

In order to arrive at the correct answer, the following thinking skills are required:

- The manipulation of formulae: different values are assigned to variables during execution, and the output will change when the values change. For example, in the following segment, the values of **`integer(5)`** and **`integer(6)`** are changed:

```
for j = 1 to 6 step 2
   integer(5) = 5 \ 2
   num = 5 + 1
   integer(6) = integer(5) mod 3
next j
```

- Breaking down and simplifying problems: students need to work through the pseudo code in two phases. They first have to complete the first for-loop and then work through the second for-loop.
- Algorithmic thinking: the sequence of the steps is important to determine the correct answer. If the sequence is changed, the output will change. Once again, if the statements **`integer(5) = 5 \ 2`** and **`integer(6) = integer(5) mod 3`** in the above segment of code is swopped around, the result will not be correct.
- Generalisation of concepts/problems: several variables are used in the algorithm and the values are changed when the algorithm executes. The concept of an array can also be applied in many other scenarios.
- Information structures: an array is used and arrays are structures used in programming in calculations to produce relevant and useful information.



© University of Pretoria

- Abstract reasoning: the use of arrays is an abstract concept and, according to the first-year programming lecturers at TUT, first-year programming students tend to struggle with its abstract nature.

The results for the entire Question 3 are summarised in Table 4.7. To be able to answer questions like Question 3, students require the following mathematical thinking skills:

- The manipulation of formulae;
- Generalisation of concepts/problems;
- Breaking down and simplifying problems;
- Information structures;
- Abstract reasoning and
- Algorithmic thinking.

All the above-mentioned skills are applied and used in answering the above question and all of the skills are important. Therefore, one cannot isolate one of the skills as being more important than another.

Table 4.7: Results: Analysis of Question 3 (Paper 3)

| Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments | | | TOTALS | TOTALS |
|---|---|---|---|---|
| **Mathematical Thinking Skills** | **Question 3 (Paper 3)** | **Programming Thinking** | | |
| The manipulation of formulae | 1 | Problem solving skills | 1 | 1 |
| Making use of mathematics to represent realities | | | 0 | |
| Behaviour of functions | | | 0 | |
| Breaking down and simplifying problems | 1 | Algorithmic thinking | 1 | 2 |
| Algorithmic thinking | 1 | | 1 | |
| Generalisation of concepts/problems | 1 | Reasoning skills | 1 | 3 |
| Information structures | 1 | | 1 | |
| Abstract reasoning | 1 | | 1 | |



© University of Pretoria

### 4.2.4 Question 4: Predict the Output

Question 4 is similar to Question 3; the students were given segments of pseudo code which should result in specific output. They had to follow the steps correctly in order to determine the exact output that will be displayed if the code is executed. Table 4.8 explains how Question 4 was analysed.

Table 4.8: Example of Question 4 (Paper 2)

Predict the exact output if the following code is processed:

```
ShowAnswer
     a = 6
     b = 12
     counter = 1
     for c = 1 to 12 step 6
          display "Round ", counter ~ on a new line
          do while c < b
               b = b - 2
          loop
          if a > c then
               a = a - 1
          else
               a = a + 1
          endif
          counter = counter + 1
     next c
     display "a = ", a              ~ on a new line
     display "b = ", b              ~ on a new line
     display "c = ", c              ~ on a new line
end
```



© University of Pretoria

The correct answer is below, as demonstrated below in the trace table (Table 4.9).

The correct answer is:

Round 1

Round 2

a = 6

b = 0

c = 13

Table 4.9: Trace table to demonstrate processing of the algorithm in Table 4.8

| a | b | c | counter | Statements | Display |
|---|---|---|---|---|---|
| 6 | | | | `a = 6` | |
| | 12 | | | `b = 12` | |
| | | | 1 | `counter = 1` | |
| | | 1 | | `for c = 1 to 12 step 6` | |
| | | | | `display "Round ", counter ~ on a new line` | Round 1 |
| | 10 | | | `do while c < b`<br>`  b = b - 2`<br>`loop` | |
| | 8 | | | | |
| | 6 | | | | |
| | 4 | | | | |
| | 2 | | | | |
| | 0 | | | | |



© University of Pretoria

| | | | | | |
|---|---|---|---|---|---|
| 5 | | | | `if a > c then`<br>`  a = a - 1`<br>`else`<br>`  a = a + 1`<br>`endif` | |
| | | | 2 | `counter = counter + 1` | |
| | | 7 | | `next c` | |
| | | | | `display "Round ", counter ~ on a new line` | Round 2 |
| 6 | | | | `if a > c then`<br>`  a = a - 1`<br>`else`<br>`  a = a + 1`<br>`endif` | |
| | | | 3 | `counter = counter + 1` | |
| | | 13 | | `next c` | |
| | | | | `display "a = ", a ~ on a new line` | a = 6 |
| | | | | `display "b = ", b ~ on a new line` | b = 0 |
| | | | | `display "c = ", c ~ on a new line` | c= 13 |

In order to arrive at the correct answer, the following thinking skills are required:

- The manipulation of formulae: different values are assigned to the variables in the algorithm, for example **`b = b - 2`** and **`counter = counter + 1`**.



© University of Pretoria

- Breaking down and simplifying problems: the students need to work through the pseudo code in two phases. They first have to complete the first for-loop and then work through the second for-loop.
- Algorithmic thinking: the sequence of the steps is important to determine the correct answer if the sequence is changed, the result (output) will be different.
- Generalisation of concepts/problems: variables and loops are used. These can be used in different ways in different scenarios.
- Abstract reasoning: the students must work through the pseudo code and calculate the correct answer.

The results for the entire Question 4 are summarised in Table 4.10. To be able to answer questions like Question 4, students require the following mathematical thinking skills:

- The manipulation of formulae;
- Generalisation of concepts/problems;
- Breaking down and simplifying problems;
- Abstract reasoning and
- Algorithmic thinking.

All the above-mentioned skills are applied and used in answering the above question. The skills are all important and one cannot isolate one of the skills as being more important than another.

### 4.2.5 Question 4: Write a complete algorithm

Writing algorithms to solve a real-world problem in pseudo code is usually Question 5 in the A-module (Papers 1 and 2) and in the B-module (Papers 3 and 4) it is Question 4. It is expected of the students to incorporate all knowledge and skills they have acquired during the semester in order to answer these questions. See Table 4.11 for an example.



© University of Pretoria

Table 4.10: Results: Analysis of Question 4 (Paper 2)

| Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments | | | Mathematical Thinking → TOTALS | Programming Thinking → TOTALS |
|---|---|---|---|---|
| **Mathematical Thinking Skills** | **Question 4 (Paper 2)** | **Programming Thinking** | | |
| The manipulation of formulae | 1 | Problem solving skills | 1 | 1 |
| Making use of mathematics to represent realities | | | 0 | |
| Behaviour of functions | | | 0 | |
| Breaking down and simplifying problems | 1 | Algorithmic thinking | 1 | 2 |
| Algorithmic thinking | 1 | | 1 | |
| Generalisation of concepts/problems | 1 | Reasoning skills | 1 | 2 |
| Information structures | | | 0 | |
| Abstract reasoning | 1 | | 1 | |

Table 4.11: Example of Question 4 (Paper 4)

The Med Rep Company, a company promoting and selling medical supplies to doctors and pharmacies, has tasked you with the development of a program to keep track of the sales for the seven salespersons in their employ in a week. Five parallel one dimensional arrays contain the following information about each salesperson: (These arrays are global, and you can assume that the first three arrays already contain values)

| salesPersNum | salesPersName | yrsExperience | weeklySales | commEarned |
|---|---|---|---|---|
| 739011 | M Gadebe | 5 | 0 | 0 |
| 437863 | K Muller | 21 | 0 | 0 |
| 912565 | P Nkosi | 8 | 0 | 0 |
| 801243 | L Brown | 3 | 0 | 0 |
| 551278 | K Motshabi | 12 | 0 | 0 |
| 221453 | Y Chuene | 4 | 0 | 0 |
| 664475 | H Phiri | 15 | 0 | 0 |

At the beginning of the week, all the weekly sales amounts must be initialised to 0, as well as the commission earned. Do this in a sub-procedure. Throughout the week, all sales are entered per salesperson. A salesperson number of -1 must indicate the end of the week and terminate the process.



© University of Pretoria

For each sale, enter the salesperson number of the salesperson who sold the car, as well as the sales amount that the car was sold for. Then, call a function that will search through the array for the salesperson number. If it is found, the subscript where the number was found must be returned. If not, a value of -1 must be returned.

The total sales of the salesperson at that subscript must be accumulated if found. If not, the user must be prompted for a correct salesperson number until one is entered. When all the sales amounts for the week have been entered, do the following:

- Calculate the commission for each of the salespersons in a sub procedure, and place it in the Commission Earned array (the fifth array). Display these, along with the salesperson name, in the main module. The commission earned is calculated as follows:

| Years' Experience | Commission Earned |
|---|---|
| 0 – 5 years | 3% on the first 100 000, 5% on sales > 100 00 |
| 6 – 10 years | 7.5% on the first 100 000, 10% on sales > 100 000 |
| More than 10 years | 12% on the first 100 000, 15% on sales > 100 000 |

- Determine the name of the salesperson that had the highest sales for the week, as well as the total sales for that salesperson in a sub procedure, and display it in the main module.

Due to the length of the suggested solution, it is included in Appendix B and only segments of the solution are included in the discussion for clarity. In order to arrive at the correct answer, the following thinking skills are required:

- The manipulation of formulae, the values of variables changed during execution of the algorithm.

```
for sp = 1 to 6

     if weeklySales(sp) > highestSales then

          highestSales = weeklySales(sp)

          highestSalesPName = salesPersName(sp)

     endif

next sp
```

- Making use of mathematics to represent reality: the algorithm must solve a real-world problem, namely calculating commission of salespeople.



© University of Pretoria

- Behaviour of functions: the question states that the students must use a function to return a value. An example of a function is shown below. A function does calculations and always returns only one value to the statement where the function is called, for example: `subscr = FindSalesPNum (salesPNum)`

```
function FindSalesPNum (valSalesPNum)
sp = 0
found = false
while sp <= 6 AND found = false
     if salesPersNum(sp) = valSalesPNum then
          found = true
          subscr = sp
     endif
loop
if found = false then
     subscr = -1
endif
return subscr
```

- Breaking down and simplifying problems: the students need to break up a larger problem in smaller parts, in this case sub procedures and functions are used.
- Algorithmic thinking: the students need to write the algorithm in such a way that it solves the problem correctly. Once again, the sequence of the steps is important to determine the correct results.
- Generalisation of concepts/problems: variables and various other programming concepts are used that can be applied in other scenarios as well.
- Information structures: an array is used and arrays are data structures (as indicated previously).



© University of Pretoria

- Abstract reasoning: the students must write the algorithm in pseudo code to solve a real-world problem working with abstract concepts like variables and arrays.

The results for Questions 4 and 5 are summarised in Table 4.12. In summary, to be able to answer Question 4 students require the following mathematical thinking skills:

- The manipulation of formulae;
- Generalisation of concepts/problems;
- Breaking down and simplifying problems;
- Information structures;
- Abstract reasoning and
- Algorithmic thinking.

Table 4.12: Results: Analysis of Questions 4 and 5 (Paper 4)

| Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments | | | | TOTALS | TOTALS |
|---|---|---|---|---|---|
| Mathematical Thinking Skills | Questions 4 and 5 (Paper 4) | | Programming Thinking | | |
| | 4 | 5 | | | |
| The manipulation of formulae | 1 | 1 | Problem solving skills | 2 | 6 |
| Making use of mathematics to represent realities | 1 | 1 | | 2 | |
| Behaviour of functions | 1 | 1 | | 2 | |
| Breaking down and simplifying problems | 1 | 1 | Algorithmic thinking | 2 | 4 |
| Algorithmic thinking | 1 | 1 | | 2 | |
| Generalisation of concepts/problems | 1 | 1 | Reasoning skills | 2 | 6 |
| Information structures | 1 | 1 | | 2 | |
| Abstract reasoning | 1 | 1 | | 2 | |

Once again, all the above-mentioned skills are applied and used in answering the above question and none of the skills is more important than the other.



© University of Pretoria

### 4.2.6 Questions 5 and 6: Practical (coding in VB.Net or C++)

Writing VB.Net or C++ code to solve a real-world problem is usually Question 6 in the A-module (Papers 1 and 2) and in the B-module (Papers 3 and 4) it is Question 4. It is expected of the students to incorporate all the knowledge and skills they have acquired during the semester when answering these questions, see Table 4.13 for an example.

Table 4.13: Example of Question 5 (Paper 4): Code the following design into C++ / VB

Create 6 global parallel arrays, as follows, to be used in the program:

The solution is a loan billing system for a bank to calculate the monthly amount each account holder must pay on their account. The information about each model is stored in 5 parallel arrays, with a sixth parallel array **accMonthDue**, initialised to 0 and to be filled with the result of the calculations:

| accNum | accName | accTotDue | accRemTerm | accIntRate | accMonthDue |
|---|---|---|---|---|---|
| 813465 | T Mahlangu | 1200.00 | 4 | 9% | 0 |
| 548926 | R Pretorius | 1570.00 | 5 | 12% | 0 |
| 662908 | K More | 5600.00 | 4 | 7% | 0 |
| 713890 | F Knauff | 5200.00 | 6 | 10% | 0 |
| 412579 | V Raphiri | 3200.00 | 2 | 6% | 0 |
| 916356 | H Raseroka | 6600.00 | 10 | 15% | 0 |
| 528349 | M Malebane | 13000.00 | 1 | 11% | 0 |
| 920345 | Z Kgoete | 1500.00 | 3 | 8% | 0 |
| 710293 | S Meintjes | 4800.00 | 10 | 13% | 0 |
| 820394 | T Radebe | 2000.00 | 4 | 9% | 0 |

For testing purposes, you can assume that there are 10 account holders.

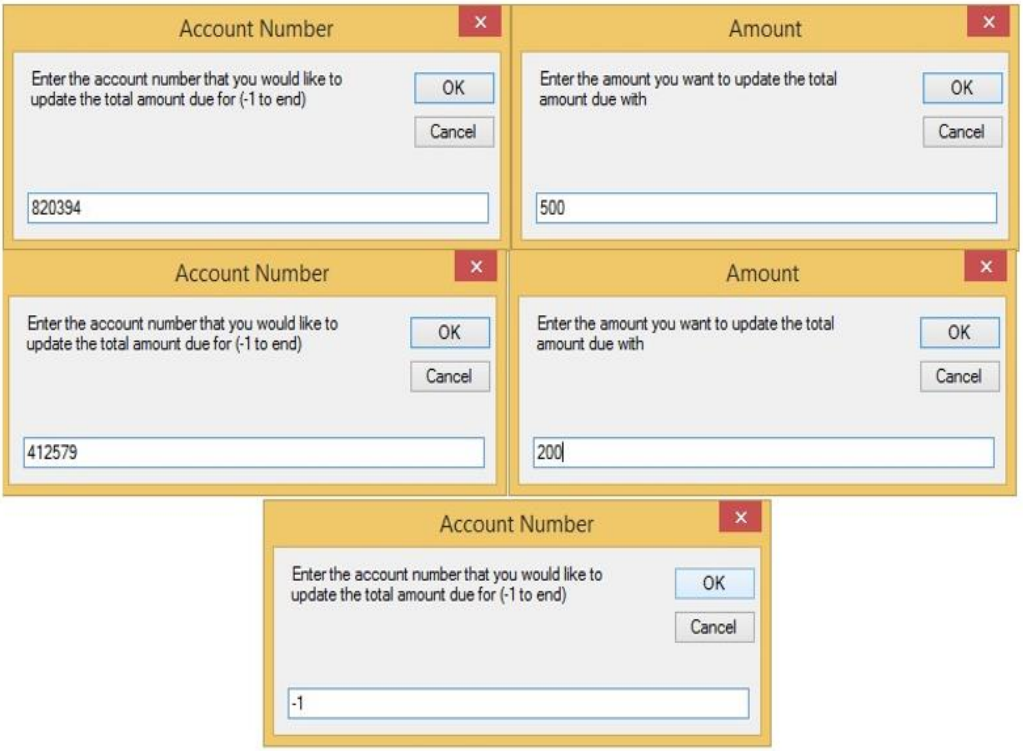


© University of Pretoria

E:\exam2015s2\paperc.exe

```
Enter the account number that you would like
to update the total amount due for (-1 to end): 820394
Enter the amount you want to update the total amount due with: 500

Enter the account number that you would like
to update the total amount due for (-1 to end): 412579
Enter the amount you want to update the total amount due with: 200

Enter the account number that you would like
to update the total amount due for (-1 to end): 253694

Account number not found

Enter the account number that you would like
to update the total amount due for (-1 to end): -1

Account Number      Account Name    Total Still Due     Term Remaining  Monthly Amount Due
=============       ============    ===============     ==============  ==================

813465               T Mahlangu         R1200.00                 4            R-108.00
548926              R Pretorius         R1570.00                 5            R-188.40
662908                  K More          R5600.00                 4            R-392.00
713890                F Knauff          R5200.00                 6            R-520.00
412579               V Raphiri          R3000.00                 2            R-180.00
916356              H Raseroka          R6600.00                10            R-990.00
528349              M Malebane          R13000.00                1             R-1430.00
920345                Z Kgoete          R1500.00                 3            R-120.00
710293              S Meintjes          R4800.00                10            R-624.00
820394                T Radebe          R1500.00                 4            R-135.00

=====================================================================
Total AmountPayable = R43970.00
The name of the account holder with the highest
amount still due is M Malebane
=====================================================================
```

**Figure 5.1 – C++ Sample output**






© University of Pretoria

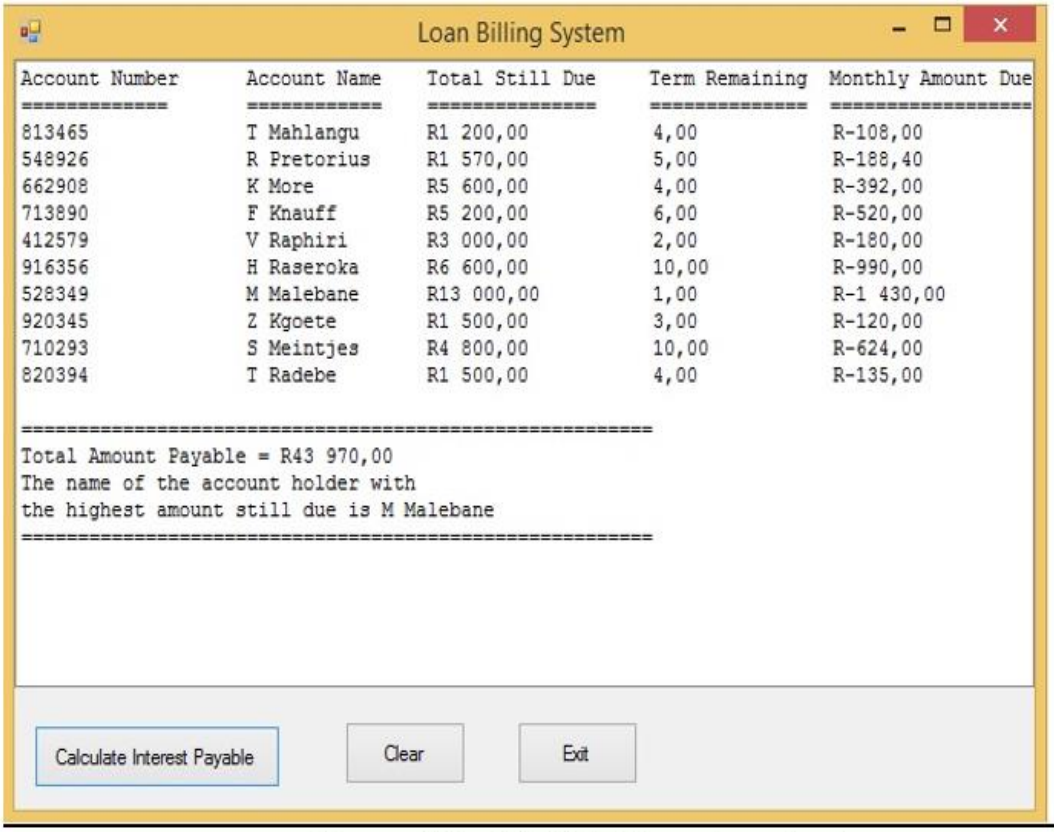



**Figure 5.2 – VB.NET Sample output**

Due to the length of the suggested solution, it is included in Appendix C. In order to find the solution, the following thinking skills are required:

- The manipulation of formulae: the values of variables change in the code to calculate the correct values.
- Making use of mathematics to represent reality: the program must solve a real-world problem, namely calculating people's loan payments.
- Behaviour of functions: students must use functions in order to return a single value per function.
- Breaking down and simplifying problems: the students need to break up a bigger problem in smaller parts, in this case sub-procedures and functions are used.
- Algorithmic thinking: the students need to write the code in such a way that it solves the problem correctly. Once again, the sequence of the steps is important to determine the correct results.
- Generalisation of concepts/problems: variables and various other programming concepts are used that can be applied in other scenarios as well.



© University of Pretoria

- Information structures: an array is used and arrays are data structures (as indicated previously).
- Abstract reasoning: the students must write the code in either C++ or VB.Net to solve a real-world problem working with abstract concepts like variables and arrays.

In summary, to be able to answer Question 5, students require the following mathematical thinking skills (see summary of results in Table 4.12):

- The manipulation of formulae;
- Making use of mathematics to represent reality;
- Behaviour of functions;
- Generalisation of concepts/problems;
- Breaking down and simplifying problems;
- Information structures;
- Abstract reasoning;
- Algorithmic thinking.

When programming on first-year level, all the above-mentioned skills are applied and used in answering the above question; therefore one cannot isolate a single skill as being more important than another.

### 4.2.7 Results of Analysis of Question Papers

All the examination papers were analysed as demonstrated in paragraph 4.2.1. The results were captured and summarised in MS Excel. Table 4.14 presents a summary of the results. A total 96 questions were analysed in the four selected question papers. The number of times a thinking skill was identified, these were tallied and the results were converted into percentages. The manipulation of formulae was present in 48 out of the 96 questions, and therefore 50% of the questions required the skill to manipulate formulae. Making use of mathematics to represent realities was identified in 17 of the 96 questions (17.71%) and the behaviour of functions 18 times in the 96 questions (18.75%). Mathematical thinking skills, the manipulation of formulae, making use of mathematics to represent reality and the behaviour of functions were also grouped as



© University of Pretoria

problem-solving skills under programming thinking skills, and 28.82% of the questions required these skills.

Breaking down and simplifying problems were present in 37 of the 96 questions (38.54%) and algorithmic thinking 60 times out of 96 (62.5%). Breaking down and simplifying problems and algorithmic thinking (as mathematical thinking skills) were grouped as algorithmic thinkings skills in programming, and 50.52% of the questions required these two thinking skills.

The mathematical thinking skill of generalisation of concepts/problems was present in 51 of the 96 questions (53.13%). The use of information structures was required in 16 times of the 96 questions (16.67%). Abstract reasoning as a mathematical thinking skill was identified as a required skill 57 times out of the possible 96 times (59.38%). These three mathematical thinking skills were also grouped as reasoning skills as part of programming thinking skills, and 43.06% of the questions required this skill.

The results of the analysis as summarised in Table 4.14 are also graphically represented in Figure 4.2 (mathematical thinking skills) and Figure 4.3 (programming thinking skills) below.

In the first graph (Figure 4.2), the eight mathematical thinking skills that were identified are represented in 96 different questions that were analysed (the light grey bars) and the number of times these skills were present in these 96 questions are indicated in the darker grey. In the second graph (Figure 4.3) the eight mathematical thinking skills were grouped together because they overlap with the three identified areas of programming thinking skills. The number of questions were therefore added and the possibilities were 288 (96+96+96), 196 (96+96) and 288 (96+96+96) respectively in the questions that were analysed (the light grey bars). The number of times these programming thinking skills were present in these questions is indicated in the darker grey.



© University of Pretoria

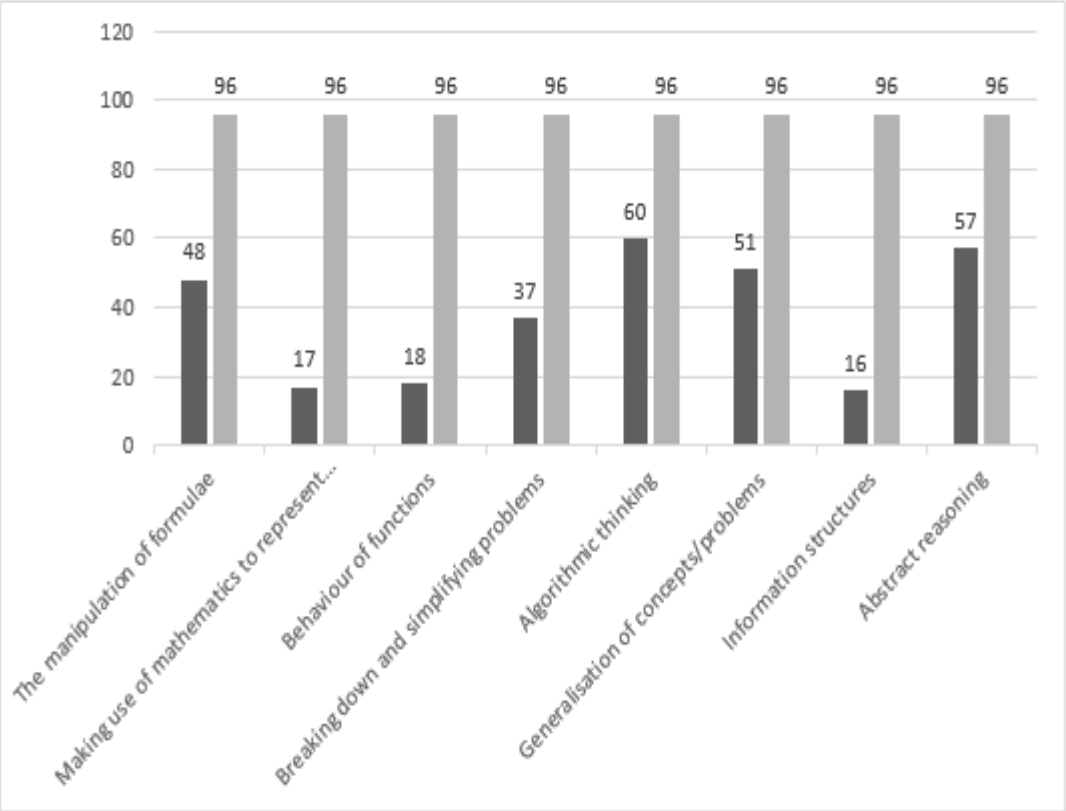



Figure 4.2: Analysis of 96 questions: Mathematical Thinking Skills

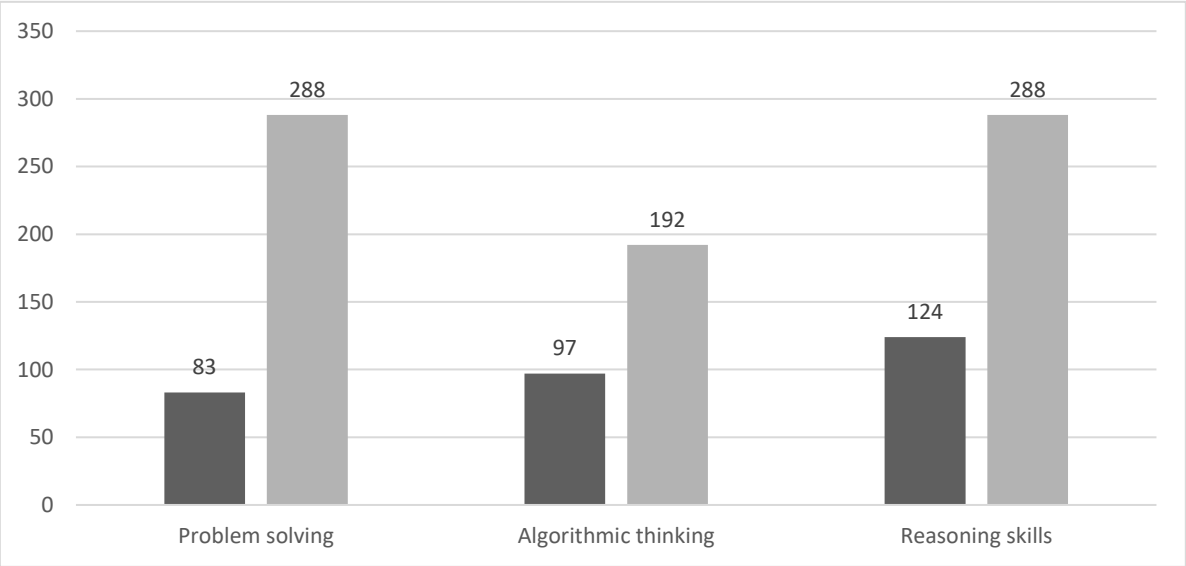



Figure 4.3: Analysis of 96 questions: Programming Thinking Skills



© University of Pretoria

Table 4.14: Summary of analysis of examination papers

| Mathematical Thinking Skills | Mathematical Thinking Skills | | | Programming Thinking Skills | | | Programming Thinking Skills |
|---|---|---|---|---|---|---|---|
| | TOTALS | POSSIBLE | PERCENTAGE | TOTALS | POSSIBLE | PERCENTAGE | |
| The manipulation of formulae | 48 | 96 | 50.00 | 83 | 288 | 28.82 | Problem solving |
| Making use of mathematics to represent realities | 17 | 96 | 17.71 | | | | |
| Behaviour of functions | 18 | 96 | 18.75 | | | | |
| Breaking down and simplifying problems | 37 | 96 | 38.54 | 97 | 192 | 50.52 | Algorithmic thinking |
| Algorithmic thinking | 60 | 96 | 62.50 | | | | |
| Generalisation of concepts/problems | 51 | 96 | 53.13 | 124 | 288 | 43.06 | Reasoning skills |
| Information structures | 16 | 96 | 16.67 | | | | |
| Abstract reasoning | 57 | 96 | 59.38 | | | | |

## 4.3 Analysis of Semi-structured Interviews

Semi-structured interviews were conducted with eight participants (P1 to P8) consisting of two examiners, two moderators and four lecturers. The lecturers who were interviewed are not currently lecturing first-year programming subjects, but were appointed in this position in the past (see Table 4.16). During the semi-structured interviews, seven questions with sub-questions were asked. The questions and sub-questions are listed in Table 4.15.



© University of Pretoria

Table 4.15: Interview questions

1. How long have you lectured first-year programming (DSO) at TUT?
2. Have you lectured first-year programming at any other institution? If yes, where and for how long?
3. Do the first years struggle with programming? If yes, why?
4. What mathematical thinking skills do <u>you</u> think first-year programming students need?
5. Does a lack of mathematical thinking skills limit their understanding of programming concepts? If yes, why?
6. What do <u>you</u> think is the most important aspect of mathematics that they need for first-year programming? In other words, knowledge of which content do they need?
7. Do you think the first-year programming students need to be able to:

a. Understand how to manipulate a formula? Why?
b. Make use of mathematics to represent realities? Why?
c. Understand the behaviour of functions? Why?
d. Use the concept of breaking down and simplifying problems? Why?
e. Use algorithmic thinking? Why?
f. Apply generalisation of concepts/problems? Why?
g. Understand and use information structures? Why?
h. Reason in an abstract way? Why?
i. Can you think of any other mathematical thinking skills your first-year programming students need?

Participants' responses to questions 1 to 6 are summarised and discussed below. Recordings and transcripts are available in digital format.

Table 4.16: Participants' roles in terms of first-year programming

| | |
|---|---|
| **P1** | Previously lectured first-year programming |
| **P2** | Examiner |



© University of Pretoria

| | |
|---|---|
| **P3** | Moderator |
| **P4** | Previously lectured first-year programming |
| **P5** | Moderator |
| **P6** | Examiner |
| **P7** | Previously lectured first-year programming, currently lecturing second- and third-year programming subjects |
| **P8** | Previously lectured first-year programming, currently lecturing second- and third-year programming subjects |

In order to determine the level of experience of the participants, the first question was "How long have you lectured first-year programming (DSO) at TUT?" The responses were as follows (see Table 4.17):

Table 4.17: Question 1: How long have you lectured first-year programming (DSO) at TUT?

| Participant | **P1** | **P2** | **P3** | **P4** | **P5** | **P6** | **P7** | **P8** |
|---|---|---|---|---|---|---|---|---|
| Number of years | 35 | 25 | 4 | 2 | 5 | 34 | 15 | 5 |

Only two of the participants (P1 and P8) have lectured first-year programming at other institutions, both for only one year (Question 2).

All the participants indicated that first-year students struggle with programming (Question 3). The reasons given by the participants clearly indicate that first-year students lack problem-solving skills (Table 4.18).

Table 4.18: Question 3: Do the first years struggle with programming? If yes, why?

| | |
|---|---|
| **P1** | They think it is difficult. They don't practise enough.<br><br>Lack of aptitude and logical thinking skills. |



© University of Pretoria

| | |
|---|---|
| **P2** | They don't have proper math skills and they were not taught to think in school. |
| **P3** | They struggle a lot. |
| **P4** | They are usually first-time computer users and learning the programming language itself is difficult and getting used to the coding is not easy |
| **P5** | They have to deal with a new environment and some come from disadvantaged backgrounds and have never used computers before. They have hardly ever clicked or touched a mouse and now they have to learn C++ or Visual Basic. It takes them longer and it adds to the stress, it is the computer, it is the language, it is the new place, and everything is just new. It takes its toll on the student. |
| **P6** | Based on the answers they give (in examinations and tests) it is clear that they don't know what they are doing, the programming and solving problems. |
| **P7** | It is their first exposure to programming and they cannot do problem-solving. |
| **P8** | They have problems with thinking and problem-solving. If they have been exposed to many examples of similar problems, they are able to re-produce the solutions. They have problems with creative problem solving, solving new, unfamiliar problems. |

Three of the participants indicated that problem-solving is one of the main reasons why first-year students struggle with programming. Other reasons that can be explored in future studies were also mentioned. These reasons include the lack of aptitude, logical thinking and mathematical skills. Three of the participants (P4, P5 and P7) mentioned the fact that most of the students at TUT are first-time computer users which indicates a lack of computer literacy skills. Another factor worth mentioning is the lack of language (English) skills which is a barrier to learning to programme (P4 and P5).



© University of Pretoria

Question 4 (What mathematical thinking skills do you think first-year programming students need?) yielded an interesting range of answers. It could be suggested that some of the participants do not have enough mathematical content knowledge to fully comprehend the term mathematical thinking skills (Table 4.19). The participants might have confused the term mathematical thinking skills with specific mathematical content. Question 6 was intended to determine the mathematical content (What do you think is the most important aspect of mathematics that they need for first-year programming? In other words, knowledge of which content do they need?). The participants' responses to Question 6 are summarised in Table 4.21. All the participants indicated that a lack of mathematical thinking skills limit students' understanding of programming concepts (Question 5). The reasons given by the participants are summarised below in Table 4.20.

Table 4.19: Question 4: What mathematical thinking skills do you think first-year programming students need?

| | |
|---|---|
| **P1** | Problem-solving, to understand the problems, logical thinking. |
| **P2** | Graphs and problem-solving, word sums. |
| **P3** | They must be able to analyse numbers, mathematical basics like mathematical operators and how to apply them in real-life. |
| **P4** | Solving equations, writing equations. Proportions, they really struggle with these. |
| **P5** | "Stuff" from school, statistics, grade 12 content, financial concepts like discount and VAT. |
| **P6** | Basic arithmetic skills, sometimes they don't even know when to add or subtract, the basics, not fancy mathematics. |
| **P7** | This participant preferred to refer to content, rather than skills and then mentioned the following content: |



© University of Pretoria

| | |
|---|---|
| | • Cartesian planes, plotting coordinates, aids visualisation that could lead to better understanding of two-dimensional arrays<br>• Use of variables and creating equations.<br>• Array (matrix algebra).<br>• Ratios and fractions.<br>• Geometry, arguments to prove or disprove answers.<br>• Boolean algebra, logical “stuff”.<br>• Different shapes, deconstruct shapes into basic elements and visualisation, area and volume. |
| **P8** | At least matric (grade 12) level. They must know their formulae, for example to calculate the volume of a cube, if they don’t know a formula they must be able to find it |

Once again, the participants emphasised the importance of problem-solving. Their responses could indicate that they regard problem-solving as part of, and inseparable from, mathematical thinking skills. They further highlighted the necessity of being able to write and solve equations from programming problem statements, and specifically referred to word sums. The fact that the students struggle with this may also be linked to the response to Question 3, where one of the participants (P5) indicated that language is a barrier to first-year students because programming problems are text-rich.

Table 4.20: Question 5: Does a lack of mathematical thinking skills limit their understanding of programming concepts? If yes, why?

| | |
|---|---|
| **P1** | They don’t know the basics of problem-solving and maths teaches problem-solving. |
| **P2** | Logical thinking and problem-solving; without being able to solve a mathematical word sum you cannot solve a problem programmatically. |



© University of Pretoria

| | |
|---|---|
| **P3** | Programming related to mathematics. If you don't understand maths, it becomes a problem if they have to solve programming problems because most of the programming problems relate to maths. |
| **P4** | It goes hand-in-hand with mathematics, think of BODMAS (order of operations), or the If-statements when dealing with mathematical operators and the application of all rules |
| **P5** | A good background of mathematics will lead to better understanding of concepts. Sometimes they struggle more with the mathematics in the problem than with the problem itself. They have to deal with the programming difficulties on top of the mathematical difficulties; it adds to the problem. |
| **P6** | Students have problems with problem-solving. |
| **P7** | Language barriers, students must do "double-coding", they are not English home language speakers, they have to decode the English problem statements and then code the solution in a programming language. Other factors as well, holistic problem. |
| **P8** | Somehow inter-related, not directly, more mathematics knowledge lead to more understanding and better problem-solving. |

When asked if the lack of mathematical thinking skills limit students' understanding of programming concepts, five of the eight participants again mentioned the lack problem-solving skills as a stumbling block for first-year students when they learn to program. One of the participants (P7) also referred to the language barrier and mentioned the principle of "double-coding", meaning that students whose first language is not English, have to translate problem statements into English and then solve and code the problems in a programming language. The relationship between mathematics and problem-solving was mentioned again.



© University of Pretoria

Table 4.21: Question 6: What do you think is the most important aspect of mathematics that they need for first-year programming? In other words, knowledge of which content do they need?

| | |
|---|---|
| **P1** | Learning to solve problems accurately.<br><br>Algebra and general concepts.<br><br>Change the subject of the formula. |
| **P2** | Problem-solving.<br><br>Algebra.<br><br>Venn diagrams to teach AND, OR and NOT (logical operators in programming).<br><br>Trigonometry to improve problem solving. |
| **P3** | Logical thinking, mastering maths logic will help with programming for first years. |
| **P4** | BODMAS (order of operations) and equations as mentioned earlier. |
| **P5** | Basic number skills.<br><br>Grade 10 to 12 mathematics.<br><br>Working with formulae (for example the area of a circle), once again, struggling to translate the mathematical formula into C++ or Visual Basic. |
| **P6** | They cannot read, interpret and solve problems. They cannot even solve with fixed values, variables even more difficult. Example: Rental car, initial reading, end reading, tariff per kilometre, they do not know what to do, they add instead of subtract. Students lack the ability to understand similar problems. |



© University of Pretoria

| | |
|---|---|
| **P7** | This participant answered this question in Question 4 and the interviewer therefore regarded this question as redundant |
| **P8** | Re-emphasise problem-solving.<br><br>Solving text-based problems, translating English into mathematics and then applying it. |

Seven of the eight interviewees again referred to problem-solving when asked to identify the most important aspect of mathematics that students need for first-year programming. Two of the participants (P6 and P8) again mentioned language as being a barrier to problem-solving and programming. Working with formulae and knowledge and understanding of variables were also mentioned.

Question 7 was divided into sub-questions that were intended to confirm the mathematical thinking skills as identified in the conceptual framework and analysed in question papers. All the participants indicated that the mentioned mathematical thinking skills are essential for first-year programming students (Question 7). When asked why they think so, the participants gave brief answers that are summarised in Table 4.22.

Table 4.22: Question 7: Do you think the first-year programming students need to be able to:

| Sub-questions | Responses | |
|---|---|---|
| **Understand how to manipulate a formula?** | **P1** | They need it, they can't change the subject of a formula |
| | **P2** | Formulate a formula, in other words, use formulae in programming, change subject of formulae |
| | **P3** | That will also assist them to solve a part of the problem |



© University of Pretoria

| | | |
|---|---|---|
| | **P4** | Calculate the unknown, change the subject of formulae |
| | **P5** | Convert formulae into a programming language like C++, dealing with the formulae and the syntax of the programming language |
| | **P6** | To do calculations |
| | **P7** | To manipulate statements and rewrite statements in programming, important in calculations |
| | **P8** | Important to mention the difference between maths and programming when using the equal to sign (=), in programming it is an assignment statement, in mathematics it is used in an equation |
| **Make use of mathematics to represent realities?** | **P1** | They lack the skills, visualisation of problems. |
| | **P2** | Core of programming |
| | **P3** | Most of the problems you get are real-life problems, they will need the mathematical knowledge to solve it |
| | **P4** | Like simple real-life problems, for example an ATM application, balance after withdrawal |
| | **P5** | Basics of OOP (object-oriented programming), model the real world |
| | **P6** | Main skill they lack, they cannot link it to reality |
| | **P7** | Financial equations, making decisions, mathematics helps to create expressions that relate to the pseudo code they use |



© University of Pretoria

| | | |
|---|---|---|
| | **P8** | Classes (OOP) are resemblances of the real world, transferred thinking, differentiate between relevant and irrelevant information |
| **Understand the behaviour of functions?** | **P1** | Very important programming skill |
| | **P2** | Principle of programming |
| | **P3** | Part of programming is to create your own functions in programming so to create functions you must know how functions behave |
| | **P4** | Use of arguments and parameters in functions and procedures like variables in maths |
| | **P5** | Makes the concept of functions easier to understand in programming |
| | **P6** | Essential programming skills |
| | **P7** | Referred to functions in question 4, outcome of graphs (Cartesian plane) |
| | **P8** | Valuable in programming |
| **Use the concept of breaking down and simplifying problems?** | **P1** | Very important programming skill |
| | **P2** | Modular programming (functions and sub-procedures) |
| | **P3** | Problems must be broken down in order to work on a certain part at a time |
| | **P4** | Easier to understand, get rid of redundant information |
| | **P5** | Problems are in words, students must extract essential information from the problem statements |



© University of Pretoria

| | | |
|---|---|---|
| | **P6** | Modular programming, solve a problem and break down in smaller parts |
| | **P7** | Core process of programming |
| | **P8** | Exactly what is done in programming, huge problems are complex, easier to approach it in smaller units |
| **Use algorithmic thinking?** | **P1** | Very important programming skill, logical thinking |
| | **P2** | Sequence of steps |
| | **P3** | Will help them with procedures in order to solve a problem they have |
| | **P4** | Problems are solved step-by-step |
| | **P5** | It is all about problem solving, if you cannot do problem-solving you cannot program |
| | **P6** | Steps in sequence, sequence always important |
| | **P7** | Solve problems in programming and mathematics step-by-step, to deduce and get a specific answer |
| | **P8** | Event-driven, procedural programming and OOP, sequence is crucial |
| **Apply generalisation of concepts/problems?** | **P1** | Write programs that apply to different sets of data |
| | **P2** | Solve general problems, any input to produce the correct output |
| | **P3** | Only answered yes and didn't elaborate further |



© University of Pretoria

| | | |
|---|---|---|
| | **P4** | Make use of variables, developing their knowledge of variables and applying it to arrays and databases, they must be able to adjust and grow |
| | **P5** | Many solutions to the same problem |
| | **P6** | Variables, students struggle with the concept |
| | **P7** | Real-world solutions and using variables, generalise a problem or present a solution in an equation |
| | **P8** | Re-applying in other instances, simple variables with descriptive names, world of programming, it develops into structures and classes (complexity increases) |
| **Understand and use information structures?** | **P1** | Easier to apply (array) |
| | **P2** | Arrays |
| | **P3** | To group related items in arrays doing the same operations on them |
| | **P4** | Arrays |
| | **P5** | Arrays, visualisation of movement of elements of the arrays |
| | **P6** | More complex applications can only be solved using arrays, if they know how data is stored it helps them to visualise how data is stored |
| | **P7** | Sets, ranges, loops, triggers, changing values applied in programming |



© University of Pretoria

| | | |
|---|---|---|
| | **P8** | Extremely important, huge hurdle, arrays are abstract, they battle with more than one dimension, and students are lost. Emphasising it in the maths may help. |
| **Reason in an abstract way?** | **P1** | Logical thinking |
| | **P2** | They lack this skill, they cannot analyse or visualise and think abstractly |
| | **P3** | Logical thinking is one of the major parts of programming |
| | **P4** | To get a solution |
| | **P5** | Essential for OOP modelling of the real world |
| | **P6** | Students struggle to think in an abstract way |
| | **P7** | It is a wide term, and students must be able to construct classes of objects in OOP (not necessarily in first year) |
| | **P8** | Mostly part of 2nd and 3rd year programming subjects, mathematics boost abstract thinking |
| **Can you think of any other mathematical thinking skills your first year programming students need?** | **P1** | No |
| | **P2** | Venn diagrams, the mathematical skills taught in the foundation subject is of great help to the students and the foundation students do better in their first year. |
| | **P3** | No |
| | **P4** | No |
| | **P5** | No |



© University of Pretoria

| | | |
|---|---|---|
| | **P6** | The basics, percentages, even and odd numbers, sum, product and difference |
| | **P7** | Ranges, collections, technical drawings, ratios are applied when drawing 3D elements |
| | **P8** | Abstractions, modules, number systems like binary and other number systems, recursion, maybe too advance for first year students |

Due to the fact that the participants are programming lecturers and not mathematics lecturers, they could not always distinguish between mathematical thinking skills and mathematical content knowledge. It is therefore relevant to refer to the mathematical content that the participants mentioned when asked which mathematical thinking skills are required for first-year programming subjects. The participants identified the following list of possible content that needs to be addressed when teaching mathematics to first-year programming students:

- Basic understanding of numbers and number systems.
- Basic algebra that would enable students to understand how to use variables and create equations. Knowledge of equations is also useful when students must change the subject of a formula to perform calculations when programming.
- Cartesian plane and plotting coordinates could lead to better understanding of arrays due to its two-dimensional nature.
- Matrix algebra to develop students' insight into multi-dimensional array.
- Ratios, fractions and percentages, specifically for financial applications.
- The use of geometry to teach students to state and then prove their arguments.
- Logical operators are widely used in programming; if students are taught Boolean algebra, it might help them understand logical operators better.
- Characteristics of shapes and solids that can assist students to visualise the solution to problems when they have to calculate volume and area.
- They must know their formulae, for example to calculate the volume of a cube. If they don't know a formula they must be able to find it.



© University of Pretoria

- Probability as applied when using Venn diagrams.
- Set theory: students don't always understand ranges of values, and making use of number lines and interval notation might lead to better understanding of If-then-else and Case select statements in programming.

## 4.4 Summary

Four examination papers of the first year programming subject, Development Software I, were analysed making use of the "Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments". The analysis of the question papers yielded results that confirm that the identified mathematical thinking skills are essential for first year programming students. The necessity for all the mathematical thinking skills has been identified in both the examination papers of the B-module that were analysed. The only mathematical thinking skill that is not required for the A-module papers that were analysed is the use of information structures.

In summary, when programming on first-year level, all the identified mathematical thinking skills had to be applied and used in answering questions in the analysed examination papers. It is also true that all the skills are essential and no specific skill can be regarded as being more important than another.

Semi-structured interviews were conducted with eight participants from the ICT Faculty at TUT. The participants all have two year or more experience of lecturing first-year programming. All the participants indicated repeatedly that the lack of problem-solving skills is one of the main reasons why first-year students struggle with programming and they even regard problem-solving as part of, and inseparable from, mathematical thinking skills. Other reasons mentioned included the lack of aptitude, logical thinking and mathematical skills. In the context of TUT specifically, it was mentioned that it is necessary to take cognisance of the fact that most of the students at TUT are first-time computer users who lack basic computer literacy skills.

Due to the fact that the participants are programming lecturers and not mathematics lecturers, their responses focused on mathematical content knowledge rather than mathematical thinking skills. However, the results of the interviews and the assessments undoubtedly came to the same conclusion and therefore confirmed that



© University of Pretoria

first-year programming students do require a set of mathematical thinking skills when learning to programme.



© University of Pretoria

# Chapter 5: Conclusions and Implications

"Many studies have shown that students often face difficulty in applying programming concepts to design a program that solves a given task" (Arshad, 2009, p. 372).

## 5.1 Introduction

To conclude this study, a final summary and comments are provided in order to situate the study in current research. In doing so, contradictions and gaps in the literature as well as limitations of the study are addressed. Recommendations for future research have been identified and these are briefly discussed.

## 5.2 Summary of Chapters

In Chapter 1, the study was introduced and contextualised by means of the rationale, a problem statement and the purpose of the study. The aim of this study was to explore and describe the mathematical thinking skills that students require for a first level programming subject towards obtaining a National Diploma in ICT. The primary question (*Which mathematical thinking skills do first-year students, studying towards a National Diploma in ICT, need*?) and the secondary question (*Which mathematical thinking skills are related to programming thinking skills?*) were formulated to guide the study. The methodological considerations were presented and, finally, concepts were clarified and the contributions that the study hopes to make were stated.

In Chapter 2, an extensive literature review was presented. This entailed investigating the thinking skills used in mathematics and programming. After researching thinking skills in general, mathematical and programming skills were explored as these are described in existing literature. The overlaps and links between mathematical thinking and programming thinking were identified and discussed in order to develop the conceptual framework of the study.

In Chapter 3, the methodological considerations that were briefly discussed in Chapter 1 were expanded into the detailed research methodology which was followed in the study. After highlighting the relevant paradigmatic perspectives, the characteristics of the interpretive paradigm were identified and discussed. The chosen research design determined the sampling procedures for data collection which entailed document and



© University of Pretoria

content analysis as well as the selection of participants for the semi-structured interviews.

Making use of the conceptual framework developed in Chapter 2, an analytical matrix was developed for the analysis of programming assessments. This enabled the researcher to identify the particular mathematical thinking skills required to successfully complete the assessments (Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments). An interview schedule was compiled with seven questions that could be asked to the selected participants. The data analysis, interpretation, trustworthiness, credibility and inter-rater reliability of the data collection processes as well as ethical considerations were also discussed in Chapter 3.

In Chapter 4, the analytical matrix as developed from the conceptual framework was used to analyse the selected question papers. Examples of the analysis and results of each type of question were described and discussed. The planned semi-structured interviews were conducted with experienced first-year programming lecturers in an effort to corroborate the results of the analysis of question papers. Transcripts were compiled from the recorded interviews after which the results were summarised and discussed.

## 5.3 Discussion of the Research Questions

A primary research question, followed by a secondary research question was formulated to guide the study. The two research questions that guided this study are (comp. Cohen, Manion & Morrison, 2011):

- Primary Research Question: Which mathematical thinking skills do first-year students, studying towards a National Diploma in ICT, need?
- Secondary Research Question: Which mathematical thinking skills are related to programming thinking skills?

It is pertinent to indicate that the aim of this study was to explore and describe the mathematical thinking skills that students require for a first level programming subject towards obtaining a National Diploma in ICT. The primary research question was therefore formulated as guide to achieve this objective. Literature was consulted to



© University of Pretoria

identify and explore a set of mathematical thinking skills first-year programming students need. The mathematical thinking skills applicable to the present study were chosen from the seminal work of Knuth (1985) and adapted to suit first-year programming students. These mathematical thinking skills are:

- The manipulation of formulae;
- Making use of mathematics to represent realities;
- Behaviour of functions;
- Breaking down and simplifying problems;
- Algorithmic thinking;
- Generalisation of concepts/problems;
- Information structures;
- Abstract reasoning.

In an attempt to confirm or contradict the necessity of the selected mathematical thinking skills, question papers were analysed by applying the "Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments" and semi-structured interviews were conducted.

The four selected examination papers were analysed in Chapter 4 (paragraph 4.2) and the results were captured and summarised in MS Excel. These summarised results are also represented in Chapter 4 (Table 4.14). A total of 96 questions from the question papers were analysed . The number of times a mathematical thinking skill was required was tallied and the results were converted into percentages. The results were sorted from the highest to the lowest prevalence in the analysed question papers in Table 5.1. Four of the mathematical thinking skills were prevalent in 50% and more of the questions, with algorithmic thinking at 62.5%, which was the highest percentage. The percentage of questions that required abstract reasoning was 59.38%, generalisation of concepts/problems were 53.13% and the manipulation of formulae 50%. Breaking down and simplifying problems was required in 38.54% of the analysed questions. The necessity of using the other three mathematical thinking skills was lower than 20%. These include the behaviour of functions, making use of mathematics to represent realities, and information structures.



© University of Pretoria

Programming thinking skills were also identified and described in order to address the secondary research question (*Which mathematical thinking skills are related to programming thinking skills?*). The programming thinking skills that were identified are problem-solving, algorithmic thinking and reasoning skills. Syslo (2015) describes algorithmic thinking as the ability to solve a problem by means of an algorithm after which the algorithm is translated and implemented in a programming language. This definition is indicative of the close relationship between algorithmic thinking and problem-solving. The programming thinking skills were linked to the mathematical thinking skills as illustrated in Figure 5.1.

Table 5.1: Mathematical Thinking Skills: Ordered Results of the Analyses of Four Question Papers

| Mathematical Thinking Skills | Mathematical Thinking Skills | | |
|---|---|---|---|
| | TOTALS | POSSIBLE | PERCENTAGE |
| Algorithmic thinking | 60 | 96 | 62.50 |
| Abstract reasoning | 57 | 96 | 59.38 |
| Generalisation of concepts/problems | 51 | 96 | 53.13 |
| The manipulation of formulae | 48 | 96 | 50.00 |
| Breaking down and simplifying problems | 37 | 96 | 38.54 |
| Behaviour of functions | 18 | 96 | 18.75 |
| Making use of mathematics to represent realities | 17 | 96 | 17.71 |
| Information structures | 16 | 96 | 16.67 |



© University of Pretoria

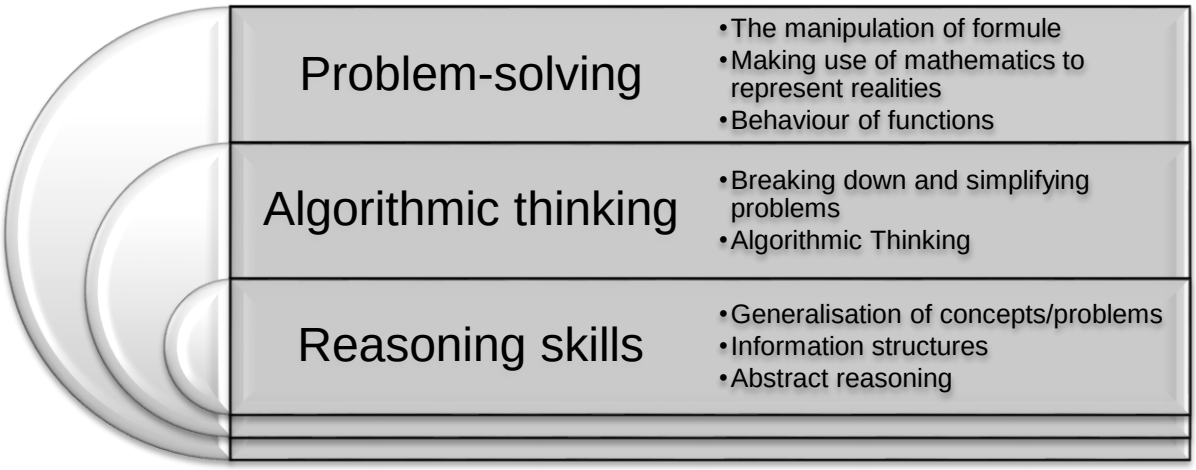



Figure 5.1: Links between Programming Thinking Skills and Mathematical Thinking Skills

When the question papers were analysed in terms of programming thinking skills, algorithmic thinking was required in 50.52% of the questions, reasoning skills in 43.06% and problem-solving in 28.82%. The links between the mathematical thinking and programming thinking skills, after the skills have been ordered from the highest to the lowest percentages, are illustrated in Table 5.2.

The results indicate a strong relationship between mathematical thinking skills and programming thinking skills. Although the analyses of the question papers did not indicate problem-solving as the most important programming thinking skill, several authors suggest that mathematical thinking is embedded in, and inherent to programming due to the fact that they regard problem-solving as the core of programming (Arshad, 2009; Byrne & Lyons, 2001; Gomes & Mendes, 2007; Rogerson & Scott, 2010). The ability to solve problems in mathematics therefore contributes towards students' success in programming because it involves problem-solving as a way of thinking (Beaubouef, Lucas & Howatt, 2001; Kinnunen, McCartney, Murphy & Thomas, 2007; Wilson & Shrock, 2001). The importance of problem-solving was further confirmed during the semi-structured interviews when the interviewees



© University of Pretoria

emphasised the importance of problem-solving as a core skill required for programming thinking.

The semi-structured interviews were conducted with eight participants consisting of two examiners, two moderators and four lecturers. During these interviews, seven questions with sub-questions were asked. The questions and sub-questions are listed in Chapter 4 (Table 4.15). When interviewed, seven of the eight interviewees referred to problem-solving as the most important aspect of mathematics that students need for first year programming. The inability to perform problem-solving has been indicated as one of the key reasons why students fail programming courses (Beaubouef & Mason, 2005; McCracken *et al*, 2001).

Table 5.2: Ordered Results of Analyses of Question Papers: Linking Mathematical Thinking Skills with Programming Thinking Skills

| Mathematical Thinking Skills | Mathematical Thinking Skills | | | Programming Thinking Skills | | | Programming Thinking Skills |
|---|---|---|---|---|---|---|---|
| | TOTALS | POSSIBLE | PERCENTAGE | TOTALS | POSSIBLE | PERCENTAGE | |
| Algorithmic thinking | 60 | 96 | 62.50 | 97 | 192 | 50.52 | Algorithmic thinking |
| Abstract reasoning | 57 | 96 | 59.38 | | | | |
| Generalisation of concepts/problems | 51 | 96 | 53.13 | 124 | 288 | 43.06 | Reasoning skills |
| The manipulation of formulae | 48 | 96 | 50.00 | | | | |
| Breaking down and simplifying problems | 37 | 96 | 38.54 | | | | |
| Behaviour of functions | 18 | 96 | 18.75 | 83 | 288 | 28.82 | Problem solving |
| Making use of mathematics to represent realities | 17 | 96 | 17.71 | | | | |
| Information structures | 16 | 96 | 16.67 | | | | |



© University of Pretoria

When asked if the lack of mathematical thinking skills limit students' understanding of programming concepts, five of the eight interviewees reiterated that the lack of (mathematical) problem-solving skills is a serious stumbling block for first-year students when they learn to programme. The interviewees specifically mentioned the students' inability to solve basic (mathematical) word problems and equations. Although solving word problems is a basic mathematical skill, it is crucial for programming because problems are almost always presented in words (Beaubouef & Mason, 2005). The interviewees therefore indicated that language proficiency might also be a relevant skill for programming.

## 5.4 Concluding Remarks

> "Mathematics is one of the disciplines that helped give birth to computer science [programming]" (Henderson, Barker, Epp, & Marion, 2003, p. 236).

The literature as well as the results of the analysed data indicated and confirmed that mathematical thinking skills are extremely important when learning to program. However, learning to program can also enhance and improve students' abilities to perform problem solving in mathematics (Feurzeig, Papert, & Lawler, 2011). Decker and Ventura (2004) found that students who took discrete mathematics and computer science together benefited and fared better in discrete mathematics than students not taking a computer science course. This strong relationship between programming and mathematics needs to be emphasised and applied when teaching programming to first-year students in order to improve their chances to succeed in their ICT studies.

## 5.5 Limitations of the study

Data was only collected from only one institution, TUT, and due to the unique profile of the students at TUT, one must be cautious not to generalise the findings. Students at TUT are mainly from rural, disadvantaged areas and first-generation to enter higher education. This may have influenced the interviewees' views and opinions as well as the format of the analysed question papers. It might be necessary to test the matrix and conduct the interviews at another university that offers the National Diploma in ICT; this also presents itself as an opportunity for future research.



© University of Pretoria

## 5.6 Recommendations for future research

In a recent study by Estey and Coady (2016), it is indicated that the trend is that a third of computer science [programming] students fail their courses. Their findings were previously confirmed by Bennedsen and Caspersen (2007) as well as Watson and Li (2014). The attrition rate at TUT is even higher and research in this regard should be conducted, after which different pedagogical approaches should be considered to (at least) increase students' levels of motivation and involvement that could, in turn, improve the quality of their learning experiences and therefore their chances to succeed (Mendes, Paquete, Cardoso & Gomes, 2012).

As anticipated, additional skills required for first-year programming have been derived from the results of the semi-structured interviews. The results of these interviews indicated that students whose first language is not English have to translate problem statements into English and then solve and code the problems in a programming language, resulting in the principle of "double-coding". First-year students at TUT have serious barriers to learning due to the fact that few of them are English mother-tongue speakers; for most of them English is a third or fourth language. Interviewees highlighted the necessity of being able to write and solve equations from programming problem statements; they specifically referred to the overlap with word sums in mathematics. The fact that the students struggle with this may also be contributed to the fact that language is a barrier to first-year students at TUT and programming problems are text-rich. Future research can focus on the challenges of non-English speaking students studying ICT (programming) at a university that only offers lectures in English.

Research focusing on best practice and teaching strategies that can be used to teach programming to first generation (higher education) students, needs to be conducted. These teaching strategies must include the application of the set of identified mathematical thinking skills as well the enhancement thereof to improve students' prospects to succeed in programming.



© University of Pretoria

## 5.7 Implications

"Learning to program is known to be a difficult task, as the students … find the new skills required difficult to master" (Kurvinen *et al*, 2016, p 308).

The outcome of this study is a set of mathematical thinking skills that needs to be developed when compiling a mathematics curriculum for first-year programming students studying towards a National Diploma in ICT. The development of such a curriculum is outside the scope of this study, but should be considered as a priority. It is pertinent that first-year programming lecturers need to be made aware of the results and implications of this study in order to apply these when teaching.

A course based on the (to be) developed curriculum, and tailored for first-year programming students, must be designed to develop the students' problem-solving skills while learning to program. First-year students in general need extensive academic support to adapt to higher education and designing this, or any other, course must include provision for support (Awang, Kutty & Ahmad, 2014; Grutzik & Ramos; 2016). Delivery methods and teaching strategies must be carefully considered and adapted to the needs of the students and the resources available on their campuses.

The latest trend is to combine face-to-face lectures with online learning (e-learning and/or mobile learning), creating a blended learning environment. Utilising blended learning effectively could result in "high-quality educational opportunities to meet the needs of students with a wide spectrum of academic needs and history" (Powell, Roberts & Patrick, p 5, 2015). It has also been found that blended learning is effective in supporting students "who fail one or more courses [and] become disengaged" (Powell, Roberts & Patrick, p 5, 2015).

## 5.8 Reflections

"Mathematics is an integral part of computer science and because of this, students must develop a good understanding of mathematics in order to be successful in computer science" (Beaubouef & Mason, 2005, p. 104).

Lecturing students in the ICT Foundation course at TUT (from 2011) gave me new insights in terms of the way we taught (lectured) programming in the past. Students



© University of Pretoria

entering the ICT Foundation course have many knowledge and skills gaps that need to be addressed while already enrolled for the National Diploma in ICT. The students need to “catch up” while learning a range of new skills needed for tertiary, and specifically ICT, studies. The subject ICT Foundation Mathematical Skills, offered in the first semester, had to be adapted in order to address the serious lack of basic mathematical skills and knowledge of the students while developing their problem-solving skills in preparation for the introduction to programming subject they follow in their second semester. This led to the need to identify the specific mathematical thinking skills that would prepare them sufficiently for programming subjects.

These results can now be used to improve the content of the mathematics and programming courses as well as the strategies followed when lecturing these subjects, specifically to the first-year and foundation students.



© University of Pretoria

## References

Allan, V. H., & Kolesar, M. V. (1996). Teaching Computer Science: A Problem Solving Approach that Works. In *Proceedings of the Annual National Educational Computing Conference,* pp. 9-21, Minneapolis, Minnesota, 11-13 June.

Arshad, N. (2009). Teaching Programming and Problem Solving to CS2 Students using Think-alouds. *Association of Computing Machinery (ACM) SIGCSE Bulletin*, *41*(1), 372-376.

Assaf, M. A. (2009). Teaching and Thinking: A Literature Review of the Teaching of Thinking Skills. . Online Submission. URL: http://eric.ed.gov/?id=ED505029

Aviles, C. B. (2000). Teaching and Testing for Critical Thinking with Bloom's Taxonomy of Educational Objectives. URL: http://eric.ed.gov/?id=ED446023

Awang, M. M., Kutty, F. M., & Ahmad, A. R. (2014). Perceived Social Support and Well-being: First-year Student Experience in University. *International Education Studies*, *7*(13), 261.

Babbie, E. R., & Mouton, J. (2012). *The Practice of Social Research*. Cape Town, South Africa: Oxford University Press.

Bales, B. L. (2002). Paradigm Privilege: Determining the Value of Research in Teacher Education Policy Making. URL: http://eric.ed.gov/?id=ED463287

Ball, D.L., Lubienski, S.T., & Mewborn, D.S. (2001). Research on Teaching Mathematics: The Unsolved Problem of Teachers' Mathematical Knowledge (4th edition). New York: MacMillan.

Barnes, D. J., Fincher, S., & Thompson, S. (1997, August). Introductory Problem Solving in Computer Science. In *5th Annual Conference on the Teaching of Computing,* Dublin City University (pp. 36-39).

Barron, I. (2013). The Potential and Challenges of Critical Realist Ethnography. *International Journal of Research and Method in Education, 36(2)*, 117- 130. doi: http://cx.doi.org/10.1080/174727X.2012.683634.



© University of Pretoria

Beaubouef, T., Lucas, R., & Howatt, J. (2001). The UNLOCK system: Enhancing Problem Solving Skills in CS-1 Students. *ACM SIGCSE Bulletin*, *33*(2), 43-46.

Beaubouef, T., & Mason, J. (2005). Why the High Attrition Rate for Computer Science Students: Some Thoughts and Observations. *ACM SIGCSE Bulletin*, *37*(2), 103-106.

Becker, J. R., & Rivera, F. (2005, July). Generalization Strategies of Beginning High School Algebra Students. In Proceedings of the 29th Conference of the International Group for the Psychology of Mathematics Education (PME) (Vol. 4, pp. 121-128). Melbourne: PME.

Bennedsen, J., & Caspersen, M. E. (2007). Failure Rates in Introductory Programming. *ACM SIGCSE Bulletin*, *39*(2), 32-36.

Bergen, S., & Reilley, R. (2005). Programming: Factors that Influence Success. *SIGSE Bulletin, 37(1)*, 411-415.

Beyer, B. K. (1984). Improving Thinking skills: Defining the Problem. *Phi Delta Kappan*, 486-490.

Bruce, K.B., Drysdale, R.L.S., Keleman, C., & Tucker, A. (2003). Why Math? *Communications of the ACM, 46(9)*, 40 – 44.

Byrne, P., & Lyons, G. (2001, June). The Effect of Student Attributes on Success in Programming. In *ACM SIGCSE Bulletin* (Vol. 33, No. 3, pp. 49-52).

Canast, J.J., Bajo, M.T., & Gonzalvo, P. (1994). Mental Models and Programming. *International Journal of Human-Computer Studies, 40*, 795-811.

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

Cerbin, B. (1988). The Nature and Development of Informal Reasoning Skills in College Students. Paper presented at the *12th National Institute on Issues in Teaching and Learning Conference*, Chicago, April 1988.



© University of Pretoria

Coffman, D. M. (2013). Thinking about Thinking: An Exploration of Pre-service Teachers' Views about Higher Order Thinking Skills. Submitted to the graduate degree program in Curriculum and Instruction and the Graduate Faculty of the University of Kansas in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

Cohen, L., Manion, L., & Morrison, K. (2011). *Research Methods in Education – Seventh Edition*. New York, USA: Routledge, Taylor & Francis Group.

Corney, M., Teague, D., & Thomas, R. N. (2010, January). Engaging Students in Programming. In *Proceedings of the Twelfth Australasian Conference on Computing Education-Volume 103* (pp. 63-72). Australian Computer Society, Inc.

Creswell, J.W., Hanson, W.E., Plano Clark, V.L., & Morales, A. (2007). Qualitative Research Designs: Selection and Implementation. *The Counselling Psychologist, 35(2)*, 236-264. doi: 10.1177/0011000006287390.

Croft, A., Croft, T., & Davison, R. (2008). *Mathematics for Engineers: A Modern Interactive Approach*. Pearson Education.

Crouch, R., & Haines, C. (2004). Mathematical Modelling: Transitions between the real world and the mathematical model. *International Journal of Mathematical Education in Science and Technology*, *35*(2), 197-206.

Cuoco, A.L., Goldenberg, E.P., & Mark, J. (1996). Habits of mind: An organizing principle for mathematics curriculum. *Journal of Mathematical Behavior*, 15(4), 375-402.

Daly, W. T. (1995). *Beyond critical thinking: Teaching the thinking skills necessary to academic and professional success* (No. 17). National Resource Center for the Freshman Year Experience & Students in Transition.

Department of Basic Education (DBE). (2014). Report on Annual National Assessment of 2014. Pretoria, South Africa: Department of Basic Education.



© University of Pretoria

Department of Higher Education and Training (DHET). (2013). White Paper for post-school education and training, building an expanded, effective and integrated post-school system. Pretoria, South Africa: Department of Higher Education.

Decker, A., & Ventura, P. (2004, March). We claim this class for computer science: A non-mathematician's discrete structures course. In *ACM SIGCSE Bulletin* (Vol. 36, No. 1, pp. 442-446). ACM.

Dehnadi, S., & Bornat, R. (2006). The camel has two humps (working title). Middlesex University, UK, pp 1-21. *URL http://www.cs.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf*.

Denzin, K.D., & Lincoln, Y.S. (Eds.). (2005). *The SAGE Handbook of Qualitative Research – Third Edition*. Thousand Oaks: Sage Publications, Inc.

Eckerdal, A., & Berglund, A. (2005). *What does it take to learn 'Programming Thinking'?* Paper presented at the Proceedings of the First International Workshop on Computing Education Research, Seattle, WA, USA.

Edwards, B. S., Dubinsky, E., & McDonald, M. A. (2005). Advanced mathematical thinking. *Mathematical Thinking and Learning*, *7*(1), 15-25.

Engelbrecht, J. (2010). Adding structure to the transition process to advanced mathematical activity. *International Journal of Mathematical Education in Science and Technology*, *41*(2), 143-154.

Ennis, R.H. (1962). A concept of critical thinking. Harvard Educational Review, *32*(1), 1962, 81-111.

Ennis, R. H. (1989). Critical thinking and subject specificity: Clarification and needed research. *Educational researcher*, *18*(3), 4-10.

Ennis, R. H. (1990). The extent to which critical thinking is subject-specific: Further clarification. *Educational Researcher*, *19*(4), 13-16.

Epp, S.S., (1994). In Schoenfeld, A. H. (Ed.), *Mathematical thinking and problem solving* (p. 257-269). Hillsdale, New Jersey: Lawrence Erlbaum Associates, Inc.



© University of Pretoria

Estey, A., & Coady, Y. (2016, July). How can we improve student workflow practices to better enable student success in CS1? In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 370-370). ACM.

Farrell, J. (2002). Programming logic and design, Introductory (2nd ed.). Boston: Course Technology

Ferreira, R. (2012). Writing a research proposal, In Maree, K. (Ed.), *Complete Your Thesis or Dissertation Successfully: Practical Guidelines (*pp 29 -39). Cape Town: Juta and Company Ltd.

Feza-Piyose, N. (2012). Language: A capital for conceptualizing mathematics knowledge. *International Electronic Journal of Mathematics Education, 7(2)*, 62-79.

Feurzeig, W., Papert, S. A., & Lawler, B. (2011). Programming-languages as a conceptual framework for teaching mathematics. *Interactive Learning Environments*, *19*(5), 487-501.

Frith, V., & Prince, R. (2009). A framework for understanding the quantitative literacy demands of higher education, *South African Journal of Higher Education, 23*(1), 83-89.

Gal-Ezer, J., & Harel, D. (1998). What (else) should CS educators know? *Communications of the ACM*, *41*(9), 77-84.

Gal-Ezer, J., & Zeldes, A. (2000). Teaching software designing skills. *Computer Science Education*, *10*(1), 25-38.

Glevey, K. E. (2006). Promoting thinking skills in education. *London review of education*, *4*(3), 291-302.

Gomes, A., Bigotte, E., Carmo, L., & Mendes, A. (2006). *Mathematics and programming problem solving.* Paper presented at the 3rd E-Learning Conference, Coimbra, Portugal.



© University of Pretoria

Gomes, A., & Mendes, A. J. (2007, June). An environment to improve programming education. In *Proceedings of the 2007 International Conference on Computer Systems and Technologies*, Bulgaria.

Gomes, A.J., Santos, A., & Mendes, A.J. (2012). *A study on students' behaviourand attitudes towards learning to program*. A paper presented at the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education, Haifa, Israel.

Goold, A., & Rimmer, R. (2000). Factors affecting performance in first-year computing. *SIGSE Bulletin, 32(2)*, 39-43.

Gordon, M. (2011). Mathematical habits of mind: promoting students' thoughtful considerations, *Journal of Curriculum Studies, 43(4)*, 457-469.

Grandell, L., Peltomäki, M., Back, R. J., & Salakoski, T. (2006). Why complicate things? Introducing programming in high school using Python. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52* (pp. 71-80). Australian Computer Society, Inc.

Greeno, J.G., (1994). In Schoenfeld, A. H. (Ed.), *Mathematical thinking and problem solving* (p. 270-278). Hillsdale, New Jersey: Lawrence Erlbaum Associates, Inc.

Grutzik, C., & Ramos, S. (2016). The role of the student support specialist: The possibilities and challenges of a long-term, proactive, and scaffolded relationship. *Community College Journal of Research and Practice*, *40*(2), 113-132.

Gubbins, E. J. (1985). Matrix of thinking skills. *Unpublished document. Hartfort, CT: State Department of Education. Cited in Sternberg, Robert J. (1986). Critical thinking: Its nature, measurement, and improvement. ERIC Document Reproduction Service*, (272882), 9-26.

Gwo-Dong, C. H. E. N., Liang-Yi, L. I., & Chin-Yea, W. A. N. G. (2012). A community of practice approach to learning programming. *TOJET: The Turkish Online Journal of Educational Technology*, *11*(2).



© University of Pretoria

Halmos, P. (1980). The heart of mathematics. *American Mathematical Monthly*, 87, 519-524.

Harel, G., & Sowder, L. (2005). Advanced mathematical-thinking at any age: Its nature and its development. *Mathematical thinking and learning*, *7*(1), 27-50.

Heid, M. K. (1989). How symbolic mathematical systems could and should affect precollege mathematics. *The Mathematics Teacher*, *82*(6), 410-419.

Henderson, P.B. (2003). Mathematical reasoning in software engineering education. *Communications of the ACM, 46(9)*, 45-50

Henderson, P. B., Barker, W., Epp, S., & Marion, W. (2003, February). Math educators, computer science educators: working together. In *ACM SIGCSE Bulletin* (Vol. 35, No. 1, pp. 236-237). ACM.

Hourigan, M., & O'Donoghue, J. (2007). Mathematical under-preparedness: the influence of the pre-tertiary experience on students' ability to make a successful transition to tertiary level mathematics courses in Ireland. *International Journal of Mathematical Education in Science and Technology, 38(4)*, 461-476.

Hu, C. (2006). It's mathematical after all – the nature of computer programming. *Education and Information Technologies, 11(1)*, 83-92

Kafai, Y. B., & Burke, Q. (2015). Computer programming goes back to school. *Education Week*, 61-65.

Kantner, J. (2008). Adult learning and numeracy: Introduction. *Adult Learning, 19(3/4)*, 4-7.

Katagiri, S. (2004). Mathematical thinking and how to teach it. *CRICED, University of Tsukuba*.

Kilpatrick, J., Swafford, J., & Findell, B. (Editors). (2001). *Adding it up: Helping children learn mathematics*. Mathematics Learning Study Committee; Center for Education; Division of Behavioral and Social Sciences and Education; National Research Council. National Academies Press.



© University of Pretoria

Kinnunen, P., & Malmi, L. (2006, September). Why students drop out CS1 course? In *Proceedings of the Second International Workshop on Computing Education Research* (pp. 97-108). ACM.

Kinnunen, P., McCartney, R., Murphy, L., & Thomas, L. (2007, September). Through the eyes of instructors: A phenomenographic investigation of student success. In *Proceedings of the third international workshop on Computing Education Research* (pp. 61-72).

Klein, H. K., & Hirschheim, R. A. (1987). A comparative framework of data modelling paradigms and approaches. *The computer journal*, *30*(1), 8-15.

Knuth, D.E. (1985). Algorithmic thinking and mathematical thinking. *The American Mathematical Monthly, 92(3)*, 170-181.

Krathwohl, D. R. (2002). A revision of Bloom's taxonomy: An overview. *Theory into practice*, *41*(4), 212-218.

Kurland, D. M., Pea, R. D., Clement, C., & Mawby, R. (1986). A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research*, *2*(4), 429-458.

Kurvinen, E., Hellgren, N., Kaila, E., Laakso, M. J., & Salakoski, T. (2016, July). Programming misconceptions in an introductory level programming course exam. in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 308-313). ACM.

Kvale, S. (2011). *Doing interviews*. London: Sage.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H. M. (2005). A study of the difficulties of novice programmers. In *ACM SIGCSE Bulletin 37*(3), 14-18.

Landy, D., & Goldstone, R. L. (2007). How abstract is symbolic thought? *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *33*(4), 720.

Lannin, J. K. (2003). Developing algebraic reasoning through generalizaton. *Mathematics Teaching in the Middle School, 8*(7), 342-348. Retrieved from



© University of Pretoria

http://search.proquest.com.uplib.idm.oclc.org/docview/231172218?accountid=14717

Lippert, S. K., & Granger, M. J. (1997). Peer learning in an introductory programming course. In Proceedings of the International Academy for Information Management Annual Conference (12th, Atlanta, Georgia, December 12-14, 1997), 123 – 130.

Lloyd, P., & Frith, V. (2013). Proportional reasoning as a threshold to numeracy at university: a framework for analysis. *Pythagoras, 34*(2), 9.

Madison, S., & Gifford, J. (2002). Modular programming: Novice misconceptions. *Journal of Research on Technology in Education*, *34*(3), 217-229.

Malterud, K. (2001). Qualitative research: standards, challenges and guidelines. *The Lancet, 358,* 483-488.

Maree, K., & Van der Westhuizen, C. (2013). Planning a research. In Maree, K. (Ed.), *First Steps in Research (*pp 24 -45). Pretoria: Van Schaik's Publishers.

Marr, B. (2001). Connecting students, sense and symbols: a workshop of practical activities from personal experience, and informed by research. In *Proceedings of ALM-7, the 7th International Conference of Adults Learning Mathematics,* Medford, Massachusetts, USA, July 6-8,200, 216-219.

Marshall, J. (2003). Math wars: Taking sides. *Phi Delta Kappan*, *85*(3), 193.

Mayer, R. E. (2002). Rote versus meaningful learning. *Theory into practice*,*41*(4), 226-232.

Marzano, R. J., & Hutchins, C. L. (1985). *Thinking skills: A conceptual framework*. Mid-continent Regional Educational Laboratory.

Marzano, R. J. (1998). What are the general skills of thinking and reasoning and how do you teach them? *The Clearing House*, *71*(5), 268-273.



© University of Pretoria

McCracken, M., Alstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B.D., Laxer, C., Thomas, L., Utting, I., & Wilusz, T. (2001). A multi-national, multi-institutional study of research assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin, 33(4),* 125-180.

McPeck, J. E. (1981). *Critical thinking and education*. Martin Robertson & Company Ltd, Oxford.

McPeck, J. E. (1984). Stalking beasts, but swatting flies: The teaching of critical thinking. *Canadian Journal of Education/Revue canadienne de l'education*, 28-44.

McPeck, J. E. (1990). Critical thinking and subject specificity: A reply to Ennis. *Educational Researcher*, *19*(4), 10-12.

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.

Milne, I., & Rowe, G. (2002). Difficulties in learning and teaching programming—Views of students and tutors. *Education and Information technologies*, *7*(1), 55-66.

Mohamed Shuhidan, S., Hamilton, M., & D'Souza, D. (2011, June). Understanding novice programmer difficulties via guided learning. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* (pp. 213-217). ACM.

Moore, T. (2004). The critical thinking debate: how general are general thinking skills? *Higher Education Research & Development*, *23*(1), 3-18.

Nieuwenhuis, J. (2013). Qualitative research designs and data gathering techniques. In Maree, K. (Ed.), *First Steps in Research (*pp 70 -98). Pretoria: Van Schaik's Publishers.

Ninness, C., Barnes-Holmes, D., Rumph, R., McCuller, G., Ford, A. M., Payne, R., Ninness, S.K., Smith, R.J., Ward, T.A., & Elliott, M. P. (2006). Transformations



© University of Pretoria

of mathematical and stimulus functions. *Journal of Applied Behavior Analysis*, *39*(3), 299-321.

O'Donoghue, J. (2002). *Mathematics or numeracy: does it really matter?* Paper presented at 9th International Conference of Adults Learning Mathematics, 17-20 July 2002, Uxbridge, London.

Overbaugh, R. C. (1993). A BASIC programming curriculum for enhancing problem-solving ability. ERIC Number: ED355921

Özyurt, Ö. (2015). Examining the critical thinking dispositions and the problem solving skills of Computer Engineering Students. *Eurasia Journal of Mathematics, Science & Technology Education*, *11*(2), 353-361.

Parnell, S. & Statham, M. (2007). An effective preparation for tertiary mathematics. *International Journal of Mathematical Education in Science and Technology, 38(7)*, 869-879.

Polya, G. (1957). *How to solve it: a new aspect of mathematical method* (2nd ed.). Princeton, New Jersey: Princeton University Press.

Powell, A., Roberts, V., & Patrick, S. (2015). Using online learning for credit recovery: getting back on track to graduation. Promising Practices in Blended and Online Learning Series. *International Association for K-12 Online Learning*.

Pretorius, CM. & Erasmus H.G. (2013). *Basic programming principles*. Cape Town, South Africa: Pearson Education South Africa (Pty) Ltd.

Putnam, R.T. (1987). Mathematics knowledge for understanding and problem solving. *International Journal of Educational Research, 25*, 464-481. doi: 10.11770193945903253909

Rabionet, S. E. (2011). How I learned to design and conduct semi-structured interviews: An ongoing and continuous journey. *The Qualitative Report*, *16*(2), 563.



© University of Pretoria

Reddy, V., Prinsloo, C., Visser, M., Arends, F., Winnaar, L., & Rogers, S. (2012). *Highlights from TIMSS 2011: the South African perspective.* Pretoria: Human Sciences Research Council (HSRC).

Richard, P. (1985). Bloom's taxonomy and critical thinking instruction. *Educational Leadership*, *42*(8), 36-39.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education, 13(2),* 137-172.

Rogerson, C., & Scott, E. (2010). The fear factor: How it affects students learning to program in a tertiary environment. *Journal of Information Technology Education*, *9*(1), 147-171.

Rogalski, J., & Samurçay, R. (1990). Acquisition of programming knowledge and skills. *Psychology of programming*, *18*, 157-174.

Saldaña, J. (2013). *The coding manual for qualitative researchers*. London: Sage.

Scott, D. (2005). Critical realism and empirical research methods in education. *Journal of Philosophy of Education, 39(4)*, 633-646.

Schoenfeld, A.H. (1992). Learning to think mathematically: problem solving, metacognition, and sense-making in mathematics. In D. Grouws (Ed.), *Handbook for Research on Mathematics Teaching and Learning* (pp. 334-370). New York: MacMillan.

Schoenfeld, A. H. (Ed.). (1994). *Mathematical thinking and problem solving*. Hillsdale, New Jersey: Lawrence Erlbaum Associates, Inc.

Schrage, J. F. (1997). Six Thinking Aspects of Programming. In *Proceedings of the International Academy for the 12th Information Management Annual Conference,* pp. 242- 249, Atlanta, Georgia.

Seabi, J. (2012). Research designs and data collection techniques, In Maree, K. (Ed.), *Complete Your Thesis or Dissertation Successfully: Practical Guidelines (*pp 81 - 95). Cape Town: Juta and Company Ltd.



© University of Pretoria

Selden, A., & Selden, J. (2005). Perspectives on advanced mathematical thinking. *Mathematical Thinking and Learning*, *7*(1), 1-13.

Smith, G. (2002a). Are there domain specific thinking skills? *Journal of Philosophy of Education*, *36*(2), 207-227.

Smith, G. F. (2002b). Thinking skills: the question of generality. *Journal of curriculum studies*, *34*(6), 659-678.

Snyder, L., Aho, A. V., Linn, M., Packer, A., Tucker, A., Ullman, J., & Van Dam, A. (1999). Being fluent with information technology. *Computer Science and Telecommunications Board, National Research Council. National Academy Press, Washington, DC*.

Spaull, N. (2013). South Africa's education crisis: The quality of education in South Africa 1994-2011. Johannesburg: Centre for Development and Enterprise.

Stanic, G., & Kilpatrick, J. (1989). Historical perspectives on problem solving in the mathematics curriculum. In R. Charles & E. Silver (Eds.). *The teaching and assessing of mathematical problem solving* (pp. 1-22). Reston, VA: National Council of Teachers of Mathematics.

Steen, L. A. (1999). Twenty questions about mathematical reasoning. In *Developing mathematical reasoning in grades K-12. National Council of Teachers of Mathematics, NCTM's*, 270-285.

Steen, L.A. (1999). Numeracy: the new literacy for a data-drenched society. *Educational Leadership, 57(2)*, 8.

Steen, L.A. (2003). Math education at risk. *Issues in Science and Technology* 19, no. 4 (Summer 2003), Retrieved April 2016 from http://issues.org/19-4/steen/.

Sysło, M. M. (2015, June). From algorithmic to computational thinking: on the way for computing for all students. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 1-1). ACM.



© University of Pretoria

Tall, D. (1995). Cognitive growth in elementary and advanced mathematical thinking. In *L. Miera and D. Carraher (Eds.), Proceedings of the 19th International Conference for the Psychology of Mathematics Education* (Vol. 1, pp 61-75). Recife, Brazil: Universidade Federal de Pernambuco.

Tall, D. (2008). The transition to formal thinking in mathematics. *Mathematics Education Research Journal, 20(2)*, 5-24.

Tall, D. (2013). *How humans learn to think mathematically: Exploring the three worlds of mathematics*. Cambridge University Press.

Tan, P. H., Ting, C. Y., & Ling, S. W. (2009, November). Learning difficulties in programming courses: undergraduates' perspective and perception. In *Computer Technology and Development, 2009. ICCTD'09. International Conference on, Vol 1*, 42-46. IEEE.

Titus, F. (2010). *A cognitive analysis of developmental mathematics students' errors and misconceptions in real number computations and evaluating algebraic expressions* (Order No. 3425036). Available From ProQuest Dissertations & Theses Global: Social Sciences. (756894992). Retrieved from http://search.proquest.com.uplib.idm.oclc.org/docview/756894992?accountid=14717

Vos, K. E. (1978). Learning style and plausible reasoning. Retrieved from http://search.proquest.com.uplib.idm.oclc.org/docview/63835943?accountid=14717

Vrchota, D. (2004). Touchstone award: challenging students' thinking with Bloom's Taxonomy. *Communication Teacher*, *18*(1), 2-5.

Wagner, C., Kawulich, B., & Garner, M. (2012). *Doing social research – a global context. Berkshire*, UK: McGraw-Hill Higher Education.

Waisman, I., Leikin, M., Shaul, S., & Leikin, R. (2014). Brain activity associated with translation between graphical and symbolic representations of functions in



© University of Pretoria

generally gifted and excelling in mathematics adolescents. *International Journal of Science and Mathematics Education*, *12*(3), 669-696.

Watson, A. (2010). Shifts of mathematical thinking in adolescence. *Research in Mathematics Education, 12(2),* 133-148.

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). ACM.

Wileman, S., Konvalina, J., & Stephens, L.J. (1981). Factors influencing success in beginning computer science courses. *The Journal of Educational Research, 74(4)*, 223 – 226.

Willoughby, S.S. (2014). Textbooks, testing, training: how we discourage thinking. Washington, DC: Mathematical Association of America.

Wilson, B.C., & Shrock, S. (2001). Contributing to success in an introductory computer science course: a study of twelve factors. *SIGCSE Bulletin, 33(1)*, 184-188

Wilson, M., Scalise, K., & Gochyyev, P. (2015). Rethinking ICT literacy: From computer skills to social network settings. *Thinking Skills and Creativity*, *18*, 65-80.

Woodley, M., & Kamin, S. N. (2007, March). Programming studio: a course for improving programming skills in undergraduates. In *ACM SIGCSE Bulletin 39(1)*, 531-535.

Zazkis, R., & Applebaum, M. (2007). Advancing mathematical thinking: Looking back to one problem. *Working Group 14: Advanced mathematical thinking, Proceedings of the Fifth Congress of the European Society for Research in Mathematics Education (CERME-5),* 2389-2397. Lamaca, Cyprus. URL: http://www.mathematik.uni-dortmund.de/~erme/CERME5b/

Zsakó, L., & Szlávi, P. (2012). ICT competences: algorithmic thinking. *Acta Didactica Napocensia*, *5*(2), 49.



© University of Pretoria

Zylka, J., Christoph, G., Kroehne, U., Hartig, J., & Goldhammer, F. (2015). Moving beyond cognitive elements of ICT literacy: First evidence on the structure of ICT engagement. *Computers in Human Behavior*, *53*, 149-160.



© University of Pretoria

## Appendix A: Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments

| Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Mathematical Thinking Skills** | **Questions** | | | | | | | | | | | | | | | **Programming Thinking** |
| | | | | | | | | | | | | | | | | |
| The manipulation of formulae | | | | | | | | | | | | | | | | Problem solving skills |
| Making use of mathematics to represent realities | | | | | | | | | | | | | | | | |
| Behaviour of functions | | | | | | | | | | | | | | | | |
| Breaking down and simplifying problems | | | | | | | | | | | | | | | | Algorithmic thinking |
| Algorithmic thinking | | | | | | | | | | | | | | | | |
| Generalisation of concepts/problems | | | | | | | | | | | | | | | | Reasoning skills |
| Information structures | | | | | | | | | | | | | | | | |
| Abstract reasoning | | | | | | | | | | | | | | | | |



© University of Pretoria

## Appendix B: Solution of Question 4 of Paper 4

```
DoSalesCalculations

~       This program enter sales in a week for a medical supply company, and calculates the commission earned by each salesperson

        call InitializeArrayValues() ✓✓

        display "Enter the salesperson number who made the sale (-1 to end)" ½

        enter salesPNum ½

        do while salesPNum <> -1 ✓

                display "Enter the sales amount "

                enter salesAmnt ½

                do ✓

                        subscr = FindSalesPNum (salesPNum) ✓✓

                        if subscr = -1 then ½

                                display "An invalid salesperson number was entered – Please re-enter)" ½

                                enter salesPNum ½

                        endif

                until subscr >= 0 ✓

                weeklySales(subscr) = weeklySales(subscr) + salesAmnt✓

                display "Enter the salesperson number who made the sale (-1 to end)"

                enter salesPNum ½

        loop ½

        call CalcCommission()✓
```



© University of Pretoria

```
      for sp = 0 to 6 ½
            display salesPersName(sp), " earned R", commEarned(sp), " in commission" ½
      next sp ½
      call DetermineHighestSales(highestSalesPName, highestSales) ✓✓
      display highestSalesPName, " had the highest total sales, with R"  highestSales ½
end                                                                      (16)
sub InitializeArrayValues()✓
      for sp = 0 to 6 ½
            weeklySales(sp) = 0 ½
            commEarned(sp) = 0 ½
      next sp
end sub ½                                                                (3)

function FindSalesPNum (valSalesPNum) ✓
      sp = 0 ½
      found = false ½
      while sp <= 6 ½ AND found = false ½
            if salesPersNum(sp) = valSalesPNum then ½
                  found = true ½
                  subscr = sp ½
            endif
```



© University of Pretoria

```
        loop
        if found = false then ½
                subscr = -1 ½
        endif
return subscr ½                                                    (6)


sub CalcCommission()✓
        for sp = 0 to 6 ½
                if yrsExperience(sp) >= 0 then ½
                        if yrsExperience(sp) <= 5 then ½
                                if weeklySales(sp) <= 100000 then ½
                                        commEarned(sp)=0.03*weeklySales(sp) ½
                                else
                                        commEarned(sp)=0.03*100000 + 0.05*(weeklySales(sp)-100000) ½
                                endif
                        else
                                if yrsExperience(sp) <= 10 then ½
                                        if weeklySales(sp) <= 100000 then
                                           commEarned(sp)=0.075*weeklySales(sp) ½
                                        else
                                           commEarned(sp)=0.075*100000 + 0.1*(weeklySales(sp)-400000) ½
                                        endif
```



© University of Pretoria

```
                        else
                              if weeklySales(sp) <= 100000 then
                                    commEarned(sp)=0.12*weeklySales(sp) ½
                              else
                                    commEarned(sp)=0.12*100000 + 0.15*(weeklySales(sp)-100000) ½
                              endif
                        endif
                  endif
            endif
      next sp
end sub ½                                                                    (7)
call DetermineHighestSales(highestSalesPName, highestSales) ✓✓
      highestSalesPName = salesPersName(0) ½
      highestSales = weeklySales(0) ½
      for sp = 1 to 6 ½
            if weeklySales(sp) > highestSales then ✓
                  highestSales = weeklySales(sp) ½
                  highestSalesPName = salesPersName(sp) ½
            endif
      next sp
end sub ½                                                                    (6)
```



© University of Pretoria

## Appendix C: Solution of Question 5 of Paper 4

### C++ Solution for Question 5

```
#include <iostream>
#include <string>
#include <iomanip>
#include<conio.h>

using namespace std;

const int ARRAYSIZE = 10;//0.5 Mark

int accNum[] = {813465, 548926 , 662908, 713890, 412579 ,916356, 528349, 920345, 710293, 820394}; //2 Marks
string accName[] ={"T Mahlangu","R Pretorius", "K More", "F Knauff",
                          "V Raphiri", "H Raseroka", "M Malebane", "Z Kgoete", "S Meintjes", "T Radebe"}; //2 Marks
double accTotDue[] = {1200.0, 1570.0, 5600.0, 5200.0, 3200.0, 6600.0, 13000.0, 1500.0, 4800.0, 2000.0}; //2 Marks
int accRemTerm[] {4, 5, 4, 6, 2, 10, 1, 3, 10, 4}; //1 Mark
int accIntRate[] = {9, 12, 7, 10, 6, 15, 11, 8, 13, 9}; //1 Mark
double accMonthlyDue[ARRAYSIZE]={0.0};//1 Mark

void updateTotalDue(); //0.5 Mark
double calcIntPayable (int, double);//0.5 Mark
void displayBillingReport();//0.5 Mark

int main()
{

       int acc;
       double interestPayable;

       updateTotalDue();//1 Mark

       for (acc = 0; acc < ARRAYSIZE; acc++)//1 Mark
       {
              interestPayable =  calcIntPayable(accIntRate[acc], accTotDue[acc]); //2 Marks
              accMonthlyDue[acc] = accMonthlyDue[acc] - interestPayable ;//2 Mark
       }
       displayBillingReport();//1 Mark
```



© University of Pretoria

```
        _getch();
        return 0;
}

void updateTotalDue()//1 Mark
{
        int num, acc;
        double amnt;

        cout << "\nEnter the account number that you would like \nto update the total amount due for (-1 to end): ";
        cin >> num; //0.5 Mark

        while (num != -1 ) //1 Mark
        {
                cout << "Enter the amount you want to update the total amount due with: ";
                cin >> amnt;  //0.5 Mark
                for (acc = 0; acc < ARRAYSIZE; acc++)//1 Mark
                {
                        if (num == accNum[acc])//1 Mark
                        {
                                accTotDue[acc] = accTotDue[acc] - amnt ;//1 Mark
                                acc = ARRAYSIZE; //0.5 Mark
                        }
                }
                if (acc == ARRAYSIZE) //0.5 Mark
                        cout << "\nAccount number not found\n"; //0.5 Mark


                cout << "\nEnter the account number that you would like \nto update the total amount due for (-1 to end): ";
                cin >> num ;//0.5 Mark
        }
}

double calcIntPayable (int iRate, double amnt)//2 Mark
{
        double intPayable;
        intPayable = iRate / 100.0 * amnt ;//1 Mark
        return intPayable; //0.5 Mark
}
void displayBillingReport()//1 Mark
{
        double totAmntPayable = 0.0;
```



© University of Pretoria

```
        double highestAmntDue = accTotDue[0]; //0.5 Mark
        string nameHighest = accName[0]; //0.5 Mark
        int acc;

        cout << "\nAccount Number      Account Name    Total Still Due     Term Remaining      Monthly Amount Due\n";
        cout << "=============      ============    =============== ==============      ==================\n";
        cout << endl;

        cout << fixed << setprecision(2);//1 Mark

        for (acc = 0; acc < ARRAYSIZE; acc++)//1 Mark
        {
                cout << accNum[acc] << setw(25) << accName[acc] << setw(10) <<"R"<< accTotDue[acc] << setw(20)
                             << accRemTerm[acc] << setw(13) <<"R"<< accMonthlyDue[acc] << endl;//3 Mark - 0.5 each output 0.5 setw

                totAmntPayable = totAmntPayable + accTotDue[acc]; //1 Mark

                if     (highestAmntDue < accTotDue[acc])//1 Mark
                {
                        highestAmntDue = accTotDue[acc]; //1 Mark
                        nameHighest = accName[acc]; //1 Mark
                }
        }
        cout << "\n=========================================================================\n";
        cout << "Total AmountPayable = R" << totAmntPayable << endl;//0.5 Mark
        cout << "The name of the account holder with the highest \namount still due is "<< nameHighest <<endl;//0.5 Mark
        cout << "=========================================================================\n";
}
//[TOTAL = 40]
```



© University of Pretoria

## VB.Net Solution for Question 5

```
Option Explicit On
Option Strict On

Public Class frmDisplayBillingReport
    Const INTARRAYSIZE As Integer = 10I '0.5 Mark
    Private intAccNum() As Integer = {813465, 548926, 662908, 713890, 412579, 916356, 528349, 920345, 710293, 820394} '2 Marks
    Private strAccName() As String = {"T Mahlangu", "R Pretorius", "K More", "F Knauff", _
                                      "V Raphiri", "H Raseroka", "M Malebane", "Z Kgoete", _
                                      "S Meintjes", "T Radebe"} '2 Marks
    Private decAccTotDue() As Decimal = {1200D, 1570D, 5600D, 5200D, 3200D, 6600D, 13000D, 1500D, 4800D, 2000D} '2 Marks
    Private intAccRemTerm() As Integer = {4, 5, 4, 6, 2, 10, 1, 3, 10, 4} '1 Mark
    Private intAccIntRate() As Integer = {9, 12, 7, 10, 6, 15, 11, 8, 13, 9} '1 Mark
    Private decAccMonthlyDue() As Decimal = {0D, 0D, 0D, 0D, 0D, 0D, 0D, 0D, 0D, 0D} '1 Mark

    Private Sub btnCalculateIntPayable_Click(sender As Object, e As EventArgs) Handles btnCalculateIntPayable.Click
        Dim intAcc As Integer
        Dim decInterestPayable As Decimal

        Call UpdateTotalDue() '1 Mark

        For intAcc = 0 To INTARRAYSIZE - 1 '1 Mark
            decInterestPayable = CalcIntPayable(intAccIntRate(intAcc), decAccTotDue(intAcc)) '2 Marks
            decAccMonthlyDue(intAcc) = decAccMonthlyDue(intAcc) - decInterestPayable '2 Marks
        Next intAcc

        Call DisplayBillingReport() '1 Mark

    End Sub

    Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
        lstDisplayBillingReport.Items.Clear()
    End Sub

    Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
        Me.Close()
    End Sub
    Private Sub UpdateTotalDue() '1 Mark
        Dim intNumber, intAcc As Integer
        Dim decAmt As Decimal
```



© University of Pretoria

```
    Integer.TryParse(InputBox("Enter the account number that you would like" & _
                              " to update the total amount due for (-1 to end)", _
                              "Account Number"), intNumber) '1 Mark
    Do While intNumber <> -1 '1 Mark
        Decimal.TryParse(InputBox("Enter the amount you want to " & _
                                  "update the total amount due with", "Amount"), decAmt) '1 Mark

        For intAcc = 0 To INTARRAYSIZE - 1 '1 Mark
            If intNumber = intAccNum(intAcc) Then '1 Mark
                decAccTotDue(intAcc) = decAccTotDue(intAcc) - decAmt '2 Mark
                intAcc = INTARRAYSIZE '0.5 Mark
            End If
        Next intAcc
        If intAcc = INTARRAYSIZE Then '0.5 Mark
            MessageBox.Show("Account number not found", "Invalid Account Number",_
                            MessageBoxButtons.OK, MessageBoxIcon.Stop) '0.5 Mark
        End If
        Integer.TryParse(InputBox("Enter the account number that you would like" & _
                              " to update the total amount due for (-1 to end)", _
                              "Account Number"), intNumber) '0.5 Mark
    Loop
End Sub

Private Function CalcIntPayable(ByVal intIRate As Integer, _
                                ByVal decAmnt As Decimal) As Decimal '2 Mark
    Dim decIntPayable As Decimal = intIRate / 100D * decAmnt '1 Mark
    Return decIntPayable '0.5 Mark
End Function
Private Sub DisplayBillingReport() '1 Mar
    Dim decTotAmntPayable As Decimal = 0D
    Dim decHighestAmntDue As Decimal = decAccTotDue(0) '0.5 Mark
    Dim strNameHighest As String = strAccName(0) '0.5 Mark

    lstDisplayBillingReport.Items.Add("Account Number      Account Name    " & _
                                      "Total Still Due  Term Remaining      Monthly Amount Due")
    lstDisplayBillingReport.Items.Add("=============       ============    " & _
                                      "===============  ==============      ==================")
    For intAcc = 0 To INTARRAYSIZE - 1
        lstDisplayBillingReport.Items.Add(intAccNum(intAcc).ToString.PadRight(20) & strAccName(intAcc).PadRight(16) & _
                                          ("R" & decAccTotDue(intAcc).ToString("N2")).PadRight(20) & _
                                          intAccRemTerm(intAcc).ToString("N2").PadRight(16) & _
```



© University of Pretoria

```
                                        "R" & decAccMonthlyDue(intAcc).ToString("N2")) '3 Marks - 0.5 each output 0.5 for
PadRight
            decTotAmntPayable = decTotAmntPayable + decAccTotDue(intAcc) '1 Mark
            If decHighestAmntDue < decAccTotDue(intAcc) Then '1 Mark
                decHighestAmntDue = decAccTotDue(intAcc) '1 Mark
                strNameHighest = strAccName(intAcc) '1 Mark
            End If
        Next intAcc
        lstDisplayBillingReport.Items.Add("")
        lstDisplayBillingReport.Items.Add("=========================================================")
        lstDisplayBillingReport.Items.Add("Total Amount Payable = R" & decTotAmntPayable.ToString("N2")) '0.5 Mark
        lstDisplayBillingReport.Items.Add("The name of the account holder with ")
        lstDisplayBillingReport.Items.Add("the highest amount still due is " & strNameHighest) '0.5 Mark
        lstDisplayBillingReport.Items.Add("=========================================================")
    End Sub
    '[TOTAL = 40]
End Class
```



© University of Pretoria

## Appendix D: Results of Analysis of Question Papers

| Mathematical and Programming Thinking Skills Matrix for the Analysis of Programming Assessments | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Mathematical Thinking Skills | Summary (Questions 1 to 6) | | | | | | | | | | | | | | | | | | Programming Thinking Skills |
| | Q 1 | Possible | % | Q 2 | Possible | % | Q 3 | Possible | % | Q 4 | Possible | % | Q 5 | Possible | % | Q 6 | Possible | % | |
| The manipulation of formulae | 14 | 40 | 35 | 18 | 40 | 45 | 4 | 4 | 100 | 4 | 4 | 100 | 4 | 4 | 100 | 4 | 4 | 100 | Problem solving |
| Making use of mathematics to represent realities | 4 | 40 | 10 | 1 | 40 | 2.5 | 2 | 4 | 50 | 2 | 4 | 50 | 4 | 4 | 100 | 4 | 4 | 100 | |
| Behaviour of functions | 12 | 40 | 30 | 0 | 40 | 0 | 0 | 4 | 0 | 2 | 4 | 50 | 2 | 4 | 50 | 2 | 4 | 50 | |
| Breaking down and simplifying problems | 11 | 40 | 27.5 | 10 | 40 | 25 | 4 | 4 | 100 | 4 | 4 | 100 | 4 | 4 | 100 | 4 | 4 | 100 | Algorithmic thinking |
| Algorithmic thinking | 25 | 40 | 62.5 | 19 | 40 | 47.5 | 4 | 4 | 100 | 4 | 4 | 100 | 4 | 4 | 100 | 4 | 4 | 100 | |
| Generalisation of concepts/problems | 16 | 40 | 40 | 19 | 40 | 47.5 | 4 | 4 | 100 | 4 | 4 | 100 | 4 | 4 | 100 | 4 | 4 | 100 | Reasoning skills |
| Information structures | 8 | 40 | 20 | 0 | 40 | 0 | 2 | 4 | 50 | 2 | 4 | 50 | 2 | 4 | 50 | 2 | 4 | 50 | |
| Abstract reasoning | 22 | 40 | 55 | 19 | 40 | 47.5 | 4 | 4 | 100 | 4 | 4 | 100 | 4 | 4 | 100 | 4 | 4 | 100 | |



© University of Pretoria

**Appendix E: Permission letter**

Tshwane University of Technology
*We empower people*

Faculty of ICT

Department of Computer Science

25 April 2016

**Request for permission to conduct research at Tshwane University of Technology**

Dear Ms Coetzee

After careful consideration of your request to obtain access to Development Software I (DSO17AT and DSO17BT) assessments of 2014 and 2015 for research purposes (towards your Master's degree), we have decided to grant permission that you can proceed with the analysis of the assessments.

We wish you success with your research and your studies.

Regards,

Dr Manoj Lall

Head of Department: Computer Science



© University of Pretoria

## Appendix F: Information leaflet and Informed Consent (Assessments)

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Faculty of Education
Fakulteit Opvoedkunde
Lefapha la Thuto

## INFORMATION LEAFLET AND INFORMED CONSENT

## PROJECT TITLE: MATHEMATICAL THINKING SKILLS NEEDED BY FIRST YEAR PROGRAMMING STUDENTS

Primary investigator: Ms Carla Coetzee

Study leader: Prof Gerrit Stols,
Department of Science, Mathematics and Technology Education (SMTE),
Faculty of Education, University of Pretoria

### Dear Potential research participant,

You are invited to participate in a research study that forms part of my formal Med (SMTE) studies. This information leaflet will help you to decide if you would like to participate. Before you agree to take part, you should fully understand what is involved. You should not agree to take part unless you are completely satisfied with all aspects of the study.

### WHAT IS THE STUDY ALL ABOUT?

The aim of this study is to identify the mathematical skills that first year students require when they enrol for a first level programming subject towards obtaining a National Diploma in ICT.

Mathematics is an entry requirement for many tertiary courses, including ICT courses. Students in South Africa demonstrates poor performance in and this affects many prospective students' programming careers negatively.



© University of Pretoria

The students entering the ICT Foundation Course at TUT have proven to be under-prepared for the problem-solving nature of the ICT Diploma. In view of this, TUT introduced a new foundation subject called ICT Foundation Mathematical Skills.

The idea behind the subject is to close the gap between Grade 12 Mathematics/Mathematical Literacy in terms of the mathematical thinking skills and concepts required for programming. This module should developed students' relevant mathematical thinking skills in preparation for programming in the National Diploma. Identifying and teaching the much needed mathematical thinking skills for effective programming should benefit students and hopefully contribute towards improved success rates.

**WHAT WILL YOU BE REQUIRED TO DO IN THE STUDY?**

In order to collect data for this research, I would like to obtain access to Development Software I (DSO17AT and DSO17BT) assessments of 2014 and 2015 to identify these skills in the assessments. The assessments will be analysed qualitatively to confirm the necessity of these identified mathematical thinking skills in first year programming. The analysis will only identify the skills and will not be used to do quality assurance of assessments. The names of the examiners and moderators will not be mentioned. The assessments will be analysed in such a way that no reference will be made to which question of which paper is analysed and not all questions of all assessments will be analysed.

**ARE THERE ANY CONDITIONS THAT MAY EXCLUDE YOU FROM THE STUDY?**

Only examiners and internal moderators will take part.

**CAN ANY OF THE STUDY PROCEDURES RESULT IN PERSONAL RISK, DISCOMFORT OR INCONVENIENCE?**

There is no risk attached to participation.

**WHAT ARE THE POTENTIAL BENEFITS THAT MAY COME FROM THE STUDY?**

The benefits of participating in this study are:

- You will make a contribution towards the improvement of the Foundation and First Year mathematics modules to enhance and improve the students' mathematical thinking skills for programming which in turn may lead to improved success rates.

**WILL YOU RECEIVE ANY FINANCIAL COMPENSATION OR INCENTIVE FOR PARTICIPATING IN THE STUDY?**

Please note that you **will not** be paid to participate in the study.



© University of Pretoria

**WHAT ARE YOUR RIGHTS AS A PARTICIPANT IN THIS STUDY?**

Your participation in this study is entirely voluntary. You have the right to withdraw at any stage without any penalty or future disadvantage whatsoever. You don't even have to provide the reason/s for your decision. Your withdrawal will in no way influence your continued working relationship with the researcher. Note that you are not waiving any legal claims, rights or remedies because of your participation in this research study.

**HOW WILL CONFIDENTIALITY AND ANONYMITY BE ENSURED IN THE STUDY?**

All information obtained during the course of this study is strictly confidential. The study data will be coded so that it will not be linked to your name. Your identity will not be revealed while the study is being conducted or when the study is reported in scientific journals. All the data sheets that have been collected will be stored in a secure place. Any information that is obtained in connection with this study and that can be identified with you will remain confidential and will be disclosed only with your permission or as required by law. The information received during the project will only be used for research purposes and will not be released for any employment-related performance evaluation, promotion and/or disciplinary purposes.

Only the researcher and the supervisors will have access to the analysis of the assessments. Your contribution will be totally anonymous and your identity will not be revealed under any circumstance. Also, nobody outside the study panel and/or research ethics committee will be able to connect the assessment in any recognisable way. The results of this study might be published in a scientific journal and/or presented at scientific meetings, but again without revealing the identity of any research participant. The original analysis will be stored in a safe place for three years, after which they will be destroyed.

**IS THE RESEARCHER QUALIFIED TO CARRY OUT THE STUDY?**

The researcher comes from the same field of study and faculty as you. This means that he/she understands the content and context of your lecturing experiences.

**HAS THE STUDY RECEIVED ETHICAL APPROVAL?**

Yes. The Faculty of Education Research Ethics Committee of the University of Pretoria has approved the formal study proposal (Ref#: SM 15/10/01). Also, the TUT Research Ethics Committee granted approval to collect data in the TUT context (Ref#: 2016/04/002). All parts of the study will be conducted according to internationally accepted ethical principles.

**WHO CAN YOU CONTACT FOR ADDITIONAL INFORMATION REGARDING THE STUDY?**

*The primary investigator, Ms Carla Coetzee, can be contacted during office hours at Tel (012) 382-9233, or on her cellular phone at 082 921 6382. The study leader, Prof Gerrit Stols, can be contacted gerrit.stols@up.ac.za. Should you have any questions regarding the ethical aspects of the study, you can contact the chairperson of the TUT Research Ethics Committee, Dr WA Hoffmann, during office hours at Tel (012) 382-6265/46, E-mail hoffmannwa@tut.ac.za. Alternatively, you can report any serious unethical behaviour at the University's Toll Free Hotline 0800 21 23 41.*



© University of Pretoria

**DECLARATION: CONFLICT OF INTEREST**

The following existing/potential conflict(s) of interest may arise:

- Participants are staff members at TUT and in the same faculty as the researcher. However, they are in a different departments and seldom interact.
- Further, the purpose of the study is not to assess quality or performance in any way, participants take part voluntary, and anonymity is guaranteed in the reporting of the results.

**A FINAL WORD**

Your co-operation and participation in the study will be greatly appreciated. Please sign the informed consent below if you agree to participate in the study. In such a case, you will receive a copy of the signed informed consent from the researcher.

---

**CONSENT**

I hereby confirm that I have been adequately informed by the researcher about the nature, conduct, benefits and risks of the study. I have also received, read and understood the above written information. I am aware that the results of the study will be anonymously processed into a research report. I understand that my participation is voluntary and that I may, at any stage, without prejudice, withdraw my consent and participation in the study. I had sufficient opportunity to ask questions and of my own free will declare myself prepared to participate in the study.

Research participant's name: _____ (Please print)

Research participant's signature: _____Date: _____

Researcher's name: _____ (Please print)

Researcher's signature: _____ Date: _____



© University of Pretoria

## Appendix G: Information leaflet and Informed Consent (Participation in Interviews)

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Faculty of Education
Fakulteit Opvoedkunde
Lefapha la Thuto

## INFORMATION LEAFLET AND INFORMED CONSENT

**PROJECT TITLE: MATHEMATICAL THINKING SKILLS NEEDED BY FIRST YEAR PROGRAMMING STUDENTS**

Primary investigator: Ms Carla Coetzee

Study leader: Prof Gerrit Stols,
Department of Science, Mathematics and Technology Education (SMTE),
Faculty of Education, University of Pretoria

### Dear Potential research participant,

You are invited to participate in a research study that forms part of my formal Med (SMTE) studies. This information leaflet will help you to decide if you would like to participate. Before you agree to take part, you should fully understand what is involved. You should not agree to take part unless you are completely satisfied with all aspects of the study.

### WHAT IS THE STUDY ALL ABOUT?

The aim of this study is to identify the mathematical skills that first year students require when they enrol for a first level programming subject towards obtaining a National Diploma in ICT.

Mathematics is an entry requirement for many tertiary courses, including ICT courses. Students in South Africa demonstrates poor performance in and this affects many prospective students' programming careers negatively.



© University of Pretoria

The students entering the ICT Foundation Course at TUT have proven to be under-prepared for the problem-solving nature of the ICT Diploma. In view of this, TUT introduced a new foundation subject called ICT Foundation Mathematical Skills.

The idea behind the subject is to close the gap between Grade 12 Mathematics/Mathematical Literacy in terms of the mathematical thinking skills and concepts required for programming.

This module should developed students' relevant mathematical thinking skills in preparation for programming in the National Diploma. Identifying and teaching the much needed mathematical thinking skills for effective programming should benefit students and hopefully contribute towards improved success rates.

**WHAT WILL YOU BE REQUIRED TO DO IN THE STUDY?**

I would like to conduct semi-structured interviews with the lecturers who are lecturing (or have lectured in the past) DSO17AT and DSO17BT.

If you decide to take part in the study, you will be required to do the following:

- To sign this informed consent form
- To participate in a semi-structured interview with the researcher. You will be required to provide your opinions and/or insights of the study theme during the discussion that will last for approximately 45 minutes. Note that the discussions will be audio-recorded.

**ARE THERE ANY CONDITIONS THAT MAY EXCLUDE YOU FROM THE STUDY?**

Lecturers with less than 2 years' experience will be excluded from this study.

**CAN ANY OF THE STUDY PROCEDURES RESULT IN PERSONAL RISK, DISCOMFORT OR INCONVENIENCE?**

There is no risk attached to participation.

**WHAT ARE THE POTENTIAL BENEFITS THAT MAY COME FROM THE STUDY?**

The benefits of participating in this study are:

- You will make a contribution towards the improvement of the Foundation and First Year mathematics modules to enhance and improve the students' mathematical thinking skills for programming which in turn may lead to improved success rates.

**WILL YOU RECEIVE ANY FINANCIAL COMPENSATION OR INCENTIVE FOR PARTICIPATING IN THE STUDY?**

Please note that you **will not** be paid to participate in the study.



© University of Pretoria

**WHAT ARE YOUR RIGHTS AS A PARTICIPANT IN THIS STUDY?**

Your participation in this study is entirely voluntary. You have the right to withdraw at any stage without any penalty or future disadvantage whatsoever. You don't even have to provide the reason/s for your decision. Your withdrawal will in no way influence your continued working relationship with the researcher. Note that you are not waiving any legal claims, rights or remedies because of your participation in this research study.

**HOW WILL CONFIDENTIALITY AND ANONYMITY BE ENSURED IN THE STUDY?**

All information obtained during the course of this study is strictly confidential. The study data will be coded so that it will not be linked to your name. Your identity will not be revealed while the study is being conducted or when the study is reported in scientific journals. All the data sheets that have been collected will be stored in a secure place. Any information that is obtained in connection with this study and that can be identified with you will remain confidential and will be disclosed only with your permission or as required by law. The information received during the project will only be used for research purposes and will not be released for any employment-related performance evaluation, promotion and/or disciplinary purposes.

Only the researcher and the supervisors will have access to the transcripts and recordings of the interviews. Your answers will be totally anonymous and your identity will not be revealed under any circumstance. Also, nobody outside the study panel and/or research ethics committee will be able to connect any answer to you in any recognisable way. The results of this study might be published in a scientific journal and/or presented at scientific meetings, but again without revealing the identity of any research participant. The original transcripts and recordings will be stored in a safe place for three years, after which they will be destroyed.

**IS THE RESEARCHER QUALIFIED TO CARRY OUT THE STUDY?**

The researcher comes from the same field of study and faculty as you. This means that he/she understands the content and context of your lecturing experiences.

**HAS THE STUDY RECEIVED ETHICAL APPROVAL?**

Yes. The Faculty of Education Research Ethics Committee of the University of Pretoria has approved the formal study proposal (Ref#: SM 15/10/01). Also, the TUT Research Ethics Committee granted approval to collect data in the TUT context (Ref#: 2016/04/002). All parts of the study will be conducted according to internationally accepted ethical principles.

**WHO CAN YOU CONTACT FOR ADDITIONAL INFORMATION REGARDING THE STUDY?**

The primary investigator, Ms Carla Coetzee, can be contacted during office hours at Tel (012) 382-9233, or on her cellular phone at 082 921 6382.



© University of Pretoria

The study leader, Prof Gerrit Stols, can be contacted gerrit.stols@up.ac.za.

Should you have any questions regarding the ethical aspects of the study, you can contact the chairperson of the TUT Research Ethics Committee, Dr WA Hoffmann, during office hours at Tel (012) 382-6265/46, E-mail hoffmannwa@tut.ac.za. Alternatively, you can report any serious unethical behaviour at the University's Toll Free Hotline 0800 21 23 41.

**DECLARATION: CONFLICT OF INTEREST**

The following existing/potential conflict(s) of interest may arise:

- Participants are staff members at TUT and in the same faculty as the researcher. However, they are in a different departments and seldom interact.
- Further, the purpose of the study is not to assess quality or performance in any way, participants take part voluntary, and anonymity is guaranteed in the reporting of the results.

**A FINAL WORD**

Your co-operation and participation in the study will be greatly appreciated. Please sign the informed consent below if you agree to participate in the study. In such a case, you will receive a copy of the signed informed consent from the researcher.

---

**CONSENT**

I hereby confirm that I have been adequately informed by the researcher about the nature, conduct, benefits and risks of the study. I have also received, read and understood the above written information. I am aware that the results of the study will be anonymously processed into a research report. I understand that my participation is voluntary and that I may, at any stage, without prejudice, withdraw my consent and participation in the study. I had sufficient opportunity to ask questions and of my own free will declare myself prepared to participate in the study.

Research participant's name: _____ (Please print)

Research participant's signature: _____Date: _____

Researcher's name: _____ (Please print)

Researcher's signature: _____ Date: _____



© University of Pretoria