AN INVESTIGATION OF THE IMPACT OF HUMAN COGNITION ON THE ACQUISITION OF COMPUTER PROGRAMMING SKILLS BY STUDENTS AT A UNIVERSITY

By

SANJAY RANJEETH
2008

Dissertation in compliance with the requirements for the Master's Degree in Technology: Information Technology, Durban University of Technology

I declare that this dissertation is my own work and has not been submitted for any other degree or examination at any other institution.

_____
Sanjay Ranjeeth

APPROVED FOR FINAL SUBMISSION

_____ _____
Dr Ramu Naidoo Date

ABSTRACT

This study aimed to explore the impact of cognitive ability on the understanding of computer programming by students enrolled for a programming course at the University of KwaZulu-Natal. The rationale for this study is provided by the general perception held by the academic community that computer programming is a difficult faculty to master. This assertion is corroborated by reports of high failure rates in computer programming courses at tertiary institutes.

A literature review was undertaken to investigate the contribution of other factors on the ability to achieve competence in computer programmer. Based on the outcome of the literature review, this study argues that cognitive ability warrants a higher priority relative to the other factors. As a strategy, cognitive science theory was consulted to establish a framework to quantify competency in computer programming. On the basis of this endeavour, two protocols were identified to facilitate the quantification process. The first was the “deep and surface” protocol used in previous studies to ascertain students’ cognitive style of understanding for computer programming. The second was an error analysis framework which was developed as part of the current study.

These protocols were used as frameworks to underpin the data collection phase of the study. This study found that at least 50% of the students enrolled in a computer programming course adopt a superficial approach to the understanding of computer programming. In order to explain this phenomenon, a cognitive ability test was administered. Here it was established that at least 39% of these students have not reached a level of cognitive development that will enable the invocation of abstract thought. The study also found that this inability to handle abstractionism, an essential requirement for success in computer programming, is reflected in the severity of errors made in computer programming assessment tasks.

TABLE OF CONTENTS

CHAPTER ONE 1

1 BACKGROUND TO THE STUDY

1.1 Introduction 1

1.2 Statement of Problems 2

1.2.1 Problem statement 2

1.2.2 Research objectives 3

1.2.2.1 Research sub-objectives 4

1.3 Rationale for the study 4

1.4 Research Design 5

1.5 Scope of the Study/Delimitations 7

1.6 Contributions to Academia 8

CHAPTER TWO 10

2 LITERATURE REVIEW

2.1 Introduction 10

2.2 Factors that influence the learning of programming 10

2.2.1 The influence of culture 11

2.2.2 Previous programming experience 11

2.2.2.1 The South African context 12

2.2.2.2 The Literary perspective 12

2.2.3 Learning Styles 14

2.2.4 Mathematical, academic and gender issues 17

2.3 The Cognitive dimension – A possible panacea 19

2.3.1 Best predictor of programming proficiency 19

2.3.2 Human cognitive theory 20

2.3.3 Research in human cognition and computer programming 21

2.3.4 Piagetian model for human cognitive development 22

2.3.4.1 The Piagetian stages of cognitive maturity 22

2.3.4.2 Piaget's theory of learning 24

2.3.5 Frame Theory 24

2.4 Conclusion of the literature review 26

CHAPTER THREE 28

3 THEORETICAL FRAMEWORK FOR PROGRAMMING ASSESSMENT

3.1 Introduction 28

3.2 The Error Analysis Framework for Computer Programming 28

3.3 Illustration of the Error Analysis Framework for Computer Programming 31

3.3.1 Input/Output Control 31

3.3.2 Looping/Iteration 33

3.3.3 Sub-routines/Procedures – Modular Programming 37

3.3.4 Flowchart Implementation Error Analysis 39

3.4 Application of Error Analysis Framework 40
3.4.1 Flowchart Implementation 40
3.4.2 Data Structure Error Analysis 42
3.5 Final Word on Error Analysis Framework 44

CHAPTER FOUR 45
4 RESEARCH METHODOLOGY
4.1 Research Paradigm 45
4.1.1 Qualitative Research 46
4.2 Research Strategy 47
4.3 The Pilot Study 48
4.4 Academic Framework for Data Collection 49
4.4.1 The Semi-structured Interview 49
4.4.1.1 General Understanding of programming related concepts 49
4.4.1.2 Programming Related Problem Solving 52
4.4.2 The Cognitive Ability Test 53

CHAPTER FIVE 58
5 DATA COLLECTION AND ANALYSIS
5.1 Overall Research Design 58
5.2 Target Population 59
5.3 Overall Sampling Strategy 60
5.4 Interviews 60
5.4.1 The Interview Sample 60
5.4.2 Interview Sessions 61
5.4.3 Interview Data Presentation 62
5.4.4 Interview Data Analysis 64
5.5 Cognitive Ability Testing 66
5.5.1 Cognitive Test Sample 66
5.5.2 Cognitive Test Session 66
5.5.3 Marking Instrument 67
5.5.4 Cognitive Data Analysis 69
5.5.5 Inferential Dimension to the Cognitive Ability Tests 73
5.6 Error Analysis 74
5.6.1 Error Analysis Sample 74
5.6.2 Error Analysis Methodology 74
5.6.3 Error Analysis Data Presentation 75
5.6.3.1 Flowchart Implementation 75
5.6.3.2 Algorithmic manipulation 77
5.6.3.3 Data Structure Implementation 80
5.6.4 Error Data Relative to Cognitive Ability 83

CHAPTER SIX 89
6 CONCLUSIONS AND RECOMMENDATIONS
6.1 Attainment of Sub-Objectives 89
6.1.1 Insight into the understanding of computer programming 89
6.1.2 Development of a Theoretical Framework for the Analysis of performance in Computer Programming 90
6.1.3 Establishing the level of cognitive development of a computer programming student 91
6.1.4 Investigation of a relationship between cognitive ability and performance in computer programming 91
6.2 Conclusions 92
6.3 Recommendations 93
6.3.1 Easing the Cognitive burden 94
6.3.2 The Scripting Languages 95
6.3.3 The Visual Languages 96
6.3.4 A Curricular Strategy 98

REFERENCES 99
Appendix A Student Interview Questionnaire
Appendix B Cognitive Ability Test
Appendix C Computer Programming Examination
Appendix D Multiple Correlation Analysis
Appendix E Ethical clearance and consent form
Appendix F Data presentation

LIST OF TABLES AND FIGURES

	Description	Page Number
Table 1	Categories of logical assessment	57
Table 2	Demographic profile of interviewees	61
Table 3	Marking Scheme for Cognitive Ability Test	67
Table 4	Frequency counts of error types relative to level of cognitive maturity	83
Figure 1	Error Framework for Input/Output	32
Figure 2	Mind map for array processing	34
Figure 3	Error framework for computation of average	35
Figure 4	Error framework for “above average” count	36
Figure 5	Error framework for procedural processing	38
Figure 6	Error framework for flowcharting	39
Figure 7	Error framework applied to flowchart from the data collection instrument	41
Figure 8	User defined data structure used for error analysis	42
Figure 9	Mental image of user defined data structure	43
Figure 10	Error framework applied to user defined data structure	44
Figure 11	A sample test of propositional logic	56
Figure 12	Research Design	58
Figure 13	Results of Interview Analysis using the Deep and Surface protocol	65
Figure 14	Cognitive Test Sample from Student SG	68
Figure 15	Graph of frequency counts for Cognitive Ability test	69
Figure 16	Samples from Cognitive Ability Test	70
Figure 17	Graph showing proportional levels of Cognitive Development	71
Figure 18	Graph of sensitivity analysis for levels of cognitive development	72
Figure 19	Sample from script T1 illustrating an arbitrary error	75
Figure 20	Sample illustrating executive errors from script R1	76
Figure 21	Samples illustrating structural errors	77
Figure 22	Sample from script BB1 illustrating an arbitrary error	78
Figure 23	Sample from script F1 illustrating a structural error	79
Figure 24	Code declaration for user defined data structure	80
Figure 25	Sample from script C2 illustrating arbitrary errors for user defined data structure	81
Figure 26	Sample from script Q3 illustrating structural errors for user defined data structure	82
Figure 27	Samples illustrating executive errors for user defined data structure	82
Figure 28	Graph illustrating error type relative to levels of cognitive maturity	84
Figure 29	Graph illustrating cognitive levels relative to error types	85
Figure 30	Regression analysis graph of arbitrary errors relative to cognitive score	85
Figure 31	Regression analysis graph of executive errors relative to cognitive score	86
Figure 32	Regression analysis graph of structural errors relative to cognitive score	87

Acknowledgements

I would like to hereby convey acknowledgement and gratitude to Dr Ramu Naidoo for his assistance in making this study possible. His expert guidance in the field of human cognition and error analysis is highly appreciated.

I would also like to thank the staff and students of the Pietermaritzburg and Westville campuses of the University of KwaZulu-Natal for availing me of their time and assistance during the data collection phase of this study.

Chapter One

1. BACKGROUND TO THE STUDY

1.1 Introduction

Since the inception of the concept of software development, the mastery of computer programming seems to have been tagged as perpetually problematic. This assertion is corroborated by the sentiments echoed by Dijkstra in his ACM Turing Award Lecture, "The Humble Programmer", where he said:

"…as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, now we have gigantic computers, programming has become an equally gigantic problem" (Dijkstra,1972).

Computer Programming forms the basis from which most students in the Information Systems/Information Technology (IS/IT) discipline launch themselves into further endeavours in the discipline. However, statistical analysis of students' performances in programming related assessments tasks reveals that the mastery of computer programming skills is not easily acquired. This assertion is supported by reports of high failure rates in programming courses at several academic institutes. This trend is also confirmed at the University of KwaZulu-Natal (UKZN) where programming related assessments have resulted in failure rates as high as 50%. Pea & Kurland (1984) cite numerous studies that allude to the deep misunderstanding of programming related concepts by adult novice programmers. The problem from a pedagogical perspective is further exacerbated by the power of industry to dictate the direction of academic curricula. According to Gruba *et al.* (2004), this intervention has propagated the prioritisation of industry demands over pedagogical concerns. These industrial expectations have lead to the proliferation of a variety of software development environments as well as different paradigms in software development. In his

capacity as chairman of the ACM Education Board Task Force on Computer Science Education, Roberts (2004) remarked that a major challenge facing computer science education is beginning to unfold in the form of:

- *Complexity*. The number of programming details that students must master has grown much faster than the corresponding number of high level concepts.

- *Instability*. The languages, libraries and tools on which introductory computer science education depends are changing more rapidly than they have in the past.

The dynamism inherent in the software development arena has propagated an ethos that prioritises functionality at the expense of pedagogy. This has increased the complexities that student programmers have to contend with, thereby placing an additional burden on their cognitive ability. This study purports to address this dilemma by focusing on the cognitive demands of learning to program as well as to ascertain the influence that the level of cognitive maturity has on the attainment of proficiency in computer programming. It is envisaged that the emergent discourse will have a catalytic effect and spawn an area of research that focuses on the lowering of barriers of entry into computer programming imposed by the constraints of human cognition.

1.2 Statement of Problems

1.2.1 Problem Statement

Most introductory courses offered within IT departments at tertiary educational institutes include computer programming as a major component of the course. However, the literature on the pedagogy of computer programming has not been effective in making a definitive contribution to facilitate an improved learning experience for students who have engaged themselves with the challenges posed by computer programming instruction. A possible consequence of the lack

of pedagogical strategy is the incidence of high failure rates in computer programming assessment tasks. An area of significance in this regard is the cognitive ability of students entering these courses. An ignorance of the cognitive dimension is attested to by the observation that programming instruction is delivered on the basis of the assumption that all students have the same cognitive ability. The consequence of this assumption has escaped the attention of academics due to the lack of any plausible framework to facilitate rigorous analyses of student performance and understanding of computer programming. Hence, a vital contribution to the pedagogy of computer programming will be the establishment of a framework to enable an empirical measure of the impact of the cognitive ability on a student's ability to acquire competency in computer programming. The task of establishing such a framework will become a sub-problem of the current study. However, at this juncture the problem statement will read as follows:

What is the effect of a student's cognitive ability on his/her ability to achieve competence in computer programming?

It is envisaged that the development of a framework, informed by theory on human cognition, for assessment in computer programming will be effective in establishing a reliable measure of a student's level of competence in computer programming.

1.2.2 Research Objectives

The main objective of this study is to establish the impact of cognition on the ability of university students to acquire competence in computer programming. In order to achieve this objective, it is deemed necessary to obtain an insight into the level of understanding of computer programming displayed by university students. It is envisaged that this knowledge will add credibility to the objective of this study. The following sub-objectives were identified in order to provide an agenda as well as to impart focus to the content of the study.

1.2.2.1 Research Sub-Objectives

1. To gain an insight into the level of understanding displayed by students in computer programming

2. To develop a theoretical framework for the analysis of performance in computer programming

3. To ascertain a student's level of cognitive development

4. To ascertain the impact that the level of cognitive development has on performance in computer programming

1.3 Background to the Research

According to Smith, Cypher & Schmucker (1996), there is a "…conceptual gap between the representations that people use in their minds when thinking about a problem and the representations that computers will accept when they are programmed". These sentiments tend to echo an unequivocal belief that the acquisition of competence in computer programming is no trivial achievement. Cognisance of these sentiments is attested to by a growing number of literary inquisitions attempting to identify factors that may contribute towards a diminishing of the impasse between machine processing and human cognition. According to Efopoulos *et al*. (2005), "…research in the area of Computer Science Education has been quite active during the past 25 years". This has resulted in the emergence of journals that have either exclusively focused on the teaching of computer programming or have a high proportion of publications relating to the teaching/learning of computer programming (e.g. Computer Science Education, International Journal of Human-Computer Studies, Association for Computing Machinery (ACM), Journal Storage (JSTOR)). These literary sources contain references to studies that purport to identify factors that may have a contributory effect on the acquisition of

expertise in computer programming. The factors that have received much attention include previous academic and computer experience, personality traits, gender, age and cognitive ability. The emergence of research vibrancy in the area of computer programming pedagogy is indicative of a problem of growing magnitude. A possible impediment to research efforts in the area is a lack of a theoretical quality framework to underpin these literary inquisitions. This has caused researchers in the field to resort to empiricism as a basis for their investigations. However, the lack of an encapsulating framework has resulted in a paradoxical situation where the combined effort is disparate in nature. The learning of computer programming is thus still highly problematic, with no definitive identification of a possible cause. In this study, it is hoped that this oversight is remedied.

1.4 Research Design

This study has a qualitative focus. Quantitative methods will be employed in the analysis of data collected. Hazzan *et al*. (2006), report that the qualitative approach is usually used for the investigation of social phenomena such as cases that have an educational agenda. The main strategy will entail the collection of data on the basis of cognitive development theory as well as the analysis of programming solutions provided by the subjects of the study. The uniqueness of the current study in comparison to previous research is the development of a theoretical framework, the focus of which is to provide a context for the data analysis. It is envisaged that a blended research approach combining qualitative elements, embodied by grounded theory, with quantitative data analysis will be ideal. However, due to the dynamic nature of qualitative research it would be pre-mature to provide a detailed plan for the design. A tentative, holistic design could be proposed at this stage and entails the following:

Literature Study: The literature study has a dual perspective. It is aimed at covering the existing body of knowledge on factors that influence the study of computer programming. There will also be an incursion into the epistemology that underpins the process of knowledge acquisition as well as human cognitive development. The level of detail is necessitated due to the strong significance attached to the cognitive dimension in the current study. The cognitive inquiry will be predominated by the Piagetian framework for cognitive development and Davis's theory on knowledge acquisition.

Establishment of Theoretical Frameworks*:* The current study will be underpinned by theoretical frameworks. The deep and surface framework for quantification of cognitive style towards learning will be introduced in the literature review. This framework is used to facilitate an analysis of students' responses to questions on issues related to computer programming. It is envisaged that this analysis will provide a meaningful insight into students' understanding of programming concepts. A significant component of this study entails the development of an error analysis framework fro computer programming. This framework will be informed by cognitive science theory on knowledge acquisition. The research methodology employed in the current study will also be contextualised according to an academic framework. The development of this framework could be viewed as an extension of the literature review. The framework is used to underpin the data collection phase of the current study.

***Data Collection*:** The target population will be students enrolled for the ISTN212 programming course at UKZN. The literature on the pedagogy of computer programming alludes to a vast array of factors that could have an influence on a student's programming ability. This study does not attempt to factor out these influences. However, cognisance will be accorded to these factors in the literature review as well as the adoption of a stratified sampling methodology. The factors identified in the literature will be used as a basis for classification of

the sample. The first sample for data collection will consist of students who will be selected to participate in a 45 minute semi-structured interview session on issues related to computer programming. Data will be collected via a tape recording of the responses given by the interviewees. The second sample will consist of students who will be randomly invited to participate in a written 30 minute cognitive ability test. The third data collection activity will entail a collation of responses to programming questions in the ISTN 212 examination question paper. This will be restricted to the examination scripts of those students who participated in the first two phases of data collection (i.e. the interview and the cognitive test).

***Data Analysis*:** The interviews will serve to record students' experience of the phenomenon of computer programming. A protocol analysis approach will be adopted. The protocol that will be used for the current study is the deep and surface protocol used by Booth (2001) in her theory of teaching and learning. On the basis of their performance in the cognitive ability test, students will be classified according to cognitive development stages using a Piagetian framework for cognitive development. Programming solutions provided by students will be analysed according to the theoretical framework developed as part of this study. Regression and correlation theory will be used to achieve convergence between the different data collection aspects of the study. The qualitative stance of the study necessitates the presentation, discussion and analysis of exemplars from the data that is collected. However, quantitative analysis will be employed to extend the credibility of the study from an inferential perspective. Data analysis will include frequency counts, regression analysis and the use of tables and graphs to illustrate averages and highlight relationships between the variables of the study.

Conclusion and Recommendations: A review of the achievements of the study will be conducted. It is envisaged that the emanating discourse will be able to make a positive contribution to the pedagogy of computer programming.

1.5 Scope of the Study/Delimitations

Due to logistical impediments, this research will be confined to 2nd year programming students at the University of KwaZulu-Natal (UKZN) within the School of Information Systems and Technology (IS&T). As this research will be conducted within an interpretivistic epistemology, the outcome is not focused on providing statistically generalisable results from the data gathering efforts. However, the research will be theoretically anchored in pedagogy and computing thereby facilitating a deeper understanding of the influence of the cognitive domain on the learning of computer programming. This assertion is corroborated by the claim made by Berglund *et al*. (2006) that the qualitative research paradigm enables the "… drawing of a more solid and significant conclusion about how students learn computing". Having put forth this argument, the paradigm of positivism and the scientific method will be maintained. There will be statistical verification tests conducted to establish the legitimacy of some of the conclusions as well as correlation and regression analysis to illustrate the inferential potential of the current research if it were to be replicated with a greater target population.

1.6 Contributions to Academia

In the course of its evolution, issues from the current study have been presented at the following academic forums:

- Ranjeeth, S and Naidoo, R (2007), "An Investigation into the relationship between the level of cognitive maturity and the types of errors made by students in a computer programming course", *College Teaching Methods and Styles Journal* **3**(2) p31
- Ranjeeth, S and Naidoo, R (2007), "Students' understanding of programming concepts at a university", *CSITEd Conference Proceedings*, Mauritius November 2007.
- Ranjeeth, S and Naidoo, R (2007), "Classification of Students' cognitive Abilities with Errors in Computer Programming" , A Conference

presentation at the 14th International Conference on Learning , University of Witwatersrand, (recommended for publication by referees' reports – International Journal of Learning - awaiting publication agreement)

- Ranjeeth, S and Naidoo, R (2007), "Deep and surface structures in Computer Programming" A presentation at the Faculty of Accounting and Informatics Research Day at the Durban University of Technology
- Ranjeeth, S (2006), "A Critique on the Viability of Software produced using the Open Source Methodology", A presentation at the Annual Faculty of Management Studies Conference at the University of KwaZulu-Natal, 2006, paper reference 001

Chapter Two

2. LITERATURE REVIEW

2.1 Introduction

The previous chapter served to provide an overview as well as an indication of the intended path for the current study. The dearth of detail will be addressed once the context for the study is established. Chapter Two will serve to establish a context, or theoretical framework for the evolution of the main themes that will emanate from the study.

The aim of the study makes strong reference to human cognition and the capacity of university students to acquire computer programming competence. Hence the literature review will culminate with a focus on aspects of human cognition and the relationship to computer programming. However, in order to introduce the cognitive dimension, it is imperative to provide adequate coverage of periphery issues that may also have an impact on the acquisition of computer programming skills. It is hoped that in trying to achieve coverage of these aspects, an implicit theoretical basis will be created that will facilitate an easier understanding of the emergent discourse.

2.2 Factors that influence the learning of computer programming

"*Software construction is a complex, socio-technical, cognitive process that requires a combination of technical, social, analytical and creative abilities*" (Rose *et al*., 2005). Hence, the literature on the pedagogical aspects of programming identifies several factors that have a contributory effect on one's ability to learn programming. According to Bishop-Clark (1995) and Bergin & Reilly (2005), these factors include previous academic and computer experience, cognitive skills and cognitive styles, student personality traits and experience in the

module. The list of factors that influence the learning of computer programming can be extended by adding in the influence of culture given the cultural diversity of South African society.

2.2.1 The influence of Culture

Since the cohort of students for this study is multicultural, it is necessary to comment on culture as an influence on programming. The influence of culture is manifested in how people think, solve problems and relate to others (collectively described by Bishop-Clark (1995) as cognitive style). According to Rose *et al*. (2005), people are conditioned to use the teaching, learning and/or problem-solving approaches that predominate in their culture. There have many studies that have alluded to the contribution of culture as a factor that may influence competence in programming. However, Bishop-Clark (1995) makes reference to these studies and provides a strong argument to conclude that "…work relating cognitive styles and personality traits to computer programming has been scattered and difficult to interpret". Hence the significance of cultural diversity as a contributory factor to leaning to program is still subject to much speculation.

2.2.2 Previous Programming Experience

Although Bergin and Reilly (2005) allude to previous computing experience as a factor that positively influences computer programming ability, this could well become something of a misnomer. For the purpose of the current study, the previous computing experience factor will be further qualified to refer to previous computer programming experience. An illustration of the significance of this seemingly subtle modification is provided by Wilson and Shrock (2002) where they found a negative correlation between programming and game playing. In a "mischievous" kind of interpretation, game playing would qualify as previous computing experience. The discussion on the effects of previous experience on computer programming proficiency is preceded by a brief reference to previous computer programming experience from a South African context.

2.2.2.1 The South African Context

In the South African context, previous programming experience may allude to students who have taken Computer Studies as a Higher Grade subject at Matric level. The Pascal programming language was the default language of implementation. Students were selected to do Computer Studies on the basis of their cognitive ability. In order to assess this ability, students were given general problem solving assessment tasks. The results of these tasks were used in conjunction with the marks obtained in Mathematics and Science to determine eligibility for enrolment in a computer programming class[1]. Hence there is every prospect that from a South African context, the previous experience factor may be somewhat diluted by the underlying contribution of superior cognitive ability.

2.2.2.2 The Literary Perspective

Wiedenbeck (2005) cites many literary references that attest to the positive effect of previous experience on computer programming. The strength of this assertion is echoed by his sentiment that "…in the CS literature the most frequently mentioned factor for success in introductory programming is previous experience". Hagan and Markham (2000) conducted a longitudinal study that attempted to establish whether previous experience had a significant effect on programming performance. The study spanned a period of two years and there was a participant response rate of approximately 33%. The outcome of this study was that students who had previous programming experience were able to leverage this knowledge to enjoy an advantage over students with no prior experience in computer programming. However, an interesting outcome of the study was that the advantage decreased over the duration of the period of instruction. The decline in the advantage gained by previous experience is more significant if the style of instruction is different from what was learnt previously. This point is illustrated by the following example, quoted by the authors of the study:

[1] The researcher can offer a verification of this assertion on the basis of personal involvement in the developnment and implementation of this policy whilst serving on the Computer Studies Provincial Committee for the KwaZulu-Natal Department of Education.

"If a student had learnt Pascal at school, her familiarity with computers, problem solving, and the basics of programming such as variables and functions would help at the beginning. However, if we were teaching an object-oriented language such as Java, she would have to learn a large number of new concepts, some of which might appear to contradict what she knew before, and would therefore be confusing" (Hagan and Markham, 2000).

In a study on computer programming teaching pedagogies, Ali (2005) referred to the concept of constructivism as "new knowledge" that was generated when information encountered cognitively interacts with what the learner already knows. Hence, in terms of this theory, previous programming experience would be pivotal in ensuring proficiency in the discipline. However, he argues that in the discipline of computing, "…knowledge of information technology changes very quickly". This can be attributed to the rapid growth of the Internet and the constantly changing paradigms of software development[2]. This assertion corroborates the observations made by Hagan and Markham (2000) in their study. In effect, this outcome dilutes the significance of previous experience as a contributory factor. On the basis of the issues presented here, there is evidence to suggest that previous experience does have a contributory role, but there is a suspicion that it may not be pivotal in order to achieve expertise in computer programming (viewpoint is supported in, Byrne and Lyons(2001); Allert (2004); and Bergin & Reilly (2005)). In light of the ambivalence of opinion on the contribution of previous programming experience, it can be inferred that there is no definitive study attesting to an overbearing effect of previous experience on programming performance. While the dictates of constructivist theory cannot be ignored, it is apparent that it does not solve the dilemma posed by the acquisition of competence in a dynamic discipline such as computer programming.

[2] An illustration of this point is embodied in the current transition from structured programming methodology to object-oriented programming. This could also include the latest tendency to move towards the agile methodology for software development

2.2.3 Learning Styles

Booth (2001) investigated the significance of the style of learning on the acquisition of competence in computer programming. She classified student programmers as either novices or experts. In order to make this classification, she conducted interview sessions with students to ascertain their level of understanding of computer programming. Individual responses were classified as a demonstration of either deep or surface knowledge of programming. The outcome of this exercise was an overall classification labelling a student as either an expert programmer if the majority of responses were classified as deep or a novice programmer if the majority of responses were classified as surface. There have been many definitions of the concepts of deep and surface learning (e.g. Booth & Morton (1997, p. 34); Martin and Saljo (1976); Rhem (1995); Cope & Horan (1998); Hughes & Peiris (2006); Simon *et al*. (2006)). A common theme in these definitions is that the surface approach to learning entails memorization, rote learning and consumption of knowledge from a quantity perspective for the purpose of reproduction at some assessment forum (such as an examination). The deep approach to learning entails intimate and quality driven understanding of content for the purpose of application and extension beyond the factual dimension. Lewandowski *et al*. (2005) cite various studies that are consistent with their assertion that "…experts form abstractions based on deep (semantic) characteristics rather than on surface (syntactic) characteristics". A listing of the characteristics of the deep and surface learning style framework gleaned from the sources mentioned above entails the following:

- Surface learning is related to passive processing that lacks reflection, uses low-level meta-cognitive skills and is extrinsically motivated.

- Deep learning is a product of active processing that is intrinsically motivated, reflective, and uses higher-level meta-cognitive strategies.

- Surface learning may result in good memory for facts and definitions, but has a limited ability to understand or use them.

- Deep learning, results in facility of thought derived from linking newly acquired facts and definitions into a conceptual framework of existing knowledge.

- Students who use surface learning may do well on tests that assess learning through knowledge of facts and definitions; they may not understand or be able to apply the memorized and superficially processed information.

- Students who use deep learning are able to understand, apply, and use information learned.

The impact made by this framework on the learning of computer programming cannot be ignored and is corroborated by the study undertaken by Booth (2001). The findings of Booth's phenomenographic[3] study into the understanding of computer programming from a deep and surface perspective were that students who approached learning to program as learning to code in a language or as passing the course exhibited a surface approach to learning thereby rendering themselves as novice programmers. In contrast, students who focused on understanding the problem domain adequately well in order to produce a solution that is applicable in a professional environment exhibited deeper learning traits thereby rendering themselves as experts. There have been many studies, underpinned by a qualitative stance, which have investigated the effect of deep and surface learning traits on the acquisition of knowledge. A corroboration of this assertion is the study conducted by Shertz and Weiser (1983) on

[3]Phenomenography is defined by Booth (2001) as an " …empirical research approach to the study of experience; the aim of phenomenographis research is to describe the essential and qualitative variation in the ways people experience phenomena or a particular phenomenon in their (generally educational) environment "

programming students at different academic levels of instruction. For the purpose of the study, students from an undergraduate programming class were labelled as novice programmers, while students from a post-graduate class were classified as experts. The subjects of this study were given an assortment of 27 problems that tested understanding of concepts such as string processing and data structures in the form of arrays, linked lists and records. The processing requirements involved sorting, searching and file and data structure manipulation. An analysis of responses revealed that novices represent computer programming problems initially on the basis of surface features without too much focus on the solution strategy. In contrast, experts displayed a deeper analytical attitude towards problem solving. This entailed a focus on solution method and data structures. A study conducted by Hughes and Peiris (2006) also had a similar agenda. In this study, students in a computer programming course were subjected to an interview to establish their learning profiles according to the deep and surface framework. However, this framework was modified to include a classification referred to as “strategic”. This was in reference to students whose learning profile was in-between surface and deep. The rationale here is that these students displayed a demeanour that was close to being classified as deep but did not warrant inclusion in this category. A correlation analysis was performed to examine the learning profile and academic performance from the perspective of grades obtained for the course. The study found a strong negative correlation between students who take a surface approach to learning and the grades achieved in the programming course. On the other hand, the correlation between deep learners and exam grade is a positive, yet weak one. This outcome was unexpected and a detailed analysis revealed that students classified as deep learners tend to have an exclusive focus on problem solving strategy and solution presentation from a holistic perspective. There is a lack of priority attached to the syntax and semantics of a solution. In an examination assessment, this will result in a lowering of grades. The study also revealed a strong positive correlation between academic performance and students profiled as strategic learners. An observation

commensurate with this outcome from the study is that the strategic learners had sufficient understanding of computer programming coupled with an extrinsic motivation to pass the examination, to ensure high grades in examination based programming assessment. The convergence of opinion on learning styles allude to the ability of experts to consolidate knowledge and organise internal knowledge structures based on meaningful conceptualisation of domain related content. This assertion also begins to point to the cognitive realm, where cognitive ability enhances the ability to consolidate and organise knowledge to facilitate meaningful interpretation of the problem domain. Hence the message inherent in the discourse on learning styles is that students who have not been exposed to previous programming tuition still have the potential to become expert programmers as a consequence of exploiting cognitive ability to engender a deeper learning approach.

2.2.4 Mathematical, Academic and Gender Issues

Bennedsen and Caspersen (2005) conducted a study to determine the predictor(s) of success in computer programming. The population consisted of 230 university students enrolled for a computer course where programming constituted 7 weeks of the course. Student biographical data was collected electronically via the university administration system. Amongst their findings, it is reported that age, gender and academic path or intended major did not have a significant bearing on computer programming performance. However, the students' mathematics score from high school had a significantly positive impact on grades obtained in computer programming. Pioro (2006) conducted a similar study to ascertain the influence of gender, academic background and problem solving ability (embodied by mathematics proficiency) on computer programming. The sample consisted of 117 students enrolled for an introductory programming course. Purposive sampling was used to ensure adequate representation in terms of gender, academic background and mathematics proficiency. The mathematics proficiency attribute is used as an indicator of problem solving ability and students who had taken a previous mathematics course were

calibrated in terms of their performance in the course. There was a strong relationship between grades obtained in the mathematics course and grades obtained in computer programming assessments administered as part of the computer programming course. However, according to Pioro, "the results derived from this study do not allow us to conclude that an academic major, or gender, constituted a significant factor in determining student programming ability". Hence, it can be clearly seen that mathematics has a significant bearing on computer programming proficiency. However, the influence of mathematics ability can be generalised to problem solving ability as is demonstrated in a study conducted by Pillay and Jugoo (2005) on students enrolled for an introductory computer programming course at UKZN as well as the Mangosuthu Technikon. This study also investigated various factors that could possibly contribute to programming performance. It was found that male students seemed to have performed slightly better than female students. However, the difference was found to be not statistically significant. This finding was similar to their conclusion that previous computer programming experience did not have a significant impact on computer programming performance. The study also concluded that there is a positive correlation between problem solving ability and computer programming performance. An interesting aspect of this study was the reference to the Piagetian concepts of "assimilation" and "accommodation"[4], where it was found that assimilators performed better in computer programming than non-assimilators (referred to in the study as "divergers"). This heralds an incursion into the cognitive dimension as a viable source of enquiry when it comes to ascertaining the impact of problem solving ability on computer programming proficiency. Previous studies have resorted to the use of mathematics assessment as a means of quantifying problem solving ability. However, according to Slavin (1991, p. 232), cognitive science theory stipulates that effective problem solving is only possible if optimal cognitive development has

[4]Assimilation refers to the incorporation of new information into existing mental structures; if the data are different from from the existing mental structure, then the new information is accommodated or transformed so that it will fit into a new mental structure

been achieved. Hence, the influence of human cognition on computer programming ability becomes a viable avenue of investigation.

2.3 The Cognitive Dimension – A Possible Panacea

2.3.1 Best Predictor of Programming Proficiency

In an attempt to ascertain effective predictors of computer programming proficiency, Evans and Simkin (1989) used a 100 question survey to collect detailed biographical data from all students enrolled for an entry level computer class. The content of the survey included questions that elicited information about students' age, gender, place of birth, first language, academic credentials, and previous computer experience as well as behavioural aspects such as vocational habits. The data collection phase was supplemented by the administering of a questionnaire that contained general problem solving tasks designed to test various logical skills. In order to measure programming proficiency, Evans and Simkin argue that in academic settings, class performance is an acceptable measure of student understanding of programming. In keeping with this claim, student marks for practical assignments, mid-term and final exams were averaged to obtain a measure of student proficiency in computer programming. A linear regression model was created using each of the biographical factors mentioned as the independent variables and the average course mark as the dependent variable. Amongst their findings, a significant observation is that few of the demographic, academic, prior computer exposure, or behavioural variables were particularly strong predictors of programming proficiency (values for *F*-test statistics for R^2 did not exceed 31%). However, it seemed as though a combination of several factors may be useful in forecasting computer proficiency. A similar regression model was constructed, but in this case the score obtained for the problem solving tasks was used as the independent variable. The results indicate that cognitive ability was a stronger predictor of programming performance than the demographic factors. However, it did not account for more than 23% of the variation in computer

proficiency. Although, this statistic instilled a measure of uncertainty in terms of the impact of the cognitive dimension, the study was successful in flagging the cognitive dimension as a possible factor of significance in influencing computer programming proficiency. Evans and Simkin conclude with the assertion that further research in the area of specific cognitive processes and its impact on computer programming is required.

2.3.2 Human Cognitive Theory

Research activity in the cognitive realm has largely occurred within the framework established by constructivist epistemologies of learning as advocated by Piaget and further entrenched by the philosophies of Schema Theory proposed by Bartlett(précised in Ajideh, 2003) and adapted by Asubel (précised in Slavin,1991, p.227), Generative Learning Theory proposed by Wittrock (précised in Slavin, 1991, p.268) and Davis's Frame Theory (1984).The underlying assumption made within this framework is that prior experiences and knowledge are required to facilitate assimilation of new information. Meaningful learning can only occur if the learner engages in the constructive process of building an internal representation of knowledge. However, the cognitive science perspective on computer programming is based on the prediction that "…a student's attainable level of programming skill may be constrained by cognitive abilities required in programming" (Pea and Kurland, 1984). The ability to deal with abstractions and engage in mental manipulations, referred to as the formal operations stage is the highest Piagetian stage of cognitive development. Research has shown that a majority of college students fail at many formal operational tasks (Schwebel, 1975). White and Sivitanides (2002) cite various studies that report on the definitive influence of cognitive development on one's ability to learn programming. It is reported in Petre and Blackwell (1997) that experts rely heavily on mental imagery in problem solving. They are able to build mental structures that represent a detailed conceptual model of the system that include abstract entities in preference to the concrete objects specific to the problem statement. Hence, it becomes apparent that contextual knowledge is

required from a foundation perspective to facilitate construction of these mental structures. This observation embodies an acceptance of constructivist and cognitive science perspectives. Hence it is apparent that there needs to be collusion between the constructivists and the cognitive scientists in order to explain the lack of ability of students to gain mastery of programming related concepts. However, attempts to take cognisance of the influence of the cognitive domain on the learning of computer programming have been typified by research designs that used mathematics and science proficiency as a substitute for cognitive ability (e.g. Bennedsen and Caspersen (2005); Pioro (2006); Campbell and McCabe (1984); Byrne and Lyons (2001); Wilson and Shrock (2002)). This strategy resulted in programming skills becoming the exclusive domain of the scientific community. The current trend in information technology (IT) has seen an increasing demand and reliance on computing power from the commercial sector. A consequence of this is the imperative to increase the accessibility of programming instruction to the commercial sector as well. However, this demand has contributed to the lowering of barriers of entry to programming instruction thereby perpetuating the problem of poor performance in programming related assessments. A viable solution to this dilemma is to start by investigating the impact of the cognitive dimension. This assertion is corroborated by Kurtz (1980) when he alludes to the predominance of abstract concepts within the discipline of computer programming, thereby necessitating the invocation of higher order cognitive skills. Hence, it is imperative to obtain an empirical measure of the impact of abstractionism on computer programming performance. However, an investigation of the impact of the cognitive dimension introduces complexities of its own.

2.3.3 Research in human cognition and computer programming

The dearth of literature in this area can be attributed to a failure of the positivist paradigm to embrace the complexities inherent in such a study. The use of empirical methods to underpin research in the psychological or cognitive domain of programming has not been conclusive from the perspectives of reliability and

generalisation (Sheil, 1981). According to Hazzan *et al*., (2006) "…the ominous tendency of the Hawthorne effect to discredit experimental research has resulted in the spawning of post-modern methodologies that are empirically qualitative". An example of one of these methodologies is that of phenomenograhic analysis as conducted by Booth (2001) and recently by Eckerdal, Thuné & Berglund (2005). These studies analysed student experiences in the learning of computer programming. The lack of success of the empirical approach in this area of research opens up the viability of post-modern methodologies. A viable strategy that becomes apparent on the basis of the study by Eckerdal, Thune & Bergland is the use of a methodology that incorporates existing frameworks established by the cognitive science fraternity in conjunction with qualitative methodologies such as phenomenography. There will be a brief overview of two such frameworks. The first is the Piagetian model of human cognitive development. This will be followed by Davis's Frame Theory on human cognition.

2.3.4 Piagetian Model for Human Cognitive Development

Jean Piaget, a Swiss psychologist and biologist developed a theory for human cognitive development based on a constructivist belief that the acquisition of knowledge is a process of continuous self-construction. Knowledge is acquired within the context created by logical mental structures that humans build for themselves over a period of time. Intellectual development is contextualised according to 4 stages of cognitive development.

In order to establish a framework for the application of Piagetian cognitive theory to the learning of computer programming, a brief outline of Piaget's theory of cognitive development is presented.

2.3.4.1 The Piagetian stages of cognitive maturity

Piagetian theory alludes to three distinct stages that characterises human cognitive development. These are elaborated in the following documentary:

Sensorimotor Period (pre-language stage - Birth to 2 years)

The cognitive system is limited to motor reflexes. However, with the passage of time/experience, these reflexes are developed into more sophisticated procedures.

Pre-operational thought (2 to 6/7 rears)

This period heralds an acquisition of representational skills in the area of mental imagery. However, there is no convincing evidence of comprehension of logical operations, although there is an ability to count and use the concept of numbers as well as the ability to classify objects according to different criteria.

Concrete Operations (6/7 to 11/12 years)

There is an understanding of concrete problems and there is adeptness at manipulation of physical entities. However, there is difficulty in the contemplation and understanding of abstract problems. The individual is not able to consider all possible logical consequences of a problem situation. There is also a lack of ability to propose solutions that are all encompassing in terms of covering the problem domain from all possible perspectives. Although Piaget theorised that this stage ends at around 11 to 12 years, many cognitive scientists believe that 11/12 is the very earliest at which this stage may end, so much so that many adults remain in this stage for the rest of their lives. The inevitable conclusion emanating from these postulates is that concrete operational thinkers may have difficulty in disciplines that require abstract thought.

Formal Operations – Propositional Operations (from 12 years)

This stage is heralded by the appearance of logical operations and hypothetic reasoning begins, as opposed to reasoning about tangible/concrete objects. There is an appreciation of combinatorial operations using a scheme of systematic analysis. Essentially, an individual is empowered with the ability to be an abstract thinker and is able to formulate and test hypotheses without actually

manipulating concrete objects. A formal operational thinker is inventive and able to apply learned material to solve unusual problems (i.e. use a structured foundation to solve an unstructured problem). In scientific problem solving, formal thinking enables adolescents to systematically manipulate variables and reason about unknowns such as algebraic values.

2.3.4.2 Piaget's Theory of Learning

Piaget postulates that there are mental structures that determine how data and new information are perceived. If the new data makes sense to the existing mental structure, then the new information is incorporated into the structure, a process called "assimilation". If the data is very different form the existing mental structure, then the new information is either rejected or the information is accommodated or transformed so that it will fit into the new structure. A concrete person will probably reject a concept requiring formal thought.

The transition from concrete operational to formal thinking

Mental development occurs because there is an innate desire to operate in a state of equilibrium. When information is received from the outside world which is too far away from the mental structure to be accommodated but makes enough sense that rejecting it is difficult, then the person is in a state of disequilibrium. The desire for equilibration is a strong motivating factor to sustain intellectual growth and enhance the transition from operational to formal thinking. A person with a less rigid personality structure and tolerance is more likely to make the transition.

2.3.5 Frame Theory

In order to understand how students think when they solve problems of a mathematical nature, cognitive scientists such as Davis and Asubel proposed a framework to explain the human thought processes that underlie problem solving. The current study attempts to apply the Davis framework using a

programming context. In order to achieve this objective, it is necessary to provide an overview of the Davis framework.

Problem solving entails the invocation of several unit steps of mental activity. When several unit steps are carried out in a sequential order, this sequence is referred to as a procedure. Based upon previous problem solving experience as well as acquisition of knowledge, these procedures are stored in human passive memory also referred to as long term memory. In order to implement these procedures in problem solving exercises, they need to be identified and transferred to operational memory or working memory. An encapsulation of a series of procedures and sub-procedures is referred to as a frame or a knowledge representation structure (KRS). Frames are stored in passive memory and contain the infrastructure for solutions to various problem situations. This infrastructure consists of frame variables or slots whose values depend on the actual input data from the problem domain itself. Once a problem situation is encountered, an appropriate frame is retrieved from passive memory and populated with the input data which is housed in frame variables. Any complex processing that may be required by the frame is achieved by the activation of appropriate procedures. Upon encountering a problem situation, a frame is selected and copied into the memory work space. An enhanced cognitive capacity will facilitate correct judgement resulting in the retrieval of the appropriate frame as well as achieving a correct mapping of the input data into the appropriate frame variables. In order to achieve this, invocation of the Piagetian concepts of “assimilation” and “accommodation” are required to facilitate correct incorporation of the variables into the cognitive process. However, it is at this juncture that errors begin to make an incursion into the problem solving process. Davis asserts that “…student errors turn out to be very regular and systematic and it is often possible to predict which wrong answer is given by a particular student” (Davis, 1968, p. 43). A corroboration of this theory is provided by Bransford *et al*. (1986) where it is asserted that in the event of stumbling blocks, students with high intellectual ability are able to adapt their

strategy to accommodate the new demands[5]. Students with low academic ability tend to have a high level of dependence on existing knowledge structures resulting in an inability to adapt these structures according to problem domain specifications. This is demonstrated when students of the latter classification tend to repeat errors leading to the non-deliverance of a solution or the deliverance of a solution that has structural flaws and contain many errors. This observation heralds a convergence of opinion on the significance of the association between error generation and cognitive ability i.e. an inspection of the errors found in a solution can be regarded as a reliable predictor of the level of cognitive development. The identification of errors as an avenue to obtain knowledge of the level of cognitive development is investigated from a computer programming perspective in the current study.

2.4 Conclusion of the Literature Review

The literature review suggests that factors such as age, gender, learning style, cultural influences and previous experience have minimal impact on the mastery of skills in computer programming. However, there is consensus on the positive impact of mathematical ability on computer programming. The influence of mathematics may be traced into the realm of cognitive ability, manifested by a demonstration of general problem solving ability. A significant observation with regards to cognitive development is that many students may well be still caught up in the concrete operational stage of intellectual development. This is manifested in an inability to solve abstract problems. As a consequence of this setback, students do not have the ability to engage in the Piagetian defined activities of “accommodation” and “assimilation”. Hence, there is an inflexible reliance on existing mental frames underpinned by knowledge that is concretely acquired. In order to identify students that have been caught up in the operational stage of human development, Davis’s Frame Theory is used as a basis to acquire an insight into the process of human cognition. A significant observation

[5] Also identified in the study by Naidoo and Jugoo(2005)

as a consequence of this theory was that errors made by students, classified as operational thinkers, are repeated with little discernable improvement in performance levels. An interesting source of inquiry emanating from this literature review is the investigation of a possible relationship between the errors made by students in programming tasks and their level of cognitive development. It is hoped that such an inquiry will help provide pedagogical insight into how to obviate learning difficulties experienced by students who are caught up in the concrete operational stage of intellectual development. The Frame Theory concept proposed by Davis provides a meaningful context that can be used as a source of reference to explain the process of human cognition.

The main achievements of the literature review can be summarised as follows:

- Elimination of factors suspected to cause poor performance in programming
- Identification of mathematical/problem solving ability as a significant contributor to success in computer programming
- The identification of cognitive learning styles according to the deep and surface classification as a possible framework to predict performance in computer programming
- Deduction that mathematical ability is a manifestation of general problem solving ability which is in turn related to the level of cognitive development
- The identification of the possibility of a relationship between the level of cognitive development, according to the Piagetian theory for intellectual development, and performance in computer programming
- The identification of Davis's Frame Theory as a cognitive source of reference for the development of theoretical assessment frameworks for computer programming
- Error analysis as a possible framework for analysis of performance in computer programming

Chapter Three

3. THEORETICAL FRAMEWORK FOR PROGRAMMING ASSESSMENT

One of the sub-objectives of the current study is the development of a theoretical quality framework for the assessment of performance in computer programming. The focus of the current chapter is the development and demonstration of this framework. The framework will be demonstrated by applying it to generic programming concepts as well as to typical programming problems that students are expected to solve in an introductory programming course.

3.1 Introduction

The development of this framework is as a consequence of the significance attached to error analysis by the cognitive science fraternity (Bransford *et al*. (1986); Davis (1984)) as a basis for unveiling flaws in the cognitive structures of learners. Naidoo (1996) adapted a generic error analysis framework proposed by Donaldson (1963) in order to establish misconceptions in the understanding of mathematics in an introductory university course. This adaptation (subsequently referred to as the Donaldson/Naidoo framework) shall be extended to include the discipline of computer programming. The emanating framework will be underpinned from a cognitive science perspective by Davis's Frame Theory (presented in section 2.3.5.2). The rationale behind this strategy is twofold. Firstly, it heralds a pioneering, concurrent incursion into the fields of human cognition, computer programming and error analysis. Secondly, it helps create the infrastructure for the data collection and analysis phase for the current study.

3.2 The Error Analysis Framework for Computer Programming

Computer programming is characterised by the core elements of data types, data operators, variables and constants, input/output control, sequential, iterative and conditional logic. This list can be extended to include data structures as well as the introduction of abstractionism via modularised programming. These

extensions will bring into play concepts such as parameter/message passing as well as local/private and global/public variables. It is pertinent to take cognisance of the assertion made by Farrell (2007, p. 27) that mastery of these concepts is essential irrespective of which development paradigm is being used. The software development paradigms that pre-dominate currently within the confines of programming instruction are the classical/structured and object-oriented (OO) paradigms. In terms of the technology of programming, these core elements have become the source of three types of errors listed as syntax errors (lack of adherence to language constraints), semantic error (unintelligible meaning) or logic error (flaw that causes an algorithm to do something other than what it was supposed to do). The syntactic and semantic errors are software recognisable. According to Schackleford (1998, p. 301) novice programmers spend a high proportion of their development effort on debugging code on a trial and error basis, thereby solving the syntactic and semantic dilemma that may prevail. This kind of intervention will help in bypassing errors detected by the compiler. However, the students may still be oblivious of conceptual flaws that may be inherent in their understanding of programming strategy. The Donaldson/Naidoo framework was constructed on the philosophy that errors made by students in the solving of mathematical problems could be a pointer to conceptual problems inherent in the teaching/learning process. The invocation of this framework from a computer programming perspective is aimed at highlighting conceptual flaws that may impede meaningful understanding of computer programming. An overview of the framework consisting of three error categories is presented in increasing order of severity. In order to provide an illustrative dimension, the task of switching the value of two variables, referred to as “A” and “B” is used.

- Arbitrary Error – subject behaves arbitrarily and failed to take into account the constraints laid down. A programming related adaptation of this would be the identification of low level errors such as syntactically driven errors or inappropriate use of syntactic constructs that would impede successful implementation of the solution – described by Schneiderman & Mayer

(1979) as "…precise, detailed and arbitrary knowledge about how the constructs in a particular language must be implemented". A solution such as: *C=A AND A=B AND B=C* will be classified as an instance of arbitrary error since it is clearly evident that the solution strategy is correct, but an unintended erroneous invocation of the "AND" operator in an incorrect context will result in a flawed implementation.

- Executive Error – subject failed to carry out manipulations though the principles involved may have been understood. A programming related adaptation of this would be manifest in solutions that may generate run-time errors or the generation of incorrect output due to semantic incorrectness. Schneiderman & Mayer (1979) described semantic knowledge to include low level concepts such as understanding the concept of assigning a value to a variable, summing the contents of an array, and understanding of high level concepts such as searching and sorting of data values. A solution such as: *A=C; C=B; B=A* will be classified as an instance of executive error. The solution strategy is correct i.e. invocation of a "temporary" variable to be used as a conduit to facilitate the switching of variable values. However the understanding of the concept of variable assignment is seriously flawed, since the variable "C" does not have an initial value and some compilers may generate a run-time error. However, many compilers may permit such an assignment statement, in which case there will be no compiler error but an error in output.

- Structural Errors – subject failed to appreciate the relationship involved in the problem or to grasp some principle essential to the solution. A programming related adaptation would be the inability to comprehend the problem domain or the inability to create a programming model to represent the problem domain. A solution such as *A=B* or *B=A* will be classified as an instance of a structural error since the solution strategy or

programming model is completely flawed. The underlying principle of data storage, use of variables and the relationship between different variables in a computer programming context are not understood. The structural error is regarded as the most serious type of error committed.

According to the Donaldson/Naidoo framework, if an error is given a dual classification, then the higher order error would take precedence. As an example, if an error is classified as executive as well as structural, then the overall classification for that error will be structural. The subsequent sections in this chapter focus on presenting a detailed illustration of the error analysis framework for computer programming.

3.3 Illustration of the Error Analysis Framework for Computer Programming

In order to present a more detailed demonstration of the error analysis framework, a generic discussion of the possibility of errors in some of the core elements of computer programming is now undertaken. This discussion attempts to establish a framework for error analysis that incorporates the dictates of cognitive science embodied by the Davis's Frame Theory, in conjunction with the Donaldson/Naidoo error analysis framework used for mathematics. This multi-dimensional approach will be illustrated diagrammatically together with a narrative to facilitate understanding.

3.3.1 Input/Output Control

The first demonstration entails an illustration using a trivial input/output sequence. A typical input/output sequence may resemble the flow diagram shown in Figure 1.

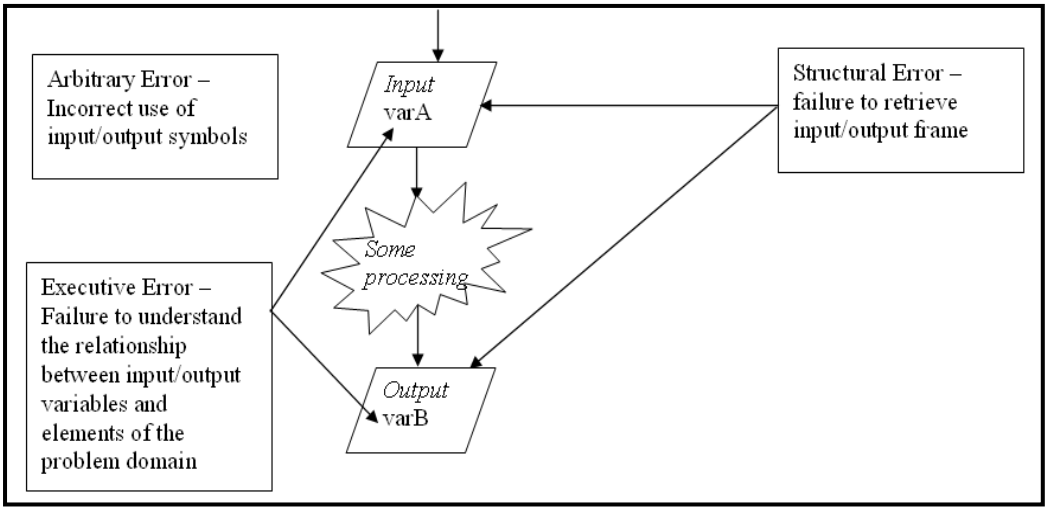


Figure 1: Error Framework for Input/Output

It should be noted from Figure 1 that the possibility of each error type occurring is flagged together with a reference to the knowledge frames (in reference to Davis's Frame Theory) that need to be invoked at that juncture. If there is a failure to recognise the need for the input of data, then the implication is that an incorrect/incomplete frame is being retrieved from passive memory in response to the problem domain. This is classified as a structural error. However, if the data input frame is invoked, but there is perhaps some confusion between input and output or the incorrect data item is input, then this is classified as an arbitrary error. The rationale here is that there was an understanding of the need to provide an interface for data input/output, as well as the relevance of providing a receiving variable, but this was not accurately implemented. If the student[6] realised the need for input/output, but did not appreciate the need to include a variable as a data receiver, then this implies that the student failed to understand the relationship between problem domain elements and the use of variables to model the problem domain. This is classified as an executive error.

[6] The refernce to "student" is in a fictional sense and alludes to the possible originator of the solution being analysed

3.3.2 Looping/Iteration

Shackleford and Badre (1993) observed that "…iteration is among the most basic and frequently applied concepts in the programmer's repertoire". They go on to assert that students do not have adequate understanding of the concept of iteration to be able to apply it effectively. This assertion is corroborated by their observation that less than half of the subjects in an introductory programming course were able to provide correct solutions to a simple averaging problem. The error analysis framework will now be presented for a similar task. This task is extended from a demonstration perspective to include the requirement that a count of the number elements that are above average is also computed. This task requires the invocation of looping, conditional and array processing constructs. In order to achieve an efficient solution for this problem, it is imperative to make use of an array data structure. From a cognitive perspective, retrieval of a frame that does not incorporate an array structure is tantamount to a structural error. In order to respond to the specifics of the problem situation, accommodation[7] has to occur, but around the context created by the array processing frame retrieved from passive memory. A solution that does not invoke array processing is possible, but highly improbable. Hence frame retrieval will involve invocation of processing that requires instantiation of (1) an array processing frame as well as (2) an iteration control frame (3), input/output frame and (4) a conditional control frame. These frames need to be integrated in a logical, coherent solution referred to by Davis (1968, p.56) as a "collage". The mental activity that has to take place before that successful instantiation of the "collage" is shown in Figure 2.

[7] Eeach of the constructs (looping, condiitonals and iteration) has to be invoked in a non-routine manner, thereby necessitating the constuction of a new mental structure (referred to as accommodation from a Piagetian perspective).

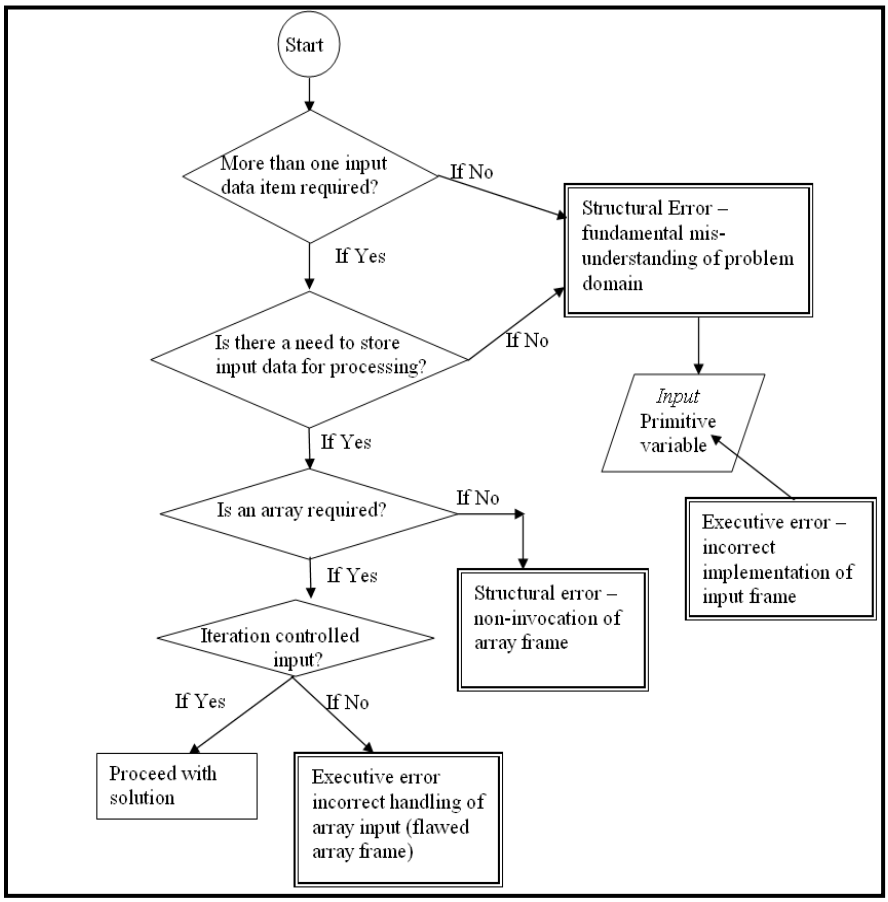


Figure 2: Mind map for array processing

Figure 2 is an illustration of the mental activity required in ascertaining the need to invoke array processing. Once this has been established, it is within this context that the other frames are invoked in achieving a solution to the problem. The cognitive interactivity that occurs at this stage is at a very high level and correct processing here is paramount to ensuring success at lower levels of detail. The remaining phases of the solution process are now presented in two parts. The first part is a replication of the task used by Shackleford and Badre (1993) in their study and the second part entails the extension to the task, warranted from a demonstration perspective for the current study. As a precursor

to the emanating discussion, the flowchart shown in Figure 3 provides an overview of a possible solution path.

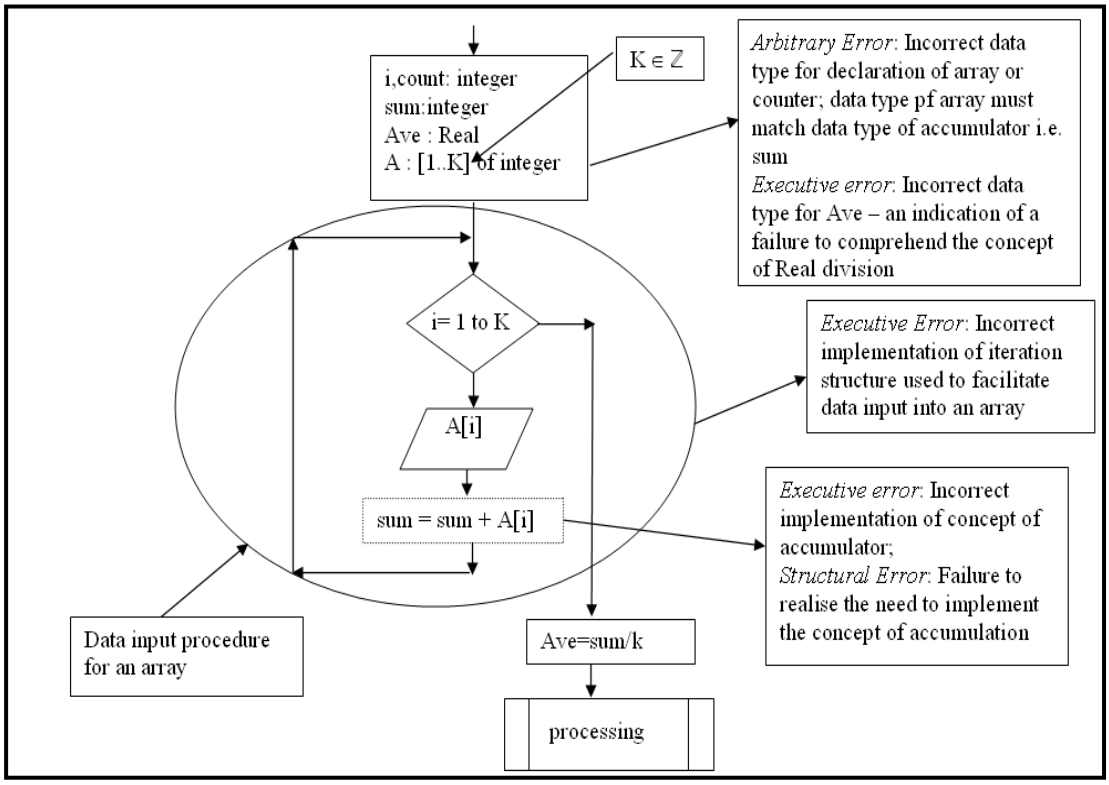


Figure 3: Error framework for computation of average

Once the recognition of array processing has been achieved, invocation of programming structures necessary for enabling processing of array elements are now necessary. The input of elements into an array has to be iteration controlled. Incorrect transfer from the problem domain to the solution structure will result in arbitrary errors. An example of such an error would be the failure to take cognisance of the specified input data type. Another example would be the failure to provide correct loop control de-limiters. The array dimensions and loop controlled de-limiters are “slots” or frame variables that are instantiated via the input data. A failure to correctly implement the iteration structure is classified as an executive error. In order to facilitate further processing and achieve a solution for the problem, it is essential to now invoke a procedure that can be directly used (assimilation) or modified for the problem domain (accommodation). Further

processing will entail invocation of a frame to handle conditional logic (branching) as well as iteration control. Failure to recognise the need to invoke conditional processing or iteration control to determine a count of input data will qualify as a structural error. A flowchart (shown in Figure 4) is used to illustrate the logic required to compute a count of the number of input items that are above average.

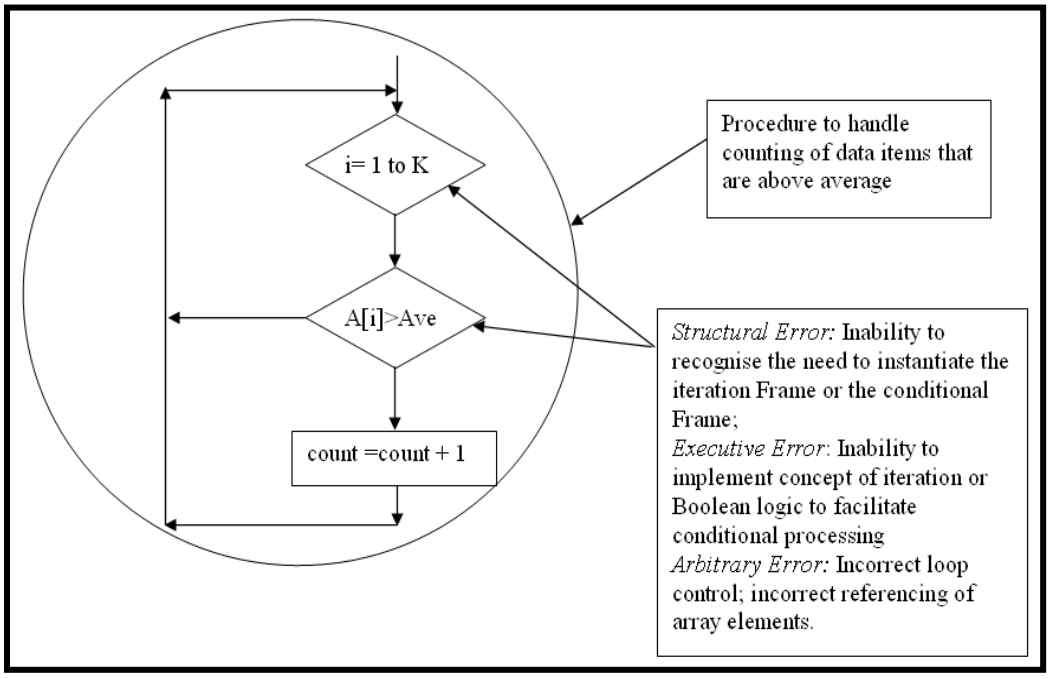


Figure 4: Error framework for "above average" count

As explained in the comments in Figure 4, the cognitive processes that are activated to facilitate problem solving use passive memory for frame and process retrieval. These frames are instantiated in active memory by replacing the variable slots with data that is obtained from the problem domain. In terms of iteration and conditional control, the incorrect choice of values to populate the frame variable slots results in arbitrary errors. However, incorrect mapping of the problem domain manifested in the form of incorrect choice of looping structures or incorrect conditions used to control the flow of processing is classified as an executive error.

3.3.3 Sub-routines/Procedures – Modular Programming

One of the basic tenets of good programming technique is the requirement that programs be developed using a modular approach in order to raise the level of abstraction. This approach facilitates easier reading, writing and debugging of programs (Shackleford, 1998, p. 89). These sentiments are further entrenched by Schach (2007, p. 196) when he refers to procedural abstraction as "… a powerful software development technique that contributes to increasing the level of portability of the software product". However, this procedural abstractionism coupled with the general level of abstract thinking required in programming increases the cognitive burden placed on programming students. Hence, from a pedagogical perspective it is only natural that this modular approach be implemented at a post-novice level of programming instruction. However, the application of the error analysis framework at this stage is still warranted. This becomes apparent when solutions to tasks requiring an understanding and implementation of modular programming is analysed. The failure to appreciate or understand the applicability of a solution that comprises of modular code is classified as a structural error. However, this structural error is not reflective of poor problem solving capacity, but rather an inability to present a solution that conforms to the dictates of the software engineering discipline. This is evidenced by the observation of programmers who prefer to deliver monolithic code rather than make use of local variables and formal parameters, thereby ensuring reusability and portability of code artefacts. At this level of programming, the non-retrieval/implementation of a cognitive frame that contextualises the requirement for a modular approach is classified as a structural error. The rationale here is that the transfer from the problem domain to the programming environment is inadequate in establishing an efficient model for the solution. In order to explain instances of executive errors in the domain of modular programming, it is imperative to explain the concept of formal parameters. Formal parameters are characterised by one of three distinct semantic modes: (1) They can receive data from the corresponding actual parameter; (2) they can transmit data to the actual parameter; (3) they can do both. These three semantic modes are called the "in

mode", "out mode" and the "inout mode" (Sebesta, 2006, p. 387). In terms of the error analysis framework, a failure to understand the relationship between these different types of parameters is classified as an executive error. Incorrect manipulation of the parameters in conjunction with the local variables in order to correctly represent the intended logic of the sub-routine would also be classified as executive errors. Arbitrary errors would include a failure to correctly match the data types from the parameter declaration to the sub-routine activation. Further instances of arbitrary errors will be a failure to comply with any data constraints placed on the parameters.

Figure 5 provides a generic illustration of the error framework in reference to procedural/modular processing.

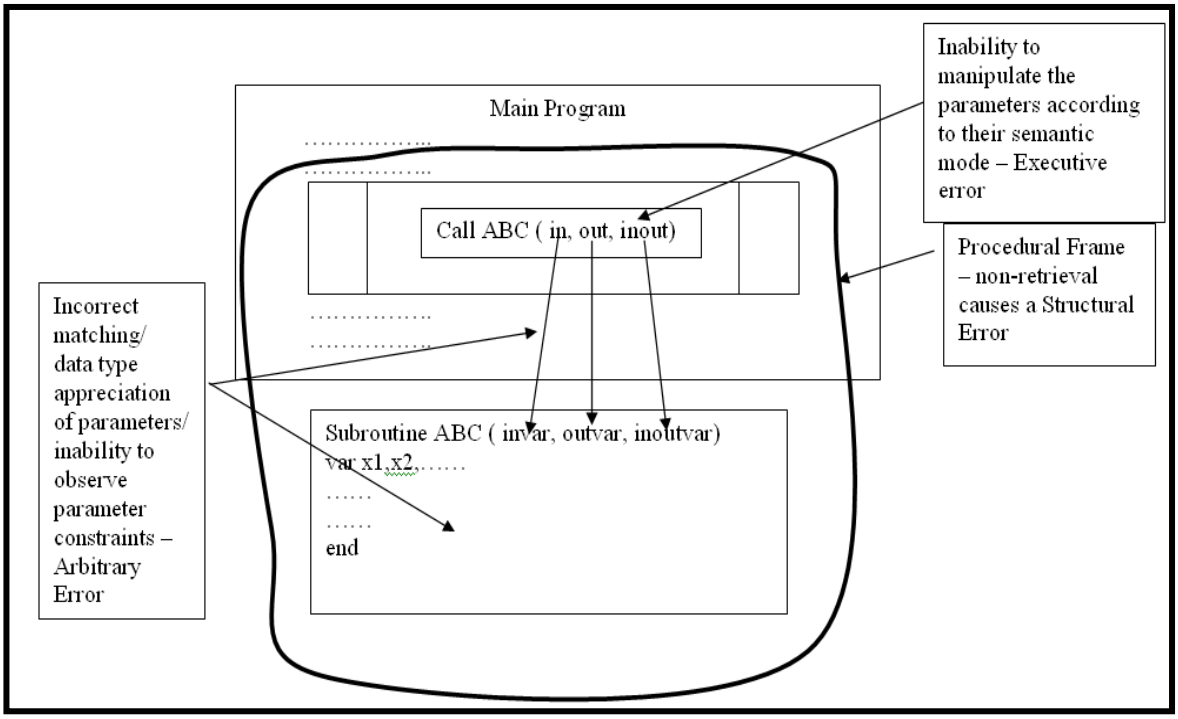


Figure 5: Error framework for procedural processing

3.3.4 Flowchart Implementation Error Analysis

"*Computer science allows us to see problems as a different kind of puzzle. Rather than defining problems as a puzzle of physical matter and physical forces, the computing perspective lets us see problems in terms of information and communication*" (Shackleford, 1998, p. 18).

Life is characterised by symbolism. An example in point is the use of letters of the alphabet to facilitate communication. Herein we have an abstract symbol system that is used to "piece" together solutions to problems that constitute the essence of life. The analogy can be drawn to programmers using flowchart symbols to "piece" together solutions to programming problems. The abstractionism provided by flowcharting ensures priority being accorded to the logic of problem solving rather than the syntactical or technical requirements of implementation detail. However, from a functional perspective, the success of the solution is measured according to the accuracy of the output. Hence, in order to present a "complete package", the programmer will have to provide code that abides by the syntactic convention of the chosen programming language. The coding phase also illuminates conceptual misunderstandings that may prevail in the programmer's interpretation of programming structures. These misconceptions emanate from the interpretation of symbolic notation representing input/output, sequential and procedural activity, conditional and iteration logic. The inability to transfer the semantics inherent in the flowchart/graphical representation of the solution into physical code artefacts will generate a series of structural, executive and arbitrary errors as is illustrated in flowchart fragment shown in Figure 6.

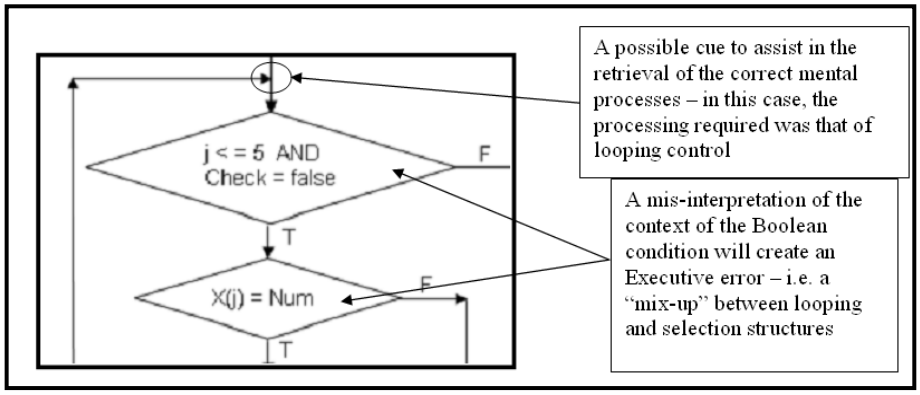


Figure 6: Error framework for flowcharting

The recognition of the conditional structure will prompt the retrieval of a mental frame that handles conditional processing. However, in this case, the conditional structure is applied in the context of establishing control over an iteration structure. A failure to recognise the mode of implementation of the control structure implies that the implementation of the control and iteration structures is flawed. Hence, in terms of the definitions in the error analysis framework, this will qualify as an executive error. However, in terms of the overall logic of the flowchart fragment, a failure to recognise the iteration structure is a serious misconception indicative of an error that goes beyond the mechanics of implementation. This situation now also qualifies to be a structural error. The conflict is resolved by attaching the higher order error classification, which in this case will be a structural error.

3.4 Application of the Error Analysis Framework

A component of the data collection phase for the current study entailed an analysis of programming solutions provided by students in a computer programming examination. An explanation of the application of the error analysis framework is undertaken in conjunction with sample questions from the programming examination (Appendix C).

3.4.1 Flowchart Implementation

In this assessment task, students were required to provide the code for a flowchart diagram. The flowchart (Figure 7) is presented with an in-line narrative, explaining the implementation of the error analysis framework.

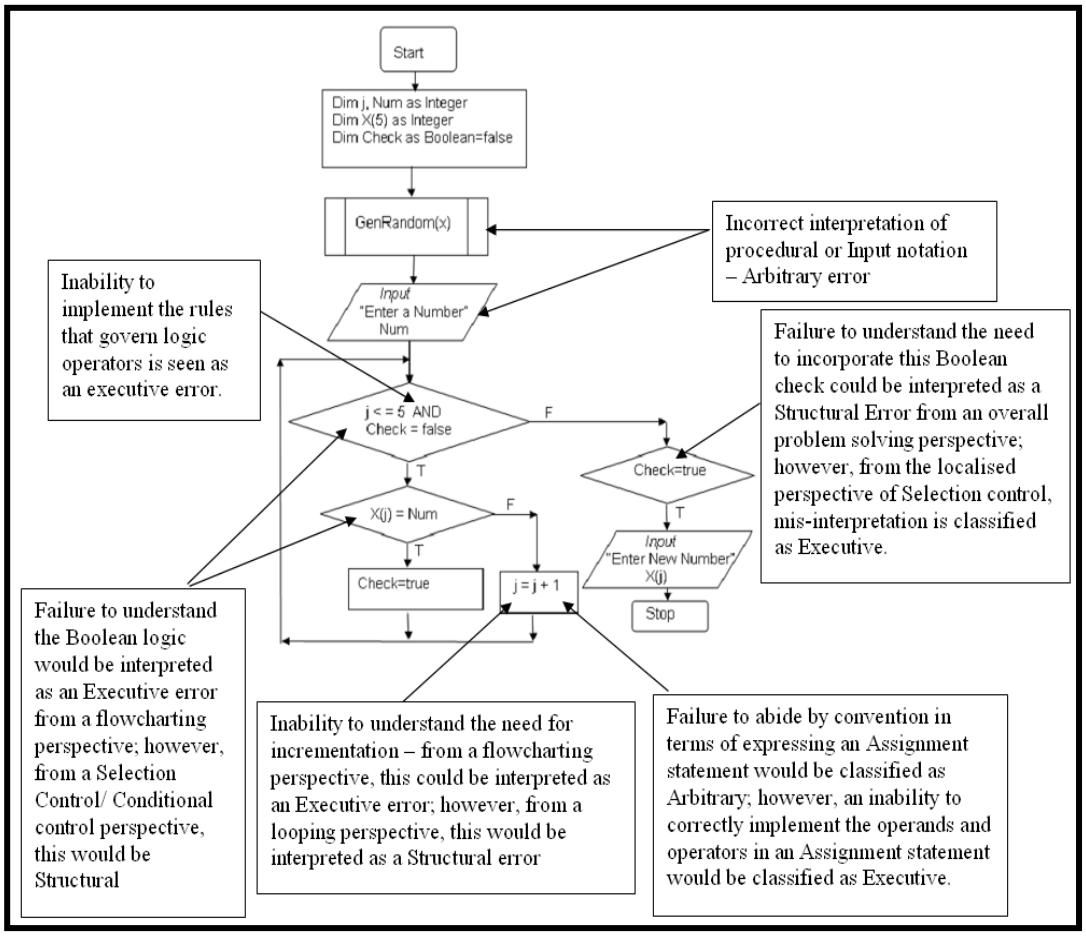


Figure 7: Error framework applied to flowchart from the data collection instrument

Flowchart implementation includes the ability to implement and manipulate flag variables[8] to control processing, logic operators to control conditional structures and incrementation and summation (accumulator variables). The inability to apply these concepts to coding from a semantic perspective is classified as a structural error. Hence, in the situation depicted in the flowchart shown in Figure 7, a flag variable and a counter are used to control loop termination. An inability to understand the logic inherent in this mechanism will tantamount to a structural error from a coding perspective. An instantiation of an executive error will be an inability to apply boolean logic in ensuring the correct implementation of the iteration structure. In an assessment of this nature, the invocation of arbitrary errors will entail simple syntactical imprecision.

[8] In reference to boolean variables used as a mechanism to control iteration

3.4.2 Data Structure Error Analysis

An implicit requirement of user defined data structures is that the student reduces the abstraction involved by creating a mental or physical image of the newly - defined structure. In order to accomplish this, the student needs to have existing knowledge and mental representations of the primitive data types and structures that are used in the construction of the user defined data structure. In order to establish a context for the subsequent discussion, the user defined data structure that students were required to comprehend is presented in Figure 8. The implementation language is Visual Basic.

```
Structure Sales
        Dim Name As String
        Dim TotalSales As Integer
End Structure
Dim Employees(9) As Sales
```

Figure 8: User defined data structure used for error analysis

To reduce the abstractionism inherent in this declaration, students will have to create a mental image that firstly consisted of primitive data structures. A subsequent synthesis is required to concretise the composite user defined structure. The processing required for comprehension of the user defined data structure shown in Figure 9 that follows.

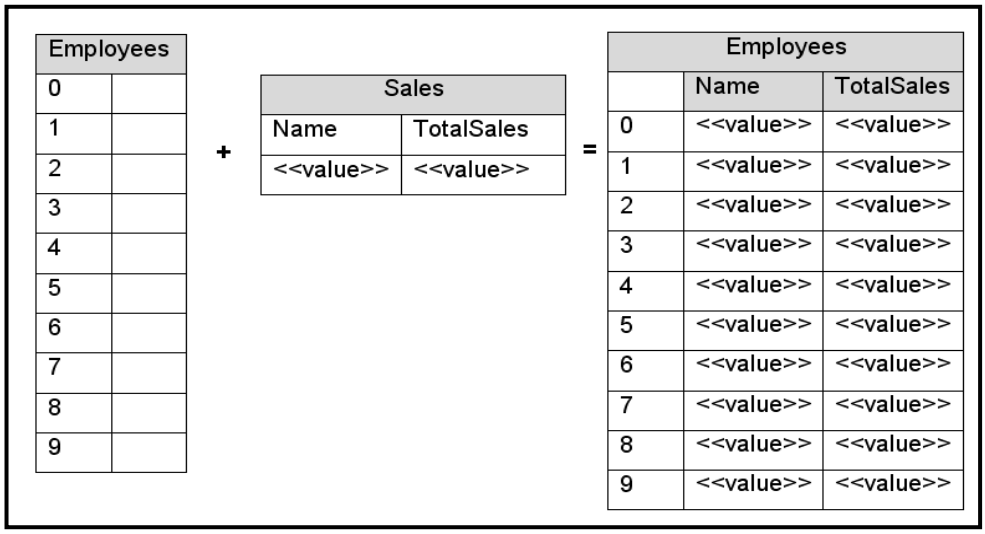


Figure 9: Mental image of user defined data structure

In order to achieve this synthesis, the student will be required to retrieve frames (assimilation) that handled arrays (Employees Array shown in Figure 9) and records (Sales Record shown in Figure 9). A process of synthesis should follow where these frames are used in the construction of a new frame (accommodation) that becomes the basis of all further manipulation. The inability to achieve assimilation and retrieve the correct base frame is instantiated as a structural error. This will render all subsequent attempts at solving the problem as futile. If assimilation has been achieved, then subsequent processing will entail a synthesis incorporating the inner workings of the array frame with the record structure frame. The student is now engaging in the process of accommodation (adapting an old frame to create a new object). A failure to successfully achieve accommodation also becomes an instance of a structural error. A question of this nature can be classified as a high level question due to its propensity for generating structural errors. The activity of frame synthesis is a major cognitive challenge that could impede a successful response to a question requiring an integration of primitive data structures. However, once progress has been made beyond the stage of frame synthesis, there is now a possibility of the instantiation of executive errors.

3.5 Final Word on Error Analysis Framework

This chapter presented the error analysis framework for computer programming. The framework has been presented in reference to generic computer programming as well as in relation to one of the data collection instruments (ISTN212 computer programming examination) used in the current study. In recognition of the significance of cognitive science theory as presented in the literature review, Davis's Frame Theory has been used to underpin the classification of errors according to the different error categories that constitute the error analysis framework. This framework will be implemented in the current study to classify errors made by students in a computer programming examination. It is hoped that this error classification exercise will provide an insight into students' understanding of computer programming.

Chapter Four

4. RESEARCH METHODOLGY

This chapter focuses on the research methodology as well as a reference to the academic underpinnings of the adopted methodology. It is imperative that such a discussion be undertaken in order to add credibility to instruments used in the data collection phase of the research process as well as the adopted research methodology.

4.1 RESEARCH PARADIGM

This study can be classified as primarily qualitative. However, quantitative analyses are seen as pivotal in establishing relationships and adding credibility to any conclusions emanating from the research effort. This strategy is commensurate with the opinions of researchers (Strauss and Corbin (1990); Lubbe and Kloppe (2004, p. 77); Berglund *et al*. (2006)) that qualitative and quantitative methods can be effectively combined in the same research project. The overriding message being delivered is that qualitative research is used to discover themes and relationships, while quantitative techniques are used to validate these ideas. This assertion became a catalyst for the design strategy of the current study. The combination of qualitative and quantitative methodologies is used to underpin the research agenda. However, according to Lubbe and Kloppe (2004; p. 74) research "… has its historical origins in science" and the scientific method is essentially one that embraces a quantitative demeanour. Hence in order to add credibility to the adopted methodology, it is imperative to establish the relevance of the qualitative paradigm to the learning of computer programming. According to Murnane (1993), programming has traditionally been considered a 'scientific' activity. Hence, a natural consequence should be that research in the field would have strong inclinations towards positivism. However, Murnane emphasises that:

"*…no studies have shown that students who perform well in the traditional sciences have any particular advantages when it comes to programming….the development of a computer language may be a scientific process, but the authoring of a program written in that language is not*" (1993, p. 216)

This observation has paved the way for programming related inquisitions that can be classified as a form of post-modernistic research. This embodies a research ethos that accommodates the mixing of diverse ideas and methodologies. It is within this framework that various strains of the qualitative research paradigm, such as phenomenography, are beginning to gain acceptance in the literary domain.

4.1.1 QUALITATIVE RESEARCH

According to Strauss and Corbin (1990, p. 17), qualitative research embodies a demeanour that is not fully dependent on statistical procedures and other means of quantification. Qualitative methods are used to better understand phenomenon about which little is already known or to gain in-depth information that may be difficult to convey quantitatively. Hoepfl (1997) presented a summary of the characteristics of qualitative research on the basis of input derived from various literary sources. These included the following key aspects:

- The researcher acts as the "human instrument" of data collection
- Qualitative researchers use predominantly inductive data analysis
- The research report is descriptive, incorporating expressive language and is classified as interpretive
- The research design is emergent as opposed to pre-determined

A specific strain of qualitative research is referred to as phenomenography. Phenomenographic research was given prominence by Booth (2001) in her

inquisitions on the learning of computer programming. According to Eckerdal, Thune and Berglund (2005), phenomenography is an empirical, qualitative research approach where "… the object of interest is how a certain phenomenon is experienced by a certain group of people". The outcome of phenomenographic research is to gain knowledge of the ways in which learners come to grips with the concepts and principles of a domain in order to develop understanding and mastery of a subject. The dominant sampling strategy is purposeful sampling. The objective here is to seek information-rich cases that can be studied in depth. This is an embodiment of an approach that prioritises the depth that can be ascertained from the data source at the expense of the volume of data. It is reported in Hoepfl (1997) that interviews are the primary strategy for data collection or it can be used in conjunction with document analysis and other techniques. Data recording is conducted via tape recording which is formally documented using a process of textual transcription.

4.2 RESEARCH STRATEGY

In accordance with these sentiments on qualitative research, this study used the following data collection instruments that are presented from an overview perspective (a detailed discussion follows in Chapter 5):

- A semi-structured interview that aimed to elicit general learning styles as well as to ascertain levels of understanding of programming related concepts. The deep and surface framework was used to classify responses. The focus areas of the interview were on general understanding of programming concepts and specific programming related problem solving. The intended outcome of this phase of data collection was on acquiring an understanding of how students experienced the phenomenon of computer programming. Hence, phenomenography now becomes a valid context for this dimension of the research.

- A cognitive ability test to determine the general level of cognitive maturity according to the Piagetian framework for cognitive development. This data collection effort is classified as part of the quantitative framework and random sampling was used as the data collection methodology.

- Analysis of student performance in a programming examination using the error analysis framework developed in Chapter Three of the current study. This document analysis effort on the basis of grounded theory is also classified as qualitative research.

The convergence that will be achieved in each of these data collection efforts is underpinned by the knowledge that there have been studies that engaged in a macroscopic investigation of the relationship between students' overall performance in programming and their general level of cognitive ability. However, the current research effort purports to intensify these investigative efforts by engaging in a deeper inquiry. The literary inquest is also instrumental in entrenching this probe. The error analysis framework is implemented in order to facilitate a microscopic investigation of students' response to programming tasks. This is accomplished by analysing programming statements in isolation to establish "executive" type misconceptions as well as holistically to establish structural flaws in understanding.

4.3 THE PILOT STUDY

The instruments used in this study were piloted with a group of seven students from the Pietermaritzburg campus of UKZN. This exploratory mission succeeding in refining and obviating interview questions that were either ambiguous or that did not generate an insightful response. It was also pivotal in enhancing the quality of the questions for the cognitive ability test. This ensured ease of understanding, thereby minimising the prospect of queries from subjects who were engaged in writing the test during the main data collection phase.

4.4 ACADEMIC FRAMEWORK FOR DATA COLLECTION

The focus here is to attach academic credibility to the design of the interview questionnaire (Appendix A) as well as the design of the cognitive ability test (Appendix B). The third data collection instrument (Appendix C) has been validated by the error analysis framework presented in Chapter Three.

4.4.1 The Semi-structured Interview

According to Lubbe and Kloppe, semi-structured interviews "… involve the preparation of an interview guide that lists a pre-determined set of questions or issues that are to be explored during the interview" (2004, p. 123). Although the instrument guiding the interview conformed to these requirements, the strategy used here was similar to that employed by Bruce *et al*. (2003) where the questions served as a 'trigger' from which further question were asked in order to illuminate student understanding of programming concepts. The questions in the interview were prepared in conjunction with literary sources and adapted in accordance with students' programming experience at this juncture. The questions included in the interview design can be demarcated into those that focus on general understanding of programming concepts and those that focus on program related problem solving.

4.4.1.1 General Understanding of programming related concepts

The focus here is to attach some literary credibility to the questions pertaining to the understanding of programming concepts in the interview design. The questions included:

- A critique on the validity of the logic inherent in the use of a variable for increment purposes (i.e. x=x+1). The assignment and incrementation of a variable presented in a concurrent arrangement would be expected to pose a significant cognitive burden on the student from a comprehension perspective. Dehnadi (2006) identifies the assignment statement as one of the major semantic hurdles that programmers have to overcome. This question attempts to unearth the mental model created by the student that is necessary to obviate the high degree of abstractionism inherent in the programming statement. This enquiry is extended to ascertain whether students are able to recognise the absurdity of the statement from a mathematical perspective.

- A critique on the need for variable declarations together with an associated data type. The rationale here is to ascertain a student's level of critical judgment. This can be analysed in the light of the knowledge that the syntactical rules of programming languages have typically been presented in a prescriptive manner. Hence, the default response expected would be that strict obedience to the prescribed rules of a language is a necessity. It was hoped that deeper knowledge and higher levels of confidence in computer programming would become apparent by the quality of the answer to this question. Warren (2001) argues that data types do not conform to the way people think and the type of a variable should be recognised via the context that it exists in rather than being prescribed via a declaration. He goes on to suggest that type declarations "…seems an unnecessary addition of no algorithmic utility and is an example of gratuitous complexity". It was envisaged that a simple inquisition of this nature would become a catalyst for a discussion that will add insight into students' understanding of fundamental programming concepts.

- An analysis of the ability to extract meaning from the symbolic representation inherent in a flowchart. It is reported that flowcharts are better than actual code in highlighting the control-flow of conditional logic (Schneiderman (2006);

Scanlan (1989)). In keeping with this assertion, the questionnaire included a task on flowchart analysis. This task tested a student's ability to apply knowledge from the analysis of the flowchart and determine the control flow for a selection structure. As an extension of this logic comprehension exercise, the student was required to identify a looping structure that was commensurate with the logic of the flowchart. A successful response to this task will require the invocation of comprehension and analysis abilities that are reflective of a deep learning approach.

- An investigation of the ability to invoke inductive reasoning and generate implicit knowledge by making generalisations after having prior exposure to specific cases within the problem domain. This aspect of reasoning was included on the basis of a study by Mancy (2006). In this study, Mancy used semi-structured interviews to elicit responses from her students about computer programming aspects that were not explicitly taught. Here, it was found that students who adopted a deep learning approach to programming were able to respond by rationally constructing generalisations that helped in the answering of questions on specific programming related issues. The study by Mancy (2006) incorporated a question on the decision of when to use an array. This question was also used as part of the current study. In terms of the context of the current research, students were exposed to examples that dictated the use of arrays, thereby acquiring explicit knowledge about array manipulation. However, knowledge about the applicability of arrays was not explicitly taught. Hence, in the interview session, students were required to interrogate their implicit knowledge and engage in the inductive exercise of making a generalization on the applicability of an array. The accuracy of the generalisation was used as a measure that contributed to the overall classification of the student according to the deep and surface framework.

4.4.1.2 Programming Related Problem Solving

The rationale behind this component of the study was to allow students to verbalise their thoughts while solving a programming related problem. According to Owen, Budgen & Brerreton (2006), "… the resulting stream of utterances helps indicate the way a subject is reasoning about how to perform the task". This strategy is combined with the strategy used by Tukiainen and Monkkonen (2002) on students doing a Business Information Degree. This study aimed to measure students' ability to solve programming related problems using the IBM Programmer Aptitude Test (PAT). The PAT was widely used in industry from 1966 onwards in order to ascertain the competence levels of prospective employees in the discipline of computer programming. However, according to Mayer & Stalnaker (1968), there was no significant correlation between this test and actual programming performance. Wolfe (1971) developed a series of tests to overcome these limitations, but the proliferation of computing technologies has impacted on programming strategy and techniques, thereby rendering the reliability/viability of these tests to be era dependent. An adaptation of the test used by Tukianen and Monkkonen is used in the current study as a basis for the problem solving component. The adaptation entailed the incorporation of two further tests. The first of these was a test of the logical realm, manifested in the form of a task to switch the value of two variables. In a pilot study[9] conducted as a pre-cursor to the current research effort, it was observed that students exhibited an innate, reflexive response to this question. This observation could be attributed to the fact that this task was similar to a class exercise. In order to test the same logical ability, a similar task was assigned, but the dimensions involved were extended to three variables. The second test required the students to find a specified name in a phone book and then describe the process that they had used to find the name. In this task, students were required to make use of the terms random, sequential and index-sequential to describe their search strategy. The frame of reference for this task is provided by a multi-national study

[9] Described in Section 4.3. of Chapter 4

conducted in Australia by Simons *et al*. (2006) to determine the possibility of a relationship between programming aptitude and cognitive ability. One of the measures of cognitive ability entailed a test of the ability to articulate a strategy for conducting a search. It was reported in this study that "…anecdotal evidence suggests that programmers are better than non-programmers at describing search processes". In a previous study (Onorato & Schvaneveldt cited in Simon *et al*. (2006)), it was found that subjects who were more comfortable with computer programming were more likely to incorporate programming constructs in their descriptions. Terms such as "sequential" and "random", typically used to express algorithmic thought, will be used in their solution strategy. Simon *et al*. (2006) confirmed this relationship and concluded that "… students who carry on to be successful programmers tend to have pre-existing strengths in strategic/algorithmic style of articulation". A significant rationale for the content in the questionnaires and the interview session for the current study, emanated from an academic framework that attached much significance to one's ability to articulate problem solving strategy in a logical, coherent manner.

4.4.2 The Cognitive Ability Test

Previous research efforts have used programming aptitude tests such as the PAT or the Aptitude Assessment Battery Programming (e.g. Tukiainen & Monkkonen (2002); Wolfe (1971); Mayer, Dyck & Vilberg (1986)) or mathematics proficiency levels (e.g. Konvalina (1983); Campbell & McCabe (1984); Bennedson & Casperson (2005)) as instruments to predict students' ability to succeed in computer programming courses. The focus of these research efforts has been to establish a relationship between the ability to acquire competence in computer programming and the cognitive ability inherent in the subjects of the study. However, a discernable feature of these cognitive assessment instruments is the lack of a didactic or cognitive development framework to add validity to much of the content. This sentiment is echoed by the comment of a forensic psychologist that educators should ignore the static nature of cognitive ability

testing and "…understand intelligence in terms of mental structures, processes, and operations - a rational basis for intellectual development in education" (Asa cited in Goodlad & Keating, 1990). It may well be difficult to ignore the wisdom inherent in these words. However, there may be significant logistical impediments that may retard the understanding of intelligence as suggested by Asa. A conciliatory compromise would be the invocation of a testing instrument informed by an established cognitive development framework that makes use of mental structures, processes and operations in its construction. The design of the cognitive ability test (Appendix B) for the current study has been constructed in accordance with the Piagetian framework for cognitive development. The test was designed with the purpose of ascertaining the ability to engage in abstract thinking. This is commensurate with the opinion expressed in literary sources that abstract reasoning is a pre-requisite for ensuring competence in programming (e.g. Mayer, Dick & Vilberg (1986); Chmura (1998)). The Piagetian framework for cognitive development uses abstract reasoning ability as a criterion for classifying cognitive development. Hence, the design of the cognitive ability test for the current research has been done in accordance with this framework. According to Piaget, at the formal operational stage, learners begin to develop logical thinking in relation to concrete/physical objects. According to Boeree (1999), a significant test to ascertain whether learners have achieved concrete operational thought is to establish the ability of learners to demonstrate the principle of conservation of area. This task is included as part of the cognitive testing routine for the current study (Appendix B, Question One). The objective is to confirm that students had reached the late concrete stage of intellectual development. In their interpretation of the Piagetian framework for cognitive development, Leongson & Limjap (2003) assert that it is the formal operational stage at which learners are able to deal with ideas and think beyond concrete reality. In a critique of Piaget's theory of intellectual development Wankat & Oreovicz (1993, p. 266) report that many adults remain in the concrete operational stage throughout their lives. Hence the main focus of the cognitive ability test was to ascertain whether students had achieved the requisite

cognitive levels for abstract problem solving by classifying them according to the categories of late concrete, early formal or late formal. The cognitive testing routine had to encompass tasks that could be regarded as instruments that would facilitate quantification of students' cognitive levels according to these categories. In order to achieve this, abstract reasoning tasks would have to be developed. The main criterion would be conformance to the Piagetian model for assessment of formal operational thought. Piaget identified sixteen binary logical operations that are inherent in formal thought. In order to facilitate assessment of mastery of the sixteen logical operations, a Propositional Logic Test (PLT) was developed by the Science Education Faculty at Rutgers University (Piburn, 1989). In the PLT, the sixteen Piagetian logical operations were classified into four logical groupings that can be described as conjunction ("and"), disjunction ("or), implication ("if-then") and bi-conditional ("if and only if" or equivalence). In a more specific classification, Lawson (1983) observed that a precondition to formal operations is to understand bi-conditional reasoning or "if and only if" logic. In terms of procedural programming, this is significant since understanding of bi-conditional reasoning is pivotal in terms of learning with respect to procedural programming (White & Sivitanides, 2002). Burton and Bruhn (2003) argued that the mastery of the basic tenets of procedural programming is vital to ensure success with respect to implementation in an object-oriented environment. This assertion makes the relevance of bi-conditional reasoning independent of the paradigm of development.

It is within this framework of reasoning that an instrument to measure the cognitive maturity of students to handle the rigours of learning to program was assembled. An example of a question that purports to test propositional reasoning adapted from Wason and Johnson-Laird (1972), is presented in Figure 11.

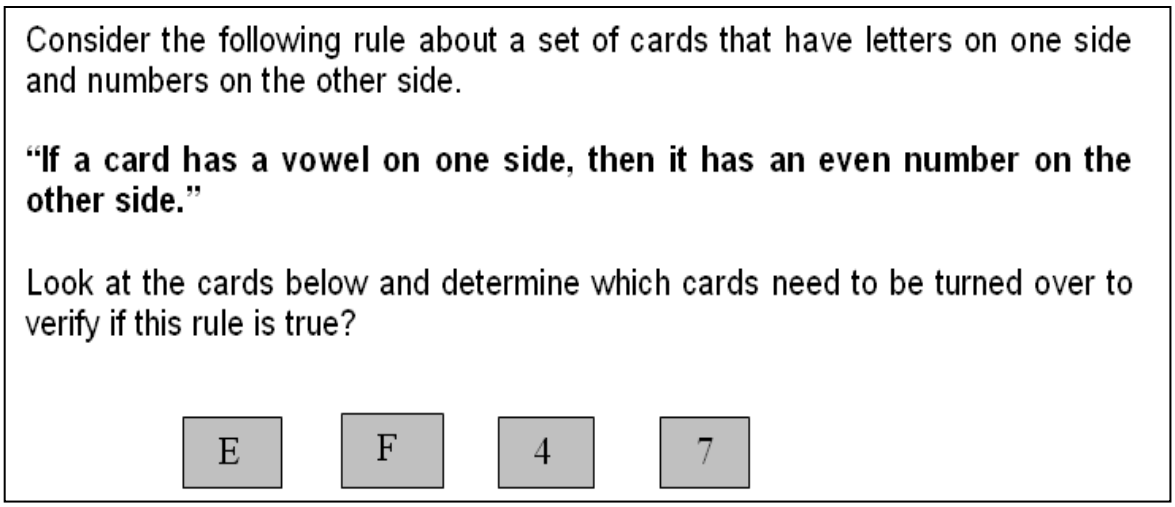


Figure 11: A sample test of propositional logic

In order to ascertain students' cognitive maturity in terms of achieving competence in formal thought, an adaptation of the PLT was used in the current study (Appendix B, questions 8, 9 and 10). However, this test had to be supplemented by assessment tasks that could achieve a more precise measure of the level of formal thought exhibited. In order to achieve this, adaptations of previous work (Kurtz (1980) and Barker & Unger (1983)) were used for the current study. In accordance with the frameworks established in both these studies the following categories of logical assessment were used to classify students into one of three Piagetian intellectual development levels (i.e. early formal, formal and late formal). The cognitive ability test entailed the following assessment areas shown in Table 1.

Table 1: Categories of logical assessment

Reasoning Skill	Intellectual Development Level
Conservation of area	Concrete
Direct Proportion	Early Formal
Probabilistic Reasoning	Early Formal
Inverse Proportion	Formal
Deductive Logic	Formal
Propositional Reasoning	Late Formal
Correlational Reasoning	Late Formal
Permutations	Late Formal

Chapter Five

5. DATA COLLECTION AND ANALYSIS

The previous chapter focused on the establishment of academic credibility for the data collection instruments. This chapter focuses on the collection, presentation and analysis of the data used in the current study.

5.1 OVERALL RESEARCH DESIGN

According to Hoepfl (1997), a qualitative study necessitates the underlying presence of a theoretical framework/protocol to justify the mechanism of data collection. In accordance with this assertion, data collection for the current study was underpinned by theoretical frameworks that are illustrated in Figure 12.

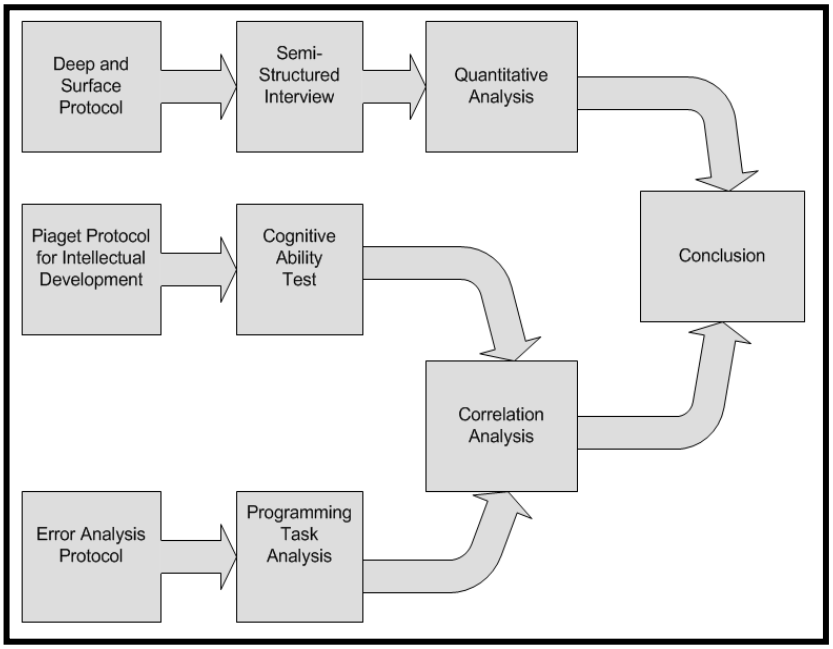


Figure 12: Research Design

The rationale for the design shown in Figure 12 is presented from an overview perspective:

- The strategy of purposive sampling was used to select students for a semi-structured interview. Data analysis was performed using the deep and surface framework to ascertain cognitive styles towards computer programming. The focus in this phase of data collection was to establish whether the cognitive style adopted by students towards computer programming was symptomatic of a "deeper" problem that pointed to a lack of understanding of computer programming fundamentals.
- Students were selected at random to participate in a cognitive ability test. The test was designed with the intention of classifying students according to the Piagetian levels of cognitive development. The focus here was to establish the level of cognitive development displayed by students from the ISTN212 computer programming class.
- An analysis of computer programming examination scripts was undertaken for a subset of students who had participated in the cognitive ability test. Correlation theory was applied to the types of errors made in the examination and the level of cognitive development demonstrated by these students in the cognitive ability test.

5.2 TARGET POPULATION

According to Lind, Marchal & Wathen (2005, p.7), "…a population is the entire set of individuals or objects of interest or the measurements obtained from all individuals or objects of interest". For the current study, the primary target population consisted of students who had completed the ISTN 212 course (Databases and Programming) in the School of Information Systems and Technology (IS&T) at the University of KwaZulu-Natal (UKZN). This course was offered at the Durban and Pietermaritzburg campuses of UKZN and consisted of seven weeks of programming instruction. The target population consisted of 261 students. Students were informed of the voluntary nature of their participation as well the non-obligatory demeanour of the interview questions. A concerted effort

was made to ensure adequate students' understanding of the data collection process. This was accomplished verbally and formally via a consent document (Appendix E - Part Two) that served as a confirmation of voluntary participation.

5.3 OVERALL SAMPLING STRATEGY

In quantitative studies, "…the most widely used type of sampling is a simple random sample" (Lind, Marchal & Wathen, 2005, p. 252). This technique does not fit well for aspects of the current study. Although random sampling was used in the cognitive testing phase of data collection, purposive sampling was used as the main thrust for the interviews and error analysis phases. The focus here was to engage data sources that were rich in content thereby facilitating interpretation from multiple perspectives.

5.4 INTERVIEWS

5.4.1 The Interview Sample

The interview sample consisted of 35 students from the target population who participated in a one hour semi-structured interview. Students were contacted telephonically and asked about previous programming knowledge, gender, language proficiency, current degree and maths and science proficiency. The selection of these specific factors was informed by the literature review and formed the basis for the purposive sampling strategy used. Interviewees were selected in order to obtain adequate representation from the spectrum of factors mentioned above. The inclusion of language proficiency was required in order to obviate any disadvantage to students who did not choose English as their primary form of communication. A demographic profile of the interviewees is provided in Table 2.

Table 2: Demographic profile of Interviewees

Age	20	21	22	23	24	Above 24
Count	5	10	11	5	2	2
Degree	Bsc	Bcomm	BSocSc	BA		
Count	10	18	5	2		
First Language	English	Zulu	Xhosa	Afrikaans	Hindu	
Count	24	6	4	1	0	
Prev Prog Exp (Years)	1..2	3..4	Above 4			
Count	22	6	7			
Gender	Male	Female				
Count	24	11				

5.4.2 Interview Sessions

The interview sessions were held over a period of two weeks. Students were asked to indicate a time preference in this two week period. In order to attach an incentive to their participation, students were each given two complimentary movie tickets. In hindsight however, it became clear that this was not necessary as there seemed to be an innate willingness on the part of students to make maximum use of this forum to express their sentiments on issues relating to computer programming. There were instances where interview sessions went beyond the intended maximum allocated time of one hour, resulting in a re-scheduling of some of the allotted time slots. Twenty students from each campus (Durban and Pietermaritzburg) of UKZN were invited to participate in the interview sessions. There was one student from the Pietermaritzburg campus and four students from the Durban campus who were absent from the interview sessions. However, this did not have a significant impact on the data as there was a liberal presence of respondents from each of the purpose areas alluded to in the previous discussion. In order to ensure that interviewees did not feel compelled to answer questions, it was emphasised that individual participation was purely from a data intensive focus and not any form of assessment or judgement of student ability.

5.4.3 Interview Data Presentation

The interview consisted of 12 questions (Appendix A). The main strategy involved classification of interviewee responses as either, "deep" or "surface"[10]. This judgement was made on the basis of the quality and depth of understanding inherent in the responses to each question. In accordance with the dictates of qualitative research, the researcher's subjective judgement, underpinned by knowledge in the subject area, is used to make the distinction between "deep" and "surface" responses. However, there were instances where an absolute classification could not be made. In these situations, a value of "neutral" was assigned to these responses. In order to demonstrate principles used in making the classification according to the deep and surface framework, a few examples of verbatim responses extracted from the interview transcripts are presented:

Here is a student response to the question on how to switch the values of three variables (Appendix A, Part C, Question 1):

"Create another constant variable X equalling ten, just as a value, to switch it and, then switch A from ten to thirty, then you just add two X, or if you want, well A could be X, so you could say A plus two A, gives you the new A, and B minus A equals the new B, and C minus A equals the new C." (Student 2, Tape 3b)

An analysis of the response reveals that the student was able to accomplish the required task for the specific problem presented. However, the lack of generalisation, curtailing any prospect of applying the same solution strategy to another set of input values for the same task will render the respondent as a surface learner in this instance. This classification is in direct contrast to the classification made for the following response:

[10] The deep and surface framework is discussed in Chapter Two (Literature Review)

"I initialise a variable temporary, then I take the value of C, and I assign the value of B to C. Oh, before that, I will assign C to a temporary variable, and then I will assign B to C, and A to B and I will assign the temporary, that temporary to A" (Student 2, Tape 3A)

In the instance above, this response is indicative of expert understanding of the problem domain where the solution offered is applicable in a generalised context. Hence, the student is classified as having "deep" learning traits in this instance. Another demonstration of the deep and surface framework is presented in relation to the following response on the applicability of an unconditional or "for…do" loop (Appendix A, Part B, Question 7).

Student *(in response to a question regarding the applicability of looping to a problem situation)*

"Do loop."

Interviewer

"The do or repeat loop. Why can't a for loop be used?"

Student

"Because for loops generally associated with arrays."

Interviewer

"Why wouldn't you say while loop? "

Student

"You can use the while loop but I'm more comfortable with a repeat loop." (Tape 1B, Student 1)

The student response is correct in this instance. In a written examination that required implementation of the looping construct, the student would probably score maximum marks. However, the explanation for the choice of looping structure is indicative of superficial or "surface" learning traits. The student was not able to offer an adequate generalisation for the implementation of the "for" looping construct. However, the student has been able to construct a personal cognitive model for looping that has been applied successfully for specific

problem situations. This conclusion is derived from the comment that "…for loops are generally associated with arrays". The problem here is that this cognitive model is not completely reliable and there are exception cases where it will not work. Hence the response provided by the student in this instance is classified as indicative of a "surface" learning approach. There were cases where it was not possible to make distinct classifications. This could be attributed to situations that required excessive intervention by the interviewer or cases where the responses given were somewhat confusing. These responses were interspersed with elements of "deep" understanding, but the impact of these responses was diluted by comments that may have been indicative of superficial understanding. In the following excerpt, the student was required to comment on the applicability of the "While" loop in preference to a "Repeat" loop. The response was as follows:

"Because the do repeat, when they ask the question 'do you want more items?' and then do everything it is While Loop; If they ask the question after doing everything it is a do Repeat Loop. "(Tape 3A, Student 2)

In this case the answer is correct and also indicative of meaningful understanding of the difference between both loops. However, the student did not allude to the significance of the fact that the "Repeat" loop was always executed at least once. The inability to make this generalisation begins to impart a measure of doubt to the inclination of classifying this response as indicative of "deep" learning. In cases such as this, there was no classification made, reflective of the researcher's neutral stance towards the response.

5.4.4 Interview Data Analysis

A count of the classification for each question was computed and used to provide an overall classification of "deep" or "surface" for each respondent[11]. In cases where there was no discernable difference between each classification, the

[11] Interview data "snapshot" presented in Appendix F – Part One

respondent was given a rating of “neutral”. A summation of the classifications reveals the outcome shown in Figure 13.

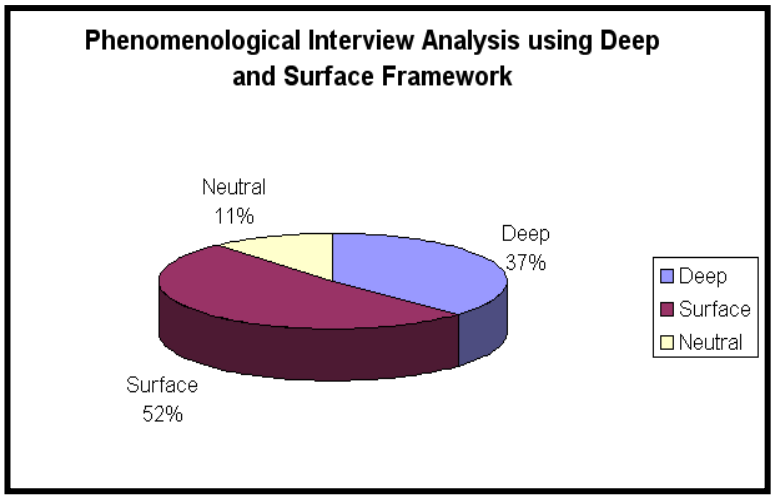


Figure 13: Results of Interview Analysis using the Deep and Surface protocol

The qualitative nature of this study precludes any adherence to the norms of quantitative methodology. However, in order to expand the potential of this study, an incursion into inferential statistics is undertaken. It should be noted that this may justifiably be viewed with suspicion since it is not the intention of the study to make inferences about the population on the basis of the sample used. In the case in discussion, the sample size is 35. However, responses from 4 of the interviewees were classified as “neutral”. Hence, this leaves a complement sample of 31 as the focus of some quantitative investigation. According to Lind *et al*. (2005, p.273), “…statistical theory has shown that samples of at least 30 are sufficiently large to allow us to assume that the sampling distribution follows the normal distribution”. Using an adjusted sample size of 31, the proportion of “deep” learners turned out to be 42% and the proportion of “surface” learners was 58%. The underlying qualitative ethos of the data gathering activities meant that with a sample size of approximately 12%, a confidence interval could be constructed for the population proportion. Applying the formula for a population

proportion, $p \pm z\sqrt{\frac{p(1-p)}{n}} = 0.58 \pm 1.96\sqrt{\frac{0.58(1-0.58)}{261}}$ and using a 95% confidence interval, we obtain the interval range to be between 50 and 62%. This indicates that at least 50% of the population will display a surface learning demeanour towards computer programming. The implication here is that the cognitive style adopted towards computer programming by at least half of the student population is problematic, warranting further investigation.

5.5 COGNITIVE ABILITY TESTING

The cognitive dimension has been identified as a valid source of enquiry for performance in computer programming (see Chapter Two of the current study).

5.5.1 Cognitive Test Sample

Random sampling was used as the strategy in this phase of data collection. A sample size of 60 students was targeted. The sample was picked randomly from the university's Student Management System (SMS). The test was administered in separate, controlled settings in Pietermaritzburg and Durban over consecutive days. An announcement requesting attendance from the "targeted" sample was made in a programming practical session in the Pietermaritzburg campus. However, there was a request from "non-targeted" students to be allowed to participate. This resulted in an unexpected turnout of 39 students from the Pietermaritzburg campus. Forty students from the Durban campus were subsequently contacted, either telephonically or via e-mail. However, only 28 students presented themselves for the test session. Hence, there were 67 participants who attended these sessions, (28 in Durban and 39 in Pietermaritzburg).

5.5.2 Cognitive Test Sessions

The cognitive ability test (Appendix B) consisted of 10 questions and the approximate duration was 30 minutes. Students were required to sign a form

(Appendix E – Part Two) that served to obtain their consent for participation in the test. The form also served to inform the students of the non-obligatory nature of the questions as well as the voluntary nature of participation.

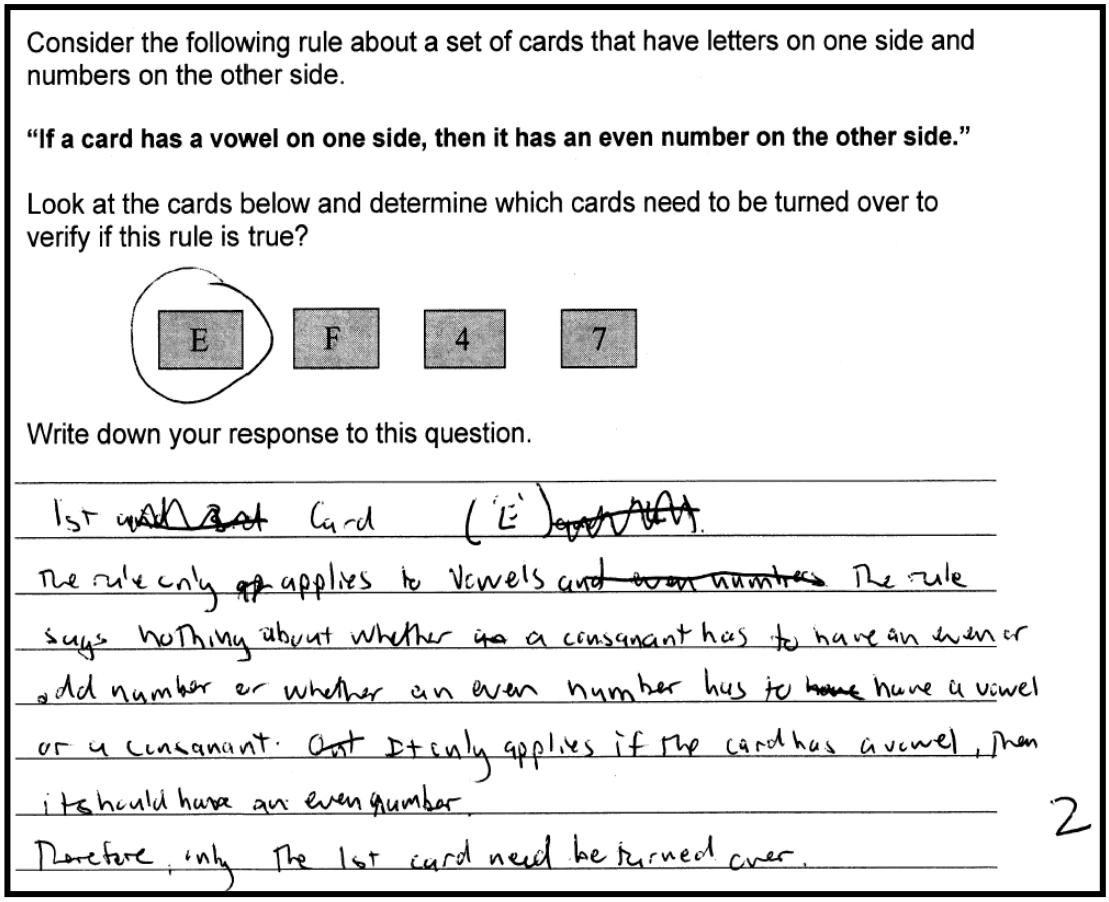
5.5.3 Marking Instrument

The focus of the test was to establish whether students had reached the highest level of cognitive maturity. Hence, 50% of the questions focused on aspects attesting to the prevalence of late cognitive development. However, these questions did contain aspects that necessitated the invocation of lower order cognitive skills. The marking scheme was adjusted according to this observation and credit was given for responses that were partially correct. The remainder of the questions (one each) was directed at early formal and mid-formal cognitive development. The marking system, differentiated according to the cognitive area being tested, is presented in Table 3:

Table 3: Marking Scheme for Cognitive Ability Test

Cognitive Classification	Number of Questions	Mark Scheme
Early Formal (EF)	3	One
Formal	2	Two
Late Formal (LF)	5	Three

An illustration of the marking scheme is provided by examining the sample presented in Figure 14.


Consider the following rule about a set of cards that have letters on one side and numbers on the other side.

"If a card has a vowel on one side, then it has an even number on the other side."

Look at the cards below and determine which cards need to be turned over to verify if this rule is true?

E F 4 7

Write down your response to this question.

1st Card ('E')

The rule only applies to Vowels and The rule says nothing about whether a consonant has to have an even or odd number or whether an even number has to have a vowel or a consonant. It only applies if the card has a vowel, then it should have an even number.

Therefore, only The 1st card need be turned over.

2

Figure 14: Cognitive Test Sample from Student SG

The student was able to identify one of the two correct answers (N.B. Card E and Card 7 are correct) and also offer a plausible explanation. The response is reflective of mental activity that establishes a concretisation of the problem domain by focusing on the physical presence of the cards. This leads to an immediate correlation of the card depicting a vowel with the requirements of the question. However, the student is able to extend this answer by making a generalisation regarding consonants. This ability to achieve such generalisation is classified as formal operational according to the Piagetian framework. Hence, in this case, a score of 2 out of a maximum of 3 marks is awarded. The student did not demonstrate higher order cognitive skills by engaging in the concept of "proof by contradiction" and recognition of the card with a "7" as a possible focus of enquiry. The maximum attainable score for the entire test was 22. Scores in the range 0 to 7 were classified as EF. The rationale here was that respondents would be required to obtain full marks for questions specifically classified as EF (a maximum of 3 marks), as well as part-marks for the EF aspects of questions

that were specifically classified as LF. Scores in the range 8 to 15 were similarly classified as formal while scores in excess of 15 were classified as LF. In order to be classified as LF, students would have had to get at least 3 of the LF tasks correct.

5.5.4 Cognitive Test Data Analysis

A tally of the final scores obtained by students is presented in Figure 15 (original data presented in Appendix F – Part two):

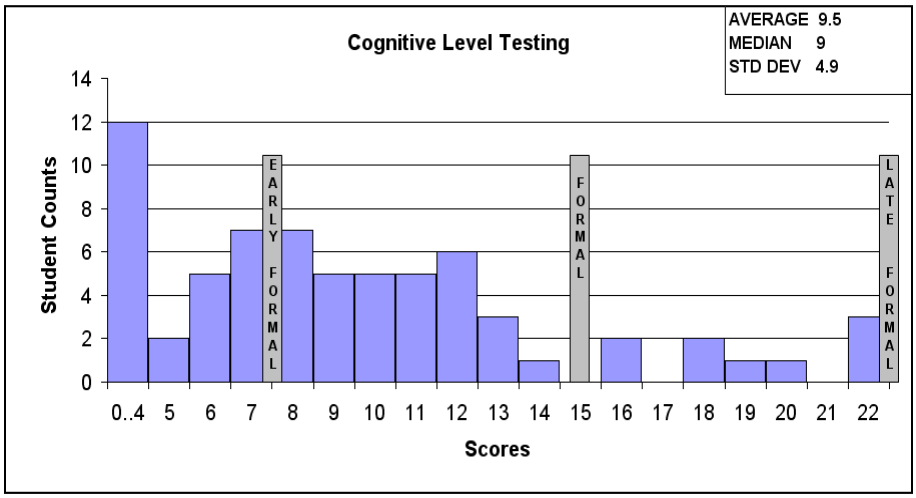


Figure 15: Graph of frequency counts for Cognitive Ability test

An initial analysis of Figure 15 reveals that a significant proportion of the sample displayed evidence of having made the transition from the operational cognitive level to the formal stage of cognitive development (approximately 61%). However, there was evidence that this transition has not been convincing and there are indications that a lack of semantic mobility may impede any further cognitive progression. This assertion is demonstrated by the response shown in the samples shown in Figure 16.

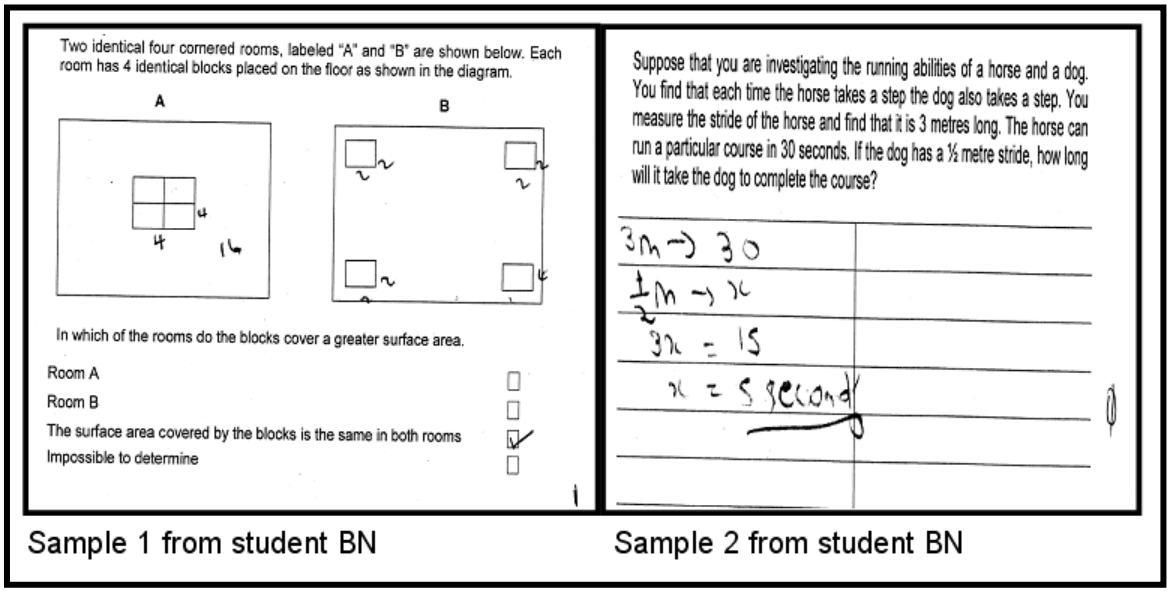


Figure 16: Samples from Cognitive Ability Test

The sample shown on the left (Figure 16, Sample 1) involved a test to ascertain an appreciation of the principle of area conservation. Although the correct response was given, it is quite significant to note from the evidence of the working shown that the student was more comfortable in the manipulation of the concretised model of the problem domain. It is expected that for a question of this nature, the student would be able to engage in abstract reasoning and provide an immediate response. According to Boeree (1999), a response to a question on the conservation of area, will be an indicator of a transition to the formal or abstract level of cognitive maturity. However, in the case presented from the sample data, it is clear that although the response is correct, there was very little abstract reasoning taking place. This assertion is corroborated in the next sample (Figure 16, Sample 2) taken from the same student. The application requires the invocation of the principle of inverse proportion. According to Kurtz (1980) and Barker & Unger (1983), the inverse proportion test is a measure to establish eligibility for the early stage of formal cognitive maturity. However, the incorrect response here is indicative of a subject who has not reached a cognitive

level that attests to satisfactory abstract thinking. From a holistic perspective, the subject in discussion achieved a score of 5 out of a possible 22 marks. This would be classified as the EF stage of cognitive development. This argument could be extended to assert that students classified as EF do not have the semantic mobility to meet the challenges of abstraction presented by computer programming. A further enquiry reveals that the average and median scores for the entire sample were 9.5 and 9 respectively. According to the classification scheme used, the average and median scores fall in the category of formal cognitive development. The graph in Figure 17 illustrates that a relatively high proportion of students would be classified as formal operational from a Piagetian perspective.

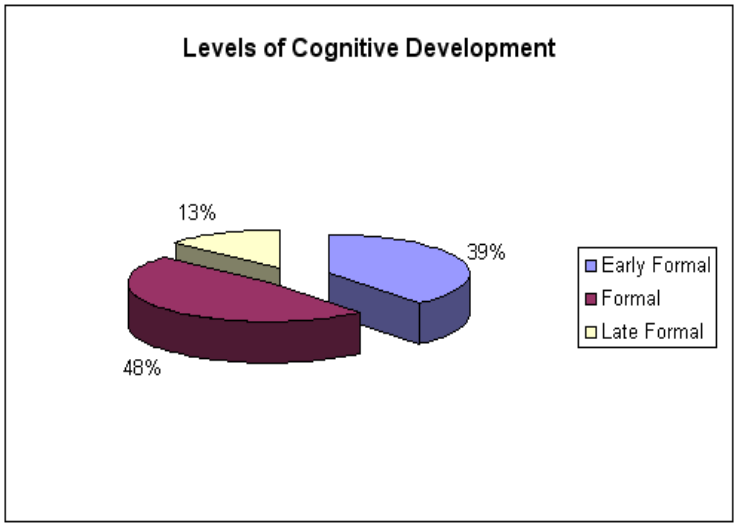


Figure 17: Graph showing proportional levels of Cognitive Development

An initial analysis of Figure 17 will reveal the significant observation that 39% percent of the sample taken did not exhibit adequate abstract reasoning skills that will be required to ensure a measure of competency in computer programming. A relatively minor component of the sample (13%) exhibited strong abstract reasoning skills rendering these subjects to be classified as LF in their cognitive development. Hence, the expectation here is that this group of the sample would have a strong grasp of propositional logic, thereby qualifying them

to be ideal candidates to achieve expertise in computer programming. A large component of the sample (48%) displayed adequate reasoning ability. This group is representative of subjects who have a satisfactory grasp of propositional logic, thereby necessitating pedagogical intervention to ensure the transition to strong formal reasoning skills. However, there is a clustering of scores around the boundary marker that represents the transition to the formal cognitive level. A sensitivity analysis of the sample in this region was conducted where the boundary marker for entry into the formal cognitive level is increased from 8 to 10, resulting in the following adjustment (Figure 18) to the output.

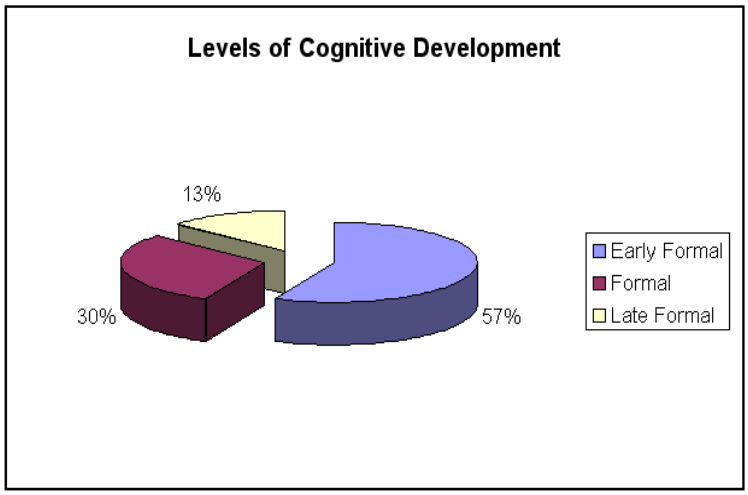


Figure 18: Graph of sensitivity analysis for levels of cognitive development

In the sensitivity analysis shown in Figure 18, the percentage contribution of the formal cognitive level experiences a significant reduction from 48% to 30% accompanied by a proportional increase in the EF category (39% to 57%). These observations contribute to the deduction that there is a strong presence of "border-line" subjects in the various cognitive levels. The implication here is that a discerning number of students do not have pre-requisite cognitive skills to handle the challenges of abstractionism presented by the discipline of computer programming. This creates a situation where students may well be doomed to failure prior to commencement of a computer programming course. Students who

have been classified as formal cognitive would be expected to exhibit a degree of propositional logic and abstract thinking. However, the tenuous nature of their presence in this category of cognitive development, as indicated by the sensitivity analysis exercise, implies that there is no guarantee of competence in the handling of abstract challenges posed by computer programming.

5.5.5 Inferential Dimension to the Cognitive Ability Tests

Using the 95% level of confidence, the interval for the population mean is calculated according to the formula, $\bar{X} \pm Z\frac{s}{\sqrt{n}} = 9.49 \pm 1.96\frac{4.9}{\sqrt{67}}$ giving an interval ranging from 8.3 to 10.7. Using this result as a catalyst, null hypothesis testing may be used to add some credibility to the suspicion that on an average, the students in the programming class have a cognitive classification that can be described as Early Formal. An adequate null hypothesis that would quantify this suspicion is that students in the programming class would have a mean score of 9 or less for the cognitive ability test (i.e. $H_0: \mu \leq 9$). According to Lind *et al*. (2005,p.329), in hypothesis testing for the mean, when the population standard deviation is unknown and the sample size is at least 30, the test statistic *z*, is computed by the formula: $z = (\bar{X} - \mu)/(s/\sqrt{n})$. Using a 5% significance level and a one-tailed test, the computed test statistic gives us a value of 0.89 which is below the critical value of 1.65. Hence the implication here is that the null hypothesis is not rejected, resulting in a conclusion that the population mean is less than 9. A p-value of 18.7% which is greater than the 5% significance is a strong indicator that the null hypothesis is true. The implication of this qualitative and quantitative exercise is that a typical student from the programming class would have a cognitive ability level that boarders on the Early Formal stage of intellectual development. This will render the student as inadequate in terms of responding to the challenges of abstractionism presented by computer programming. In order to further investigate this assertion, an analysis of errors was undertaken to relate errors made in programming tasks to cognitive levels.

5.6 ERROR ANALYSIS

Previous research efforts on the learning of computer programming have focused on the use of task based assessments, examinations and tests as the main instrument of evaluation (e.g. McCracken *et al*. (2001); Bergin & Reilly (2005); Simon *et al*. (2006)). Students' performance in these assessments were analysed with reference to the overall symbol or mark achieved. In order to obtain a deep insight into the cognitive activity that prevails in response to programming related assessments, an error analysis of student responses in a programming examination was undertaken. This exercise entailed the application of the error analysis framework presented in Chapter Three of the current study.

5.6.1 Error Analysis Sample

The error analysis framework was applied to the ISTN212 examination scripts of 46 of the 67 students who participated in the cognitive ability test. A purposive sampling stance was adopted where the selection was random in nature, but there was a bias in ensuring that there was adequate representation from each cognitive category (early formal, formal and late formal).

5.6.2 Error Analysis Methodology

A frequency count of each category of error was computed. The errors were classified as arbitrary, executive or structural. This was done in accordance with the error analysis framework developed in Chapter 3. In the case where an absolute classification could not be made because of the possibility of an error falling into one of two categories, the higher level classification would take precedence. In the case where there was no response or minimal response to a question, an overall error classification of structural was made. The rationale here is that the student was not able to achieve a problem domain translation into a cognitive framework that would facilitate a programming solution. According to the error framework, this will qualify as a structural error. However, these cases were problematic as the main focus of qualitative research is to engage in

samples that are rich in content from an analysis perspective (Hoepfl, 1997). Apart from this setback, many of the samples analysed were crucial in illuminating prevailing error patterns.

5.6.3 Error Analysis Data Presentation

A demonstration and discussion of the error classification is presented with reference to a subset of the sample data[12]. This discussion will be presented under the categories of flowchart implementation, algorithmic manipulation and data structure implementation. These are the broad areas that comprised the examination based assessment.

5.6.3.1 Flowchart Implementation

In this assessment task, students were required to write the code for a flowchart diagram (Appendix C, Question One). A sub-task entailed the writing of the source code for a procedure that is invoked in the flowchart. The procedure entailed the generation and storage of random integer values into an array data structure. An application of the error analysis framework to samples from the data collected is presented according to the different error types.

Arbitrary Error

An instance of an arbitrary error for this task is presented in Figure 19 using an example from the sample data:

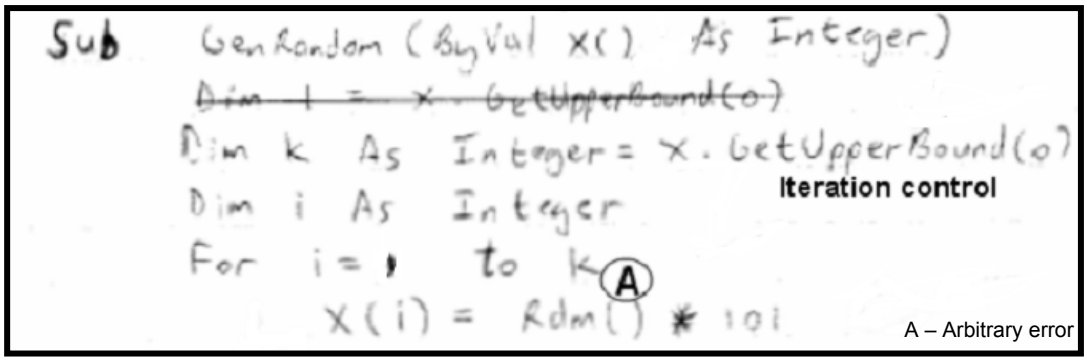


Figure 19: Sample from script T1 illustrating an arbitrary error – denoted be the letter A

[12] Error frequency count data is presented in Appendix F – Part Three

It can be observed in Figure 19 that the implementation of the random number function is flawed. However, the student had knowledge that this function had to be invoked, although the constraint that the number generated had to be an integer, was not observed. Another possible instance of a similarly classified error is the possibility that the iteration structure control values are incorrect. From a marking perspective, this will result in a loss of marks. However, from an error analysis perspective, these errors are classified as arbitrary. The implication here is that there is no serious cognitive flaw and remedial action should be trivial.

Executive Error

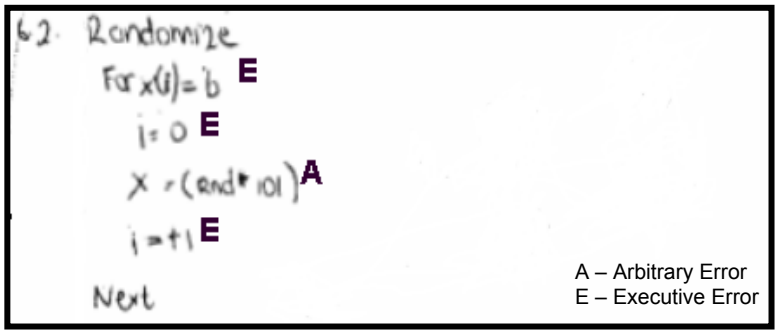


Figure 20: Sample illustrating executive errors from script R1

An analysis of the sample shown in Figure 20 reveals four incidences of executive errors. Three of these are related to the looping construct. These can be further classified as the lack of understanding of the concepts of loop control, variable initialisation and counter incrementation. The fourth error manifests in the inability to implement the counter control as an array index. From a cognition perspective, the student was able to create a logical map of the problem domain into a plausible programming related solution. However, the lack of ability to present the solution in a manner that conforms to proper programming

convention qualifies these errors to be executive. It should also be observed that from a mark allocation perspective, this response will earn no marks. However, from the error analysis perspective, the student will be accorded credit for invoking the correct solution structure.

Structural Error

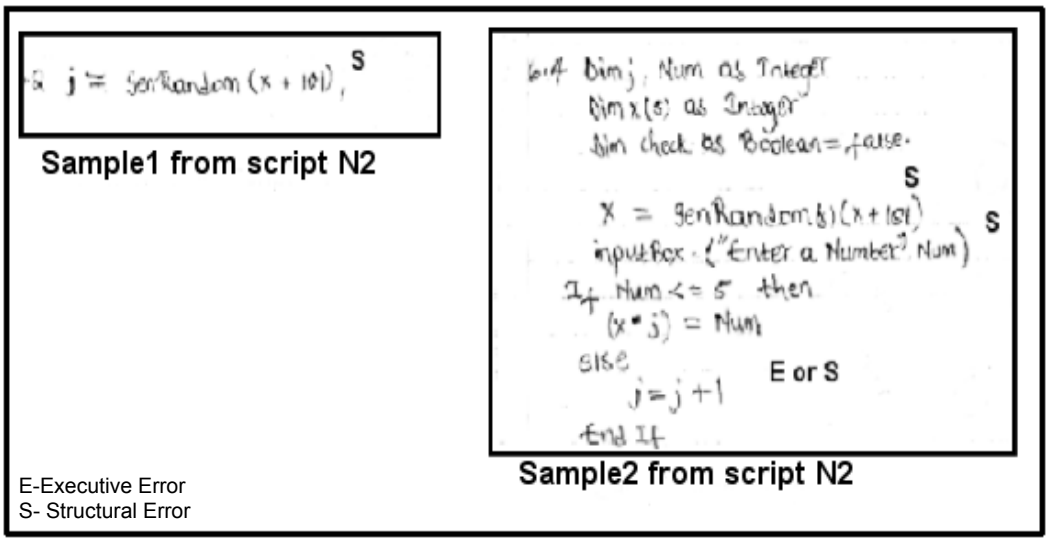


Figure 21: Samples illustrating structural errors

The structural errors shown in Figure 21 are indicative of a lack of comprehension of the problem domain. The incoherent nature of the solution indicates an attempt to randomly put together learned programming constructs that have little relevance to the problem domain. The response in Figure 21, Sample 1, shows complete oblivion of the need to invoke iteration structures. In Figure 21, Sample 2 there is a clear indication of a lack of coherent sequencing. The input control is included on an arbitrary basis rather than a logical one. In the same sample (i.e. Figure21, Sample 2) there is also evidence of an inability to correctly implement a counter increment within a looping structure. This error manifests a situation where the error could be classified as executive or structural. The executive error could be as a consequence of a failure to correctly implement a counter increment within a looping structure. However, the overall lack of awareness of the applicability of the looping construct in the solution also

renders this error to be structural. In terms of the error analysis framework, the higher order error, which in this case, is the structural error, will take precedence.

5.6.3.2 Algorithmic manipulation

The students were given the task of analysing a distorted code fragment (Appendix C, Question Two) designed to sort a set of array elements, and construct a proper solution.

Arbitrary error

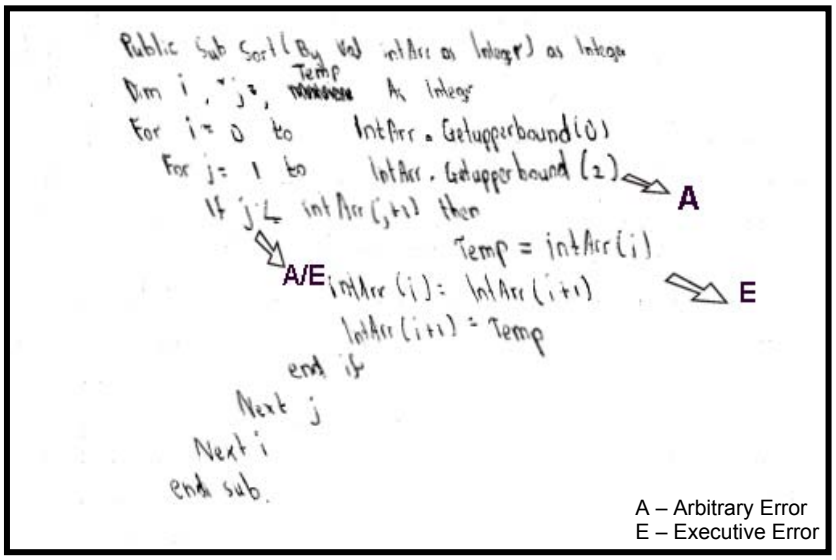
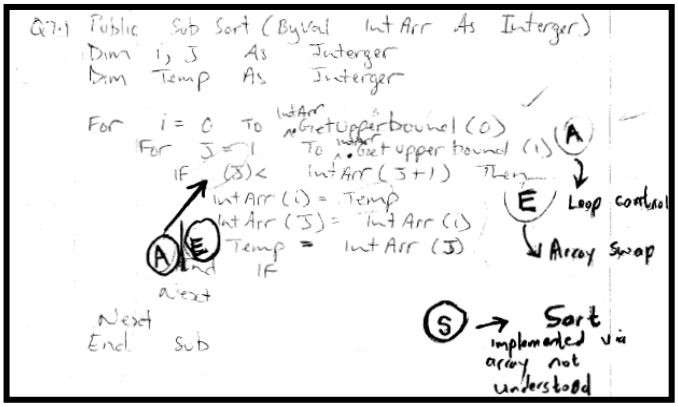


Figure 22: Sample from script BB1 illustrating an arbitrary error

An analysis of a sample response shown in Figure 22 reveals two incidences of arbitrary errors. The first instance (marked A in the sample) is a clear case of a failure to abide by the constraints with respect to the inner loop. The implementation of the looping construct is flawed with regards to the incorrect looping control de-limiters (i.e. the upper bound of the inner loop is incorrect). However, the awareness and sequential coherency of the iteration structure as presented in the sample eliminates the prospect of a structural or executive error.

Hence the failure to enforce the correct boundary values for the iteration structure can be classified as arbitrary. This sample also highlights the dilemma of making an absolute error classification when there is a possibility of a dual classification. In the second error instance (marked A/E in the sample) the operands are incorrect. This could be classified as arbitrary as the student could have had the intention of doing a comparison of array elements, but by omitting the array variable, this becomes a comparison between two array indexes. This renders the solution to be incorrect and a minor modification is necessitated. However, this error could be symptomatic of a serious flaw in the student's cognitive model for array processing. An error of this nature could thus be classified as an incorrect implementation of the array concept. In this case, the error will be classified as executive. According to the error analysis framework, this dilemma is resolved by according precedence to the higher order error, classifying the error as executive.

Structural Error



A – Arbitrary Error
E – Executive Error
S – Structural Error

Figure 23: Sample from script F1 illustrating a structural error

In order to identify structural errors, a more holistic judgement had to be made. In the sample shown in Figure 23, there are concrete instances of arbitrary and executive errors. However, the judgement made to signal the presence of a structural flaw is as a consequence of the following observations from the sample:

- The lack of understanding of the selection structure controlling the array sort algorithm
- Failure to recognise the difference between array elements and array indexes
- Inability to comprehend the logic inherent in the swapping of the values of two variables

These flaws contribute to a response that entails the random piecing together of code segments with no coherent sequencing that may facilitate the production of a logically viable solution. Such an exercise will result in a proliferation of arbitrary and executive errors. However, the underlying structure of the solution is flawed, resulting in the instantiation of a structural error.

5.6.3.3 Data Structure Implementation

Students were given the task of analysing a user defined data structure[13] declaration (shown in Figure 24) for subsequent processing tasks.

```
Structure Sales
        Dim Name As String
        Dim Sales() As Integer
End Structure
Dim Employees(9) As Sales
```

Figure 24: Code declaration for user defined data structure

In order to become effective users of the data structure, students had to conceptualise the structure and become comfortable with the idea of engaging in subsequent manipulation of it. However, this conceptualisation process will be constrained by the student's level of understanding of data structures. This will present a situation where students with poor assimilation qualities, may present a solution that is comprehensively flawed, reflective of complete oblivion to the

[13] Discussed generically in Section 3.4.2

problem domain. In such an instance, error analysis from an arbitrary and executive perspective becomes redundant and the only error that can be identified is structural as illustrated in Figure 25.

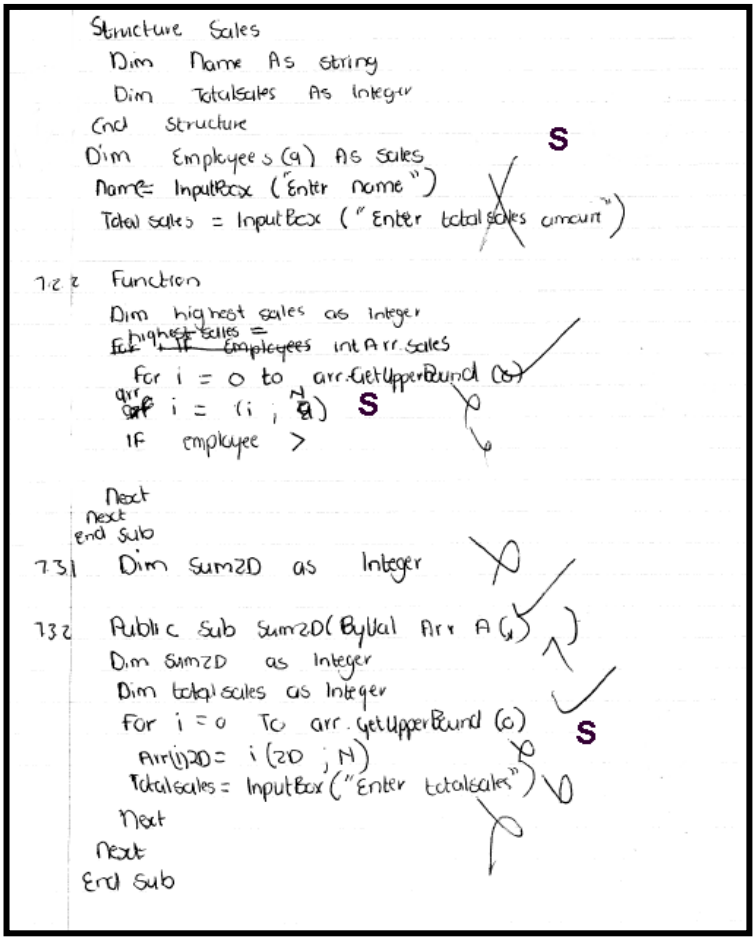


Figure 25: Sample from script C2 illustrating structural errors for user defined data structure

A study of this response shown in Figure 25 reveals complete cognitive disorientation with respect to the data structure. The student is aware that input needs to take place and the processing involves array manipulation, necessitating the invocation of iteration structures. The response is assembled using this knowledge coupled with a fervent attempt to establish a link with the fields in the data structure. From a mark scoring perspective, this effort will result in the allocation of a few marks. However, from an error analysis perspective, a holistic judgement is made that renders such a response as structurally flawed.

In contrast, there were many solutions that were coherent and logically correct. However, in the process of presenting the solution, arbitrary and executive errors were detected as illustrated in the following samples (Figures 26 and 27):

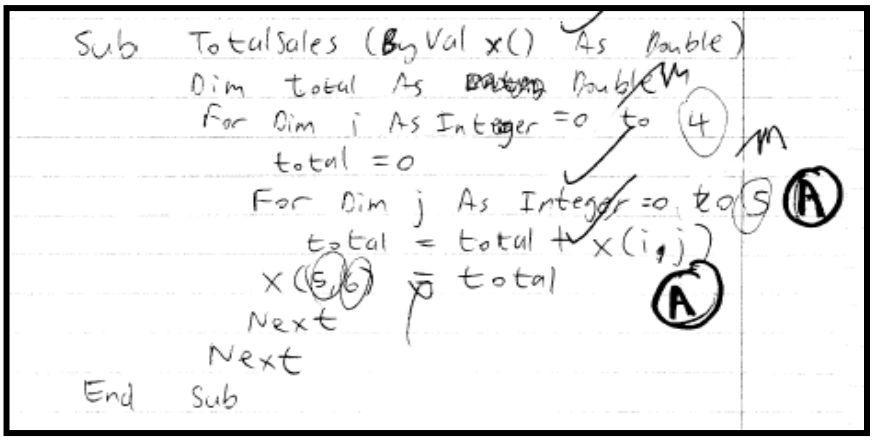


Figure 26: Sample from script Q3 illustrating arbitrary errors for user defined data structure

The sample in Figure 26 shows a case of errors manifested in the form of incorrect loop control and array referencing indexes. However, the solution is coherent from a holistic perspective and the implementation is merely flawed by incorrect observation of the looping and array constraints. Hence errors of this type are classified as arbitrary. There were cases where the solution reflected genuine comprehension of the data structure declaration. However, flaws in the manipulation resulted in the instantiation of executive errors as illustrated by the samples shown in Figure 27.

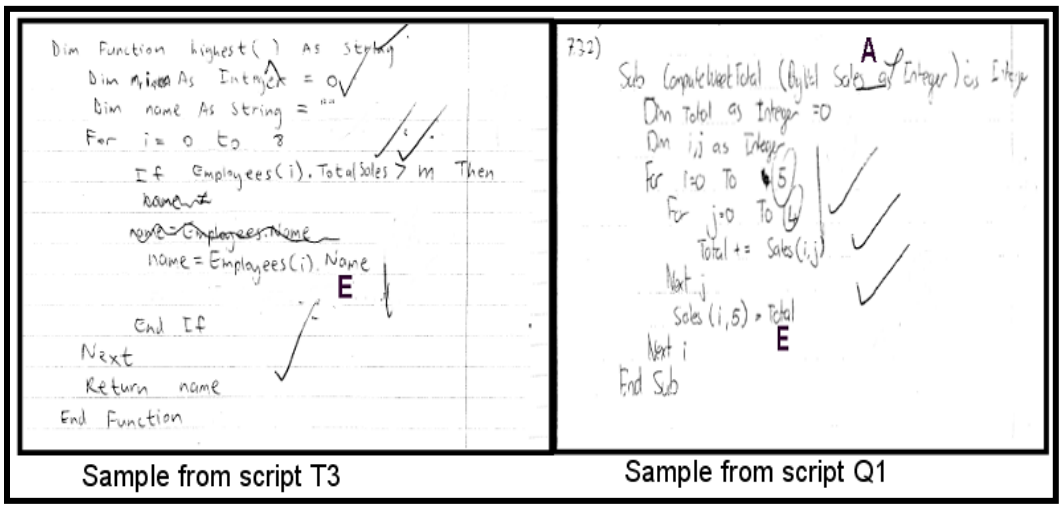


Figure 27: Samples illustrating executive errors for user defined data structure

In the first instance (Figure 27, script T3), the solution attempts to search an array for details about the record that has the highest value in one of the specific fields. Although the solution strategy is correct, the implementation is flawed because the temporary variable that is used as source of comparison with the data items in the different records of the array is not updated. Although, the solution strategy entails the return of the correct data item, the implementation is incorrect. In the second instance (Figure 27, script Q1), the solution strategy is correct. However an implementation flaw (re-initialisation of the accumulator variable at the end of the loop) will result in incorrect output. In both the cases illustrated by the samples, these errors are classified as executive.

5.6.4 Error Data Relative to Cognitive Ability

In order to extend the interpretivistic demeanour of the error analysis exercise, the types of errors made by each student was collated and tabulated according to the three different programming categories. The dimensions of this tabulation were extended to incorporate the cognitive maturity levels which were established during the cognitive ability tests. Table 4 reflects the data outcome of this exercise.

Table 4: Frequency counts of error types relative to level of cognitive

Description	Cognitive Maturity Level	Error Frequency		
		Arbitrary	Executive	Structural
Flowchart Implementation	1	13	17	16
	2	7	5	4
	3	5	2	0
Algorithmic Implementation	1	13	11	22
	2	8	6	5
	3	6	1	0
Data Structure Manipulation	1	12	20	34
	2	8	9	20
	3	8	2	1

Averages of errors in each category were used to create a graphical representation that reflected a summary of the data from Table 4. The graph

(Figure 28) shows a trend that highlights cognitive maturity level as a significant contributory factor to the overall set of errors detected.

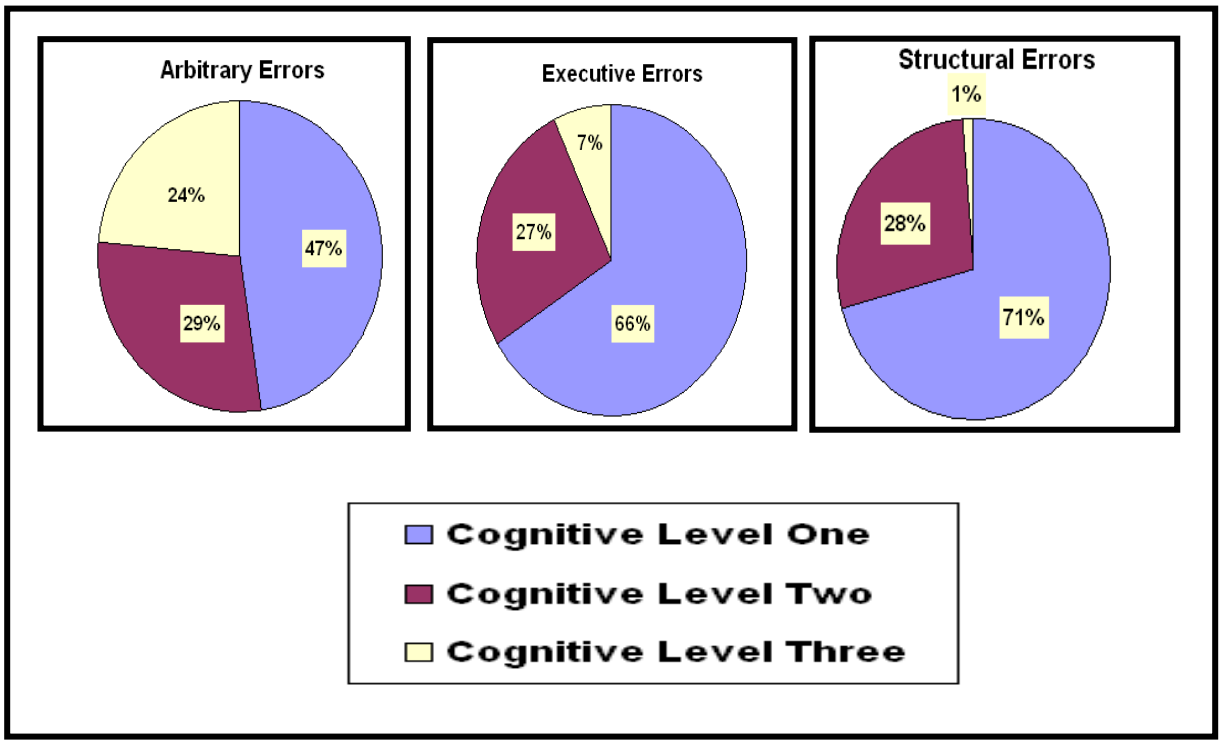


Figure 28: Graph illustrating error type relative to levels of cognitive maturity

The relatively low frequency counts for executive and structural errors in the category of flowchart analysis (Table 4) can be attributed to the fact that this question did not incorporate elements of abstract reasoning. However, questions that required an invocation of problem solving skills, incorporating a high level of abstract reasoning, resulted in a trend that could be summarised as follows:

Students with a higher level of cognitive maturity (levels two and three) have a significantly lower incidence of executive and structural errors. There is a strong negative correlation between error counts and the levels of cognitive maturity.

Another significant observation with respect to error trends within each of the cognitive groups is revealed in Figure 29.

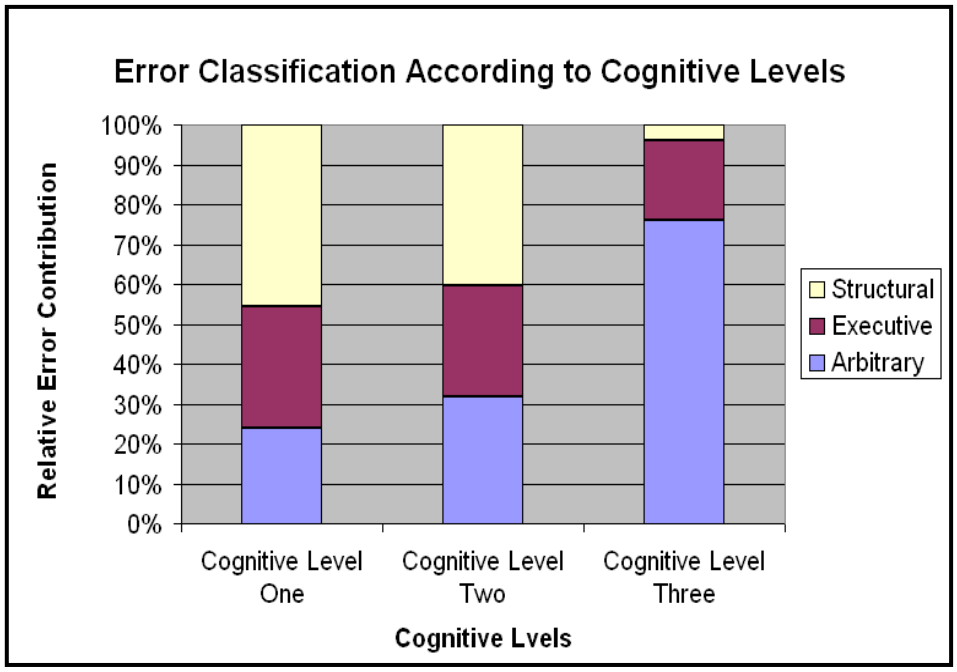


Figure 29: Graph illustrating cognitive levels relative to error types

The percentage contribution of structural errors shown in Figure 29 diminishes significantly with each transition from a lower cognitive level to a higher one. This observation is corroborated by the regression analysis shown in Figure 30.

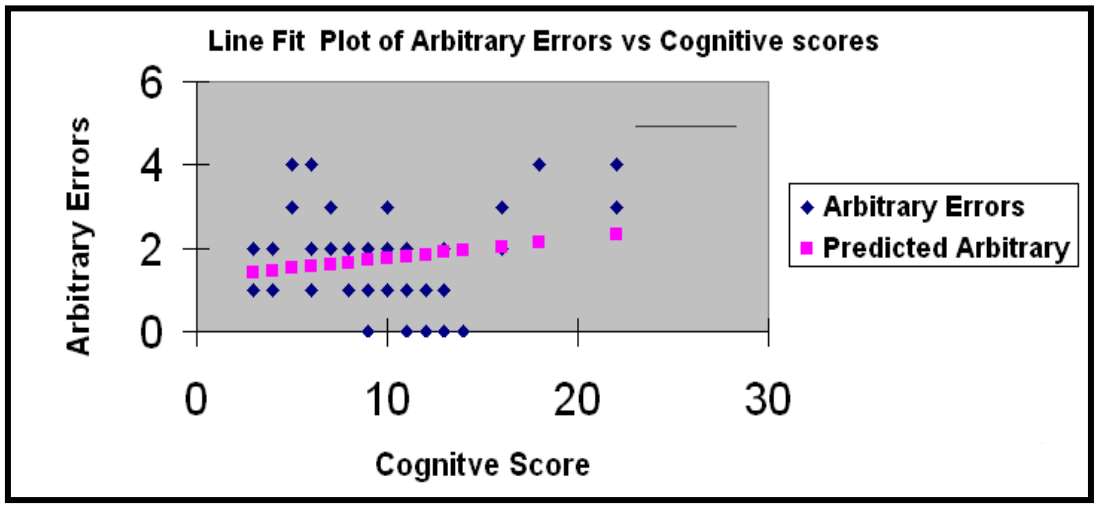


Figure 30: Regression analysis graph of arbitrary errors relative to cognitive score

A regression analysis of arbitrary errors plotted against cognitive scores is not conclusive in establishing any relationship between the variables. A correlation co-efficient of 0.21 indicates a weak positive relationship between the cognitive

scores and the frequency of arbitrary errors, implying that there is a subtle increase in the frequency of arbitrary errors when the cognitive scores are increased. However a deeper inquisition reveals that this is not surprising in terms of the implementation of the error analysis framework. According to the framework, in situations where an absolute error classification could not be made, and there is a possibility that two different errors may prevail, the higher order error would take precedence. In keeping with this rule, it was observed that although the subjects from the lower cognitive level did make numerous arbitrary errors, these errors may have been overridden by a higher order error classification after the solution was judged holistically. For subjects with higher cognitive scores, the error classification was much easier to make, and arbitrary errors were not elevated to a higher order error classification. An investigation of the possibility of a relationship between executive errors and cognitive scores revealed the following relationship shown in Figure 31.

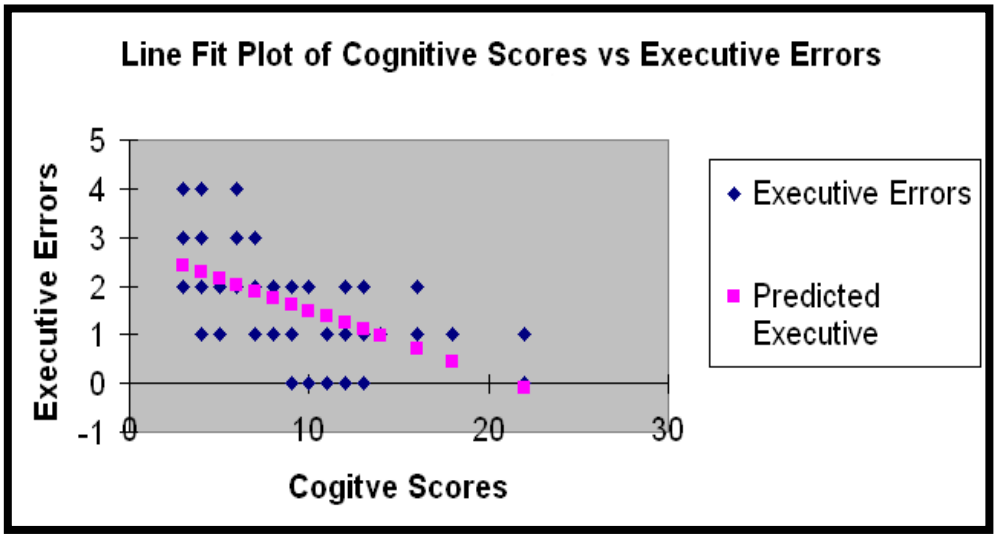


Figure 31: Regression analysis graph of executive errors relative to cognitive score

The strength of the relationship illustrated in Figure 31 becomes more significant than that which was observed with arbitrary errors (Figure 30). However, the negative correlation co-efficient of 0.58 is not entirely convincing. A possible

explanation for the executive errors is that in certain instances, such errors could be attributed to a lack of understanding of the problem domain, in which case, these could well be classified as structural. This error generating cascading effect is detrimental in facilitating any kind of meaningful analysis of the problem situation. However, the data presented does begin to allude to the tendency for movement towards a generalisation that the error severity is inversely correlated to the cognitive scores. This pre-mature generalisation is given more credibility by the regression analysis between structural error frequency and cognitive scores shown in Figure 32.

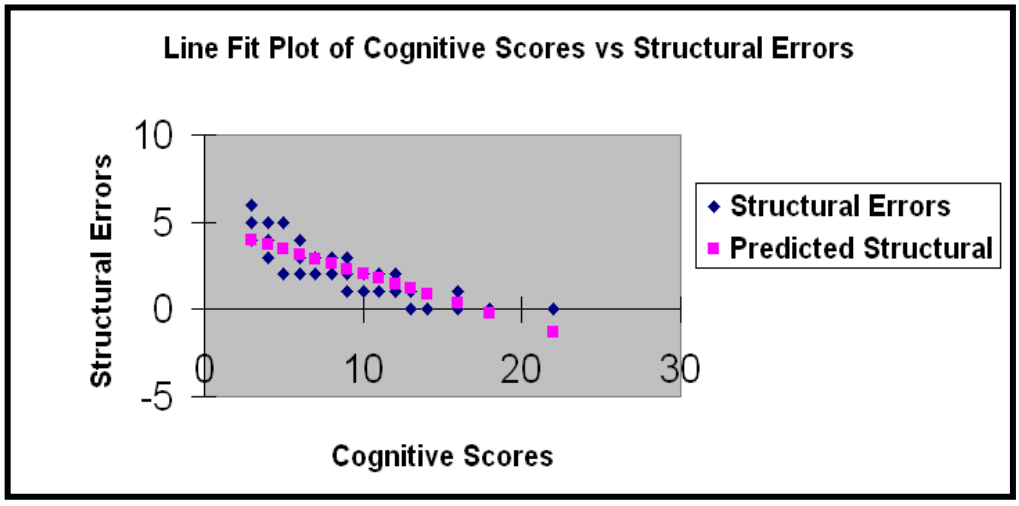


Figure 32: Regression analysis graph of structural errors relative to cognitive score

The graph in Figure 32 shows a significant relationship (correlation co-efficient of negative 0.83) between the cognitive scores and the structural error frequency. The immediate conclusion is that students with higher cognitive ability are less prone to committing errors of a structural nature. The implication herein is that these students are more adept at surmounting the abstractionism inherent in computer programming thereby obviating the possibility of committing structural errors. The inconclusiveness of the previous correlation analyses (Figures 30 and 31) could well be explained by the assertion that once the hurdle of abstractionism is overcome there is still the possibility that these students are

prone to syntactical error and to a lesser extent, semantic errors. Hence, this may offer a possible explanation for the proliferation of arbitrary and executive errors amongst the various cognitive groups.

Chapter SIX

6. CONCLUSIONS AND RECOMMENDATIONS

The conclusions and recommendations in the current chapter are underpinned by the error analysis theoretical framework presented in Chapter Three as well as the analyses emanating from the data collection activities presented in Chapter Five. However, prior to delving into the conclusions of the research, it is only appropriate to offer comments on the effectiveness of the research paradigm in facilitating the attainment of the sub-objectives of the study.

6.1 Attainment of Sub-Objectives

6.1.1 Insight into the understanding of computer programming

In order to achieve this sub-objective, a phenemenographic approach was used to obtain an outside view of the mental processes that prevail within the minds of students when they are required to think about computer programming tasks. The exuberance detected in response to the forum provided to these students was indicative of the sincerity of the responses given. The use of a small sample size did not dilute the richness of the information gathered. Although the current study was restrictive in making use of the deep and surface protocol for the analysis of these responses, a fundamental assumption underlying phenomenography is that the data acquired may well be analysed in a number of qualitatively different ways (e.g. see Appendix D). The results indicate that a high proportion (47% of the sample extrapolated to 50% of the population) of students in the introductory programming class exhibited only surface understanding of computer programming concepts. This outcome helped achieve the sub-objective under discussion as well as provide evidence to suggest that the cognitive style adopted by students towards the understanding of computer programming concepts was superficial, thereby impeding the possibility of the attainment of expertise status as a computer programmer. On the basis of the

conclusions in literature review (Chapter Two), this demeanour could be attributed to a lack of mental capacity by students to handle the rigours of abstractionism posed by the discipline of computer programming. However, this assertion became the subject of enquiry for the remaining sub-objectives of the study.

6.1.2 Development of A Theoretical Framework for the Analysis of performance in Computer Programming

Informed by the recommendations of the cognitive science fraternity, errors made in programming tasks were used as a basis for analysing programming solutions provided by students in a computer programming examination. The focus of this framework was to facilitate the identification of structural flaws that may underpin students' thought processes in computer programming. The theoretical error analysis framework was successfully developed after adequate consultation with the Donaldson/Naidoo framework that had a similar agenda (for the discipline of mathematics). The implementation of the framework on sample data was preceded by an application to generic programming concepts in order to illustrate the logic employed in its mode of operation. This framework was sufficiently well developed for implementation in a computer programming examination. The purpose here was to facilitate a microscopic investigation of mental activity that informed student responses to computer programming tasks. The default situation of using student examination marks to facilitate a similar enquiry can be viewed with a measure of doubt. This assertion is made on the basis of the argument that students with a higher cognitive ability are still very much prone to making arbitrary errors (confirmed in the regression chart presented in Figure 30), for which they will be penalised under examination protocol. Similarly, students with lower cognitive ability may be given sufficient credit for responses that can be classified as arbitrary, but do not form an integral component of a complete, coherent solution. Hence the final mark achieved in an examination may be somewhat of a misnomer in establishing the potential for expertise in

computer programming. The error analysis framework purports to obviate this problem.

6.1.3 Establishing the level of cognitive development of a computer programming student

The Piagetain framework for cognitive development was used to classify students according to various levels of cognitive development. A cognitive ability test, informed by the Piagetian framework for human cognitive development was devised and administered to 67 students. The response rate of 26% was fairly representative of the population and sufficient to engage in inferences about the population. It was established that the highest proportion of students fall into the pre-formal or formal classification in terms of the Piagetian framework for cognitive development. Hence the sub-objective of ascertaining the students' levels of cognitive development was achieved. However, more significantly, the implications of the outcome of this line of inquisition reveals that many of the students enrolled for programming courses do not have the ability to engage in the level of abstract thinking required by computer programming.

6.1.4 Investigation of a relationship between cognitive ability and performance in computer programming

This phase of the study embodied a synthesis of the sub-objectives discussed in 6.1.2 (error analysis framework) and 6.1.3 (level of cognitive development) in conjunction with regression analysis theory to establish the possibility of a significant relationship. The error analysis framework was used to facilitate quantification of student performance in computer programming according to errors. This was achieved by doing a frequency count of the different types of errors made in a computer programming examination. The error frequencies were then correlated with the students' levels of cognitive development. The integration of the results of the error analysis and cognitive testing routines were instrumental in facilitating an investigation of a possible relationship between these variables of the study, heralding the successful attainment of the sub-

objective under discussion. In terms of the relationship, it was significant to observe a strong negative correlation between the severity of errors made and the level of cognitive development. The implication provides a pointer to the assertion that cognitive development is a strong predictor of proficiency in computer programming.

6.2 Conclusions

The study was conducted in the context of poor student performance in computer programming. This study has established that a possible cause for poor performance is a lack of deep understanding of fundamental concepts that underpin the discipline of computer programming. The literary inquisition into factors that may influence the level of understanding in computer programming revealed that the cognitive dimension is a possible area of investigation. Hence, this study undertook to establish the exact nature of the influence exerted by the cognitive dimension. The sub-objectives of the study succeeded in establishing frameworks to facilitate the quantification of the qualitative variables representing cognitive development and programming competence. The study provided sufficient evidence testifying to the following conclusions:

A high proportion of students enrolled for an introductory programming class do not have the necessary cognitive ability to handle the requirements of abstractionism imposed by computer programming. As a consequence:

- students engage in surface/superficial learning with the intention of passing the examination for the course
- the frequency of errors reflecting a lack of understanding of programming concepts (executive) is inversely related to the level of cognitive development -
- the frequency of errors reflecting complete disorientation with respect to presenting a programming solution for a specific problem (structural

errors) has a strong inverse relationship with the level of cognitive development

Hence, students with a lower cognitive ability engage in a superficial cognitive style that is commensurate with their ability to handle the cognitive challenges posed by computer programming. This resulted in a proliferation of executive and structural errors. The study also found that there is no significant relationship between the frequency of arbitrary errors and the level of cognitive ability.

6.3 Recommendations

In keeping with the domain of expertise of the author of this study, the recommendations will be pedagogically anchored. According to Ali (2005), "…teaching is a scholarly activity and a life-long learning process with no single method or pedagogy that is always most effective". From a computer programming perspective, cognisance needs to be accorded to the prospect that there are students of varying cognitive ability coupled with varying cognitive styles of learning that enter a programming course. An expectation should be that a large proportion of students will exhibit a surface approach to learning and according to Booth (2001), this entails learning to code in a language or as passing the course. It is envisaged that this kind of practice would not be condoned by the academic community. However, the current pedagogical examining style as espoused by Bloom's Taxonomy requires that assessment tasks be categorised to cater for the varying cognitive levels. These range in a continuum from simple recall skills to higher order analytical skills. It is within the confines of this taxonomy, that students with a surface knowledge of programming concepts may achieve their objective of passing a programming course. The conclusion of this study adds impetus to the argument that current teaching strategies are susceptible to the prospect of allowing students, with unsustainable abilities as programmers, to be deceived into classifying themselves as expert programmers. A strategy to obviate the "examination

driven" agenda typical of the academic world is for academics to analyse computer programming solutions using the error analysis framework developed in this study. An inordinate prevalence of structural errors in computer programming solutions is reflective of a "deep" misunderstanding of computer programming fundamentals. The fruition of such a situation could be symptomatic of a lack of cognitive development that would impede the invocation of abstract reasoning.

Programming teaching strategy/curriculum design should be based on the perceived level of cognitive maturity of the student population as presented by the current study. This assertion is supported in previous work in this area by, amongst others, White and Sivitanides (2002), Leongson & Limjap(2003), Bergin & Reilly (2005) and Dehnadi (2006). A diversified strategy would ensure an element of dynamism that represents a deviation from the traditional teaching strategies which are based on the assumption that all students have an equivalent cognitive capacity to handle the learning of new material. Teaching strategies at various levels of intensity need to be explored in order to compensate for the lower cognitive learners and make inroads into the reduction of structural errors. This approach could enhance the possibility of deeper learning traits being manifesting across a wider range of computer programming students.

6.3.1 Easing the Cognitive burden

This study highlights the cognitive burden placed by the demands of computer programming on college level students in an introductory programming course. A recommendation from this study is that the academic community should engage in pedagogical inventiveness coupled with intelligent manipulation of the available technology to lighten this cognitive burden. As part of an effort to reduce the cognitive load inherent in programming, academic institutes could focus on the so called "tools of the trade" or the choice of programming language as a possible source of avenue for investigation. Implementation of this strategy

could entail the use of scripting and visual programming (VP) languages[14] as a teaching tool. Argumentative support for this strategy is provided in Warren (2001) and Pea & Kurland (1984).

6.3.2 The Scripting Languages

According to Warren (2001), the scripting languages "…are sophisticated and powerful environments in which students can learn the basics of programming without the unnecessary complications entailed in the alternative approach". The alternative approach refers to the "system programming languages" such as Java, C++, Basic and others of a similar ilk. The rationale for supporting this stance is that the development environment, consisting of a text editor, is simple, lightweight (in terms of resource consumption) and highly portable (a text based application that can be executed on any Web browser). The main focus of this argument on interface simplicity is that it forces the teacher to concentrate on programming concepts. This is an affirmation of Warren's sentiments that "… it is better to do without some features than to overload the environment with too much gratuitous complexity". A deeper inquisition brings the issue of strict, static data typing found in all "system programming languages". The requirement that variables have to be given an initial type is a machine/compiler related issue. According to Du Boulay *et al*. (1999), this immediately increases the level of abstractionism and is not consistent with the way people think. This view is endorsed by Warren in his comment that "…the novice can recognize the type from the context accurately enough to write elementary programs". This cognitive burden is obviated in the scripting languages where variables are dynamically typed, the consequence being that there is no need to declare them. An interesting reference is made to an excerpt taken verbatim from the interview session in the data collection phase of the study:

***Lecturer**: Now when you use a variable in programs, you use dim to declare variables and you always declare a variable to have a type, such as integer, float, char, Boolean. Do you think it is important to declare variables with a type?*

[14] VP is explained in Section 6.3.3 – p. 96

***Student**: Yes*

***Lecturer**: Why would you say that?*

***Student**: Because it depends on what you intend to do with it, if you use it for maths you can't leave it as a character and can't manipulate the information. So depends on what you are using it for it is very important to have a type*

***Lecturer**: What if I was to show you a programming language and you didn't have to declare variable types, you just use the variable however you want. Do you think you could work in this programming language?*

***Student**: Yes, definitely, it will make the programming experience easier. The focus of programming is to solve problems, if you have help to solve a problem, why not use it? The language used should be providing help to us not become the problem itself"*

(Student 1, Tape 5A)

In this excerpt, the quality of the response is indicative of a subject knowledgeable in computer programming. The nature of the response is also an affirmation of the need to focus on problem solving from a human perspective rather than attaching priority to technical requirements. The case for reducing the cognitive burden is also supported by Du Boulay *et al*. (1999). when they introduce the concept of a notional machine as being the "…idealized model of the computer implied by the constructs of the programming language". They go on to assert that novices have very little idea of the properties of the notional machine and that they should be introduced to programming through languages that "embody simple notional machines"[15].

6.3.3 The Visual Languages

VP has experienced a much needed impetus by the success of the object-oriented programming (OOP) methodology. VP entails the manipulation of visual objects on a computer screen and "…drag and drop generation of code segments" (Sebesta, 2006, p.23). In a sense, the level of abstractionism typical

[15] This can be achieved by having a small language with a few constructs, thereby limiting the domain of action but also at the same time keeping the range of machine actions clear and concise to the programmer.

of most “system programming languages” is significantly reduced within the VP environment. The reasoning here is that a manipulation of visual objects embodies a concretisation of some aspects of the notional machine (referred to in 6.3.2). The availability of increased processing capabilities has facilitated interventions of this nature as is embodied by the recent graphical demeanour of the latest operating systems[16]. In order to facilitate this concrete manipulation of objects, most VP languages are complemented by a sophisticated integrated development environment (IDE) that “…make the coding aspect of programming far more efficient, allowing the programmer to concentrate on higher level issues of program design, efficiency and elegance” (Pea and Kurland, 1984). A possible interpretation in terms of the reference to “higher level issues” is that students could now concentrate on the minimisation of structural errors knowing that the presentation of the solution is largely handled by the VP environment. According to White and Sivitanides (2002), the strategy of using VP as a tool to soften the abstract cognitive demands of programming will greatly favour students who are classified as concrete operational or pre-formal (in reference to the Piagetian framework for cognitive development). This visual strategy is aptly illustrated by the Visual Studio IDE released by the Microsoft Corporation. This line of thought has been extended to the server based Web environment. Currently, technologies such as Active Server Pages (ASP) and Hypertext Preprocessor (PHP) have dominated as the main server-side programming languages. However, the abstractionism inherent in these technologies has increased the complexity of the notional machine for novice programmers, thereby increasing the cognitive barrier of entry into server-side programming. In an attempt to reduce the abstractionism inherent in server-side programming, the release of the ASP.Net technology has heralded a transformation in the mechanism of server side software. The consequence has been the lowering of barriers of entry into the “arena” of software development for the general user. However, according to White and Sivitanides, there is a dearth of empirical evidence to

[16] The use of graphical user interfaces (GUI) for operating systems such as Windows from Microsoft, Linux with its X Windows system, and Apple Macintosh with its OS X

support a hypothesis of this nature making the intervention of VP as a viable avenue for further research.

6.3.4 A Curricular Strategy

A possible curricular strategy (currently being employed at UNZN in the School of IS&T) could entail the following:

- The initial stages of introductory programming establish a pedagogical focus on the reduction of structural and executive errors. In order to achieve this, programming concepts and constructs should be given exclusive focus without being diluted by issues such as development environment, interface and compiler and requirements. The main focus should be on problem solving and programming constructs. The syntactic and technical requirements of the development environment should be given an unobtrusive status. In terms of current technology, it is the opinion of the researcher that a scripting language such as Javascript, would provide the ideal platform to achieve this.
- The introduction of a more professional computer programming development environment is incumbent upon the establishment of a structurally secure foundation for problem solving and programming semantics. However, in terms of cognitive progression, there is still every possibility that a high proportion of students are in the late concrete, early formal stage of cognitive development. In order to simplify the transition to higher levels of abstractionism, VP would ideal. VP incorporates elements of concrete manipulations that would immediately place the concrete operational students at ease. The modern VP interfaces such as Microsoft's Visual Studio and Borland's Builder environment is also fully object oriented thereby enhancing the challenge posed to those students who have achieved full formal cognitive development.

REFERENCES

Ajideh, P. (2003), "Schema Theory-Based Pre-Reading Tasks: A neglected Essential in the ESL Reading Class", *The Reading Matrix*, **3**(1) [online at http://www.readingmatrix.com/articles/ajideh/article.pdf , accessed at 4th April 2007]

Ali, S. (2005), "Effective Teaching Pedagogies for Undergraduate Computer Science", *Mathematics and Computer Education*, **39**(3), 243-257

Allert J., (2004), "Learning Style and Factors Contributing to Success in an Introductory Computer Science Course", *Proceedings of IEEEInternational Conference on Advanced Learning Technologies*, IEEE Computer Society, 385 -389

Barker, R. and Unger E.A., (1983), "A predictor for success in an introductory programming class based upon abstract reasoning development", *Proceedings of the fourteenth SIGCSE technical symposium on Computer science education,* 154 - 158

Bennedsen, J. and Caspersen, M.E., (2005), "Revealing the programming process", *Proceedings of the 36th SIGCSE technical symposium on Computer science education* , Vol. **37**, 186-190

Bergin, S. and Reilly, R. (2005), "Programming: Factors that influence Success", *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, 411-415

Berglund, A., Daniels, M. & Pears, A., (2006), "Qualitative research projects in computing education research: An Overview", *ACM International Conference Proceeding Series*; **165**, 25-33

Bishop-Clarke, C. (1995), "Cognitive style, personality, and computer programming", *Computers in Human Behaviour*, **11**(2), 241-260

Boeree, G.C., (1999) Jean Piaget: Personality Theories, http://www.social-psychology.de/do/pt_piaget.pdf , accessed: 12th February 2007

Booth, S., (2001), "Learning to program as entering the datalogical culture: A phenemenographic exploration", presented at the 9th European Conference for Research on Learning and Instruction, Fribourg, Switzerland

Booth, S. & Morton, F. (1997) *Learning and Awareness* Lawrence Erlbaum Associates Publishers ISBN: 0805824553

Bransford, J., Sherwood, R., Vye, N., & Rieser, J. (1986), "Teaching thinking and problem solving", *American Psychologist*, Vol. **41**, 1078-1089

Burton, P. and Bruhn, R. (2003) "Teaching Programming in the OOP Era", *SIGCSE Bulletin*, ACM Publishers, **35**(2), 111-114

Byrne, P and Lyons, G., (2001) "The effect of student attributes on success in programming", *Sixth annual conference on Innovation and technology in computer science education*, 49-52.

Campbell, P.F. and McCabe, G.P., (1984), "Predicting the success of freshmen in a computer science major", *Communications of the ACM*, **28**(3), 37-382

Chmura, A. (1998) "What abilities are necessary for success in computer science?", *ACM SIGCSE*, 30(4), 55-58

Cope, C. and Horan, P. (1998), "Toward an understanding of teaching and learning about information systems", *ACM International Conference Proceeding Series*; Vol 3, 188 – 197

Davis, R.B. (1984). *Learning Mathematics, The Cognitive Science Approach to Mathematics Education*. Kent:Croom-Helm Ltd.

Dehnadi, S. (2006), "Testing Programming Aptitude", Presentation at the PPIG Annual Conference, Brighton [online at: http://www.cs.mdx.ac.uk/research/PhDArea/saeed/S_Dehnadi_ppij-2006__2.pdf , accessed: 8th February 2007]

Dijkstra, E.W. (1972), "The Humble Programmer", *Communications of the ACM*, **15(**10), 859-866

Donaldson, M. (1963) *A study of children's thinking*. London: Tavistok Publications.

Du Boulay, B., O'Shea, T. and Monk, J. (1999), "The black box inside the glass box: presenting computing concepts to novices", *International Journal of Human-Computer Studies*, **51**(2), 265-277

Eckerdal, A., Thune, M. and Berglund, A., (2005), "What does it take to learn 'programming thinking'?", *Proceedings of the 2005 international workshop on Computing education research*, 135 – 142, ISBN:1-59593-043-4

Efopoulos, V., Dagdilelis, V., Evangelidis and G. Satratzemi, M. (2005) "WIPE: a programming environment for novices", *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, ACM Press, 113-117

Evans, G. and Simkin, M (1989), " What best predicts computer Proficiency?", *Communications of the ACM*, **32**(11), 1322-1327

Farrell, J., (2006), *An object-oriented approach to programming logic and design*, Thomson publishers London

Goodlad, J. Keating,P., (1990) *Access to knowledge: An Agenda for our Nation's Shools*, 145-157, ISBN: 0-87447-330-6

Gruba, P., Moffat, A., Søndergaard, and Zobel, J. (2004), "What Drives Curriculum Change?" *Proceedings of the 6th conference on Australian Computing Education*, Australian Computer Society, **30**, 109-117

Hazzan, O., Dubinsky, Y., Eidelman, L., Sakhnini, V. and Teif, M. (2006) "Qualitative Research in Computer Science Education", *ACM SIGCSE Bulletin* **38**,(1)

Hoepfl, M.C., (1997) "Choosing Qualitative Research: A Primer for Technology Education Researchers", *Journal of Information Technology* **9**(1), 47-63

Hughes, J. and Peiris, J. (2006), "Assisting CS1 students to learn: learning approaches and object-oriented programming", *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, 275-279

Konvalina, J. Wileman, S.A. and Stephens, L.J. (1983), "Math proficiency: a key to success for computer science students", *Communications of the ACM*, **27**(11), 1108-1113

Kurtz, B., (1980), "Investigating the relationship between the development of abstract reasoning and performance in an introductory programming class", *Proceedings of the 11th SIGCSE Technical Symposium on Computer Science Education*, 110 – 117 , ISSN: 0097-8418

Lawson, A.E. (1983), "The Acquisition of formal Operational Schemata during Adolescence: The Role of the Biconditional", *Journal of research in Science Teaching*, **20**(4), 347-356

Leongson, J. and Limjap, A., (2003), "Asessing the Mathematics Achievement of College Freshmen using Piaget's Logical Operations", Presentation at

the Hawaii International Conference on Education, Waikiki, [online at www.cimt.plymouth.ac.uk/journal/**limjap**.pdf , accessed 14th May 2007]

Lewandowski, G., Gutschow, A., McCartney, R., Sanders, K. and Shinners-Kennedy, D. (2005), "What novice programmers don't know", *Proceedings of the 2005 international workshop on Computing education*, 1-12

Lind, D., Marchal, G. and Wathan, S. (2005), *Statisitcal Techniques in Business Economics*, McGraw-Hill, NY

Lubbe, S. & Kloppe, R. (2004), *Introduction to Research Design – An Interdisciplinary Approach*, Dolphin Cost Publishers, Durban, South Africa

Mancy, R. (2006), "Using Interviews to investigate implicit knowledge in computer programming", *Proceedings of the 7th International Conference on Learning Sciences*, 460-466, Bloomington, Indiana

Mayer, D. B. and Stalnaker, A.W. (1968), "Selection and evaluation of computer personnel – the research history of SIG/CPR", *Proceedings of the 1968 ACM National Conference (23rd ACM National Conference)*, 657-670

Mayer, R., Dyck, J. and Vilberg, W., (1986), "Learning to program and learning to think: what's the connection?", *Communications of the ACM*, **29**(7), 605 - 610

Marton, F. and Säljö, R. (1976), "On qualitative differences in learning: Outcome and Process", *British Journal of Educational Psychology*, **46,** 4-11

McCracken, M.W., Tew, A.E. and Guzdial, M. (2005), " Impact of alternative introductory courses on programming concept understanding", *Proceedings of the 2005 international workshop on Computing education*, 25 – 35, ACM Press

Murnane, J.S., (1993), "The psychology of computer languages for introductory programming courses", *New Ideas in Psychology*, 11(2), 213-228

Naidoo, R. (1996) "Errors made in Differential Calculus by First Year Students at a University", MPHIL Thesis UWC

Owen, S., Budgen, D. and Brereton, P., (2006), "Protocol analysis: a neglected practice", *Communications of the ACM*, **49**(2), 117-122

Pea, D.R. and Kurland, M., (1984), "On the Cognitive effects of learning Computer Programming" , *New Ideas in Psychology*, **2**(2), 137-168

Petre, M. and Blackwell, A., (1997), "A glimpse of expert programmers' mental imagery", *Proceeding of the 7th workshop on Empirical studies of programmers*, 109-123, ISBN: 0-89791-992-0

Piburn M. (1989), "Key to the Propositional Logic Test", Unpublished document, available at http://www.cs.utexas.edu/users/almstrum/PLT/key.pdf , accessed 8th January 2007

Pillay, N. and Jugoo, V.R., (2005), An investigation into student characteristics affecting novice programming performance, *ACM SIGCSE Bulletin archive*, **37** (4), 107-110

Pioro, T.B., (2006) "Introductory computer programming: gender, major, discrete mathematics, and calculus", *Journal of Computing Sciences in Colleges*, **21(**5), 123-129

Rhem, J., (1995), "Deep/Surface Approaches To Learning: An Introduction", The National Teaching and learning forum [online at http://www.ntlf.com/html/pi/9512/article1.htm , accessed 2nd February 2007]

Roberts, E. (2004), "Resources to Support the Use of Java in Introductory Computer Science", *Proceedings of the 35th SIGCSE Technical Symposium on Computer science Education* , ACM Press, 233-234

Rose, E., Le Heron, J., and Sofat, I. (2005), "Student Understanding of Information Systems Design, Learning and Teaching: A Phenomenographic Approach", *Journal of Information Systems Education*, **16**(1), 183-195

Scanlan, D.A. (1989), "Structured Flowcharts outperform pseudocode: An experimental comparison", *IEEE Software*, **6**(5), 28-36

Schach, S.R., (2007) *Object-Oriented & Classical Software Engineering*, McGraw-Hill, New York

Schwebel, M. (1975), "Formal operations in first year college students", *Journal of Psychology*, Vol **91**, 133-141

Sebesta, R., (2006), *Concepts of Programming Languages* (7th Ed.) Pearson Education, USA

Shackleford, R. (1998), *Computing and Algorithms,* Addison Wesley Longman, USA

Shackleford, R. and Badre, A. (1993) "Why can't smart students solve simple programming problems?", *International Journal of Man-Machine Studies* **38** (6), 985 - 997

Sheil, B.A., (1981), "The Psychological Study of Programming", *ACM Computing Surveys (CSUR) archive*, **13**(1), 101 - 120

Shertz, J. and Weiser, M. (1983), "Programming problem representation in novice and expert programmers," *International Journal of Man-Machine Studies*, Vol. **19**, 391-398

Shneiderman, B. (1982), "Control flow and data structure documentation: two experiments", *Communications of the ACM*, **25**(1) , 55-63

Shneiderman, B. and Mayer, R. (1979), "Syntactic/semantic interactions in programmer behavior: A model and some experimental results", *The International Journal of Computer and Information Sciences*, Vol. **8**, 219-238

Simon, S., Fincher, S., Robins, A., Baker, B., Box, I., Cutts, Q., de Raadt, M., Haden,P., Hamer, J., Hamilton, M., Lister, R., Petre, M,. Sutton,K., Tolhurst, D. and Tutty, J (2006), "Predictors of success in a first programming course", *Proceedings of the 8th Austalian conference on Computing education*, Vol **52,** 189 - 196

Smith, D.C.,Cypher, A. and Schmucker, K. (1996), " Making programming easier for children (Bridging the gap between programming and People)", *Communications of the ACM*, 3(5), 58-67

Slavin, R. (1991) *Educational Psychology: Theory into Practice* (3rd Edition), Prentice Hall, ISBN: 0-13-237751-9

Strauss, A., and Corbin, J. (1990) *Basics of Qualitative Research: Grounded theory procedures and techniques*, Newbury Park, CA: Sage Publications Inc

Tukianinen, M. and Monkkonen, E., (2002), "Programming Aptitude as a prediction of learning to program", *Proceedings of the 14th Workshop of the Psychology of Programming Interest Group*, Brunel University

Wankat, C. & Oreovicz, S. (1993) *Models of Cognitive Development: Piaget and Perry* McGraw-Hill

Warren, P., (2001), “Teaching Programming Languages using scripting languages”, *Journal of Computing Sciences in Colleges*, **17**(2), 205-216

Wason, P. and Johnson-Laird, P. (1972) *Psychology of Reasoning: Structure and Content*. Cambridge: Harvard University Press

White, G. and Sivitanides, M. (2002), “A theory of the relationships between cognitive requirements of computer programming languages and programmers' cognitive characteristics” , *Journal of Information Systems Education*, **13**(1) 59-66.

Wiedenbeck, S. (2005), “Factors affecting the success of non-majors in learning to program”, *Proceedings of the 2005 international workshop on Computing education research*, Seattle, WA, USA, 13 - 24

Wilson, B.C. and Shrock, S. (2002) “Contributing to success in an introductory computer science course: A study of twelve factors”, *Proceedings of the 32th SIGCSE Technical Symposium on Computer Science Education*, ACM Press, NY, 184-188.

Wolfe, J.M., (1971), “Perspectives on testing for programming aptitude”, *Proceedings of the 1971 Annual Conference of the ACM*, 268-277

Appendix A

STUDENT INTERVIEW – COMPUTER PROGRAMMING

PART A

1. Please provide the following personal details:

   a. Age:_____

   b. Gender: Male □ Female □

   c. Home Language: _____

2. How would you describe your experience as a computer programmer before taking the ISTN212 course?

   None □ Minimal □ Intermediate □ Extensive □ Expert □

3. What is your outlook towards computer programming? You may denote your response according to one or more of the following categories, or you may answer in your own words.

   Dull □ Not much interest □ Required as part of your curriculum □ No intention of studying programming any further □ Challenging □ Exciting □ Intend to continue with the study of programming □

PART B

1. What ability or skill do you think is most important in order to be successful in programming? You may choose to answer the question in your own words, or select from the alternatives provided.

   Be able to master the syntax of a programming language □

   The ability to think logically □

   The ability to debug a program effectively □

   The ability to design a solution for a program □

   The ability to manipulate technology □

2. Consider the following programming statement:

   X = X + 1

   Offer a comment on the logic of this statement.

3. Do you think that a variable should have a data type (such as Integer, Real, Character…).Why?

4. Describe the range of numbers that will make the following condition true.

   (Temp<10) AND NOT (Temp<5)

5. How do you decide when to use an array?

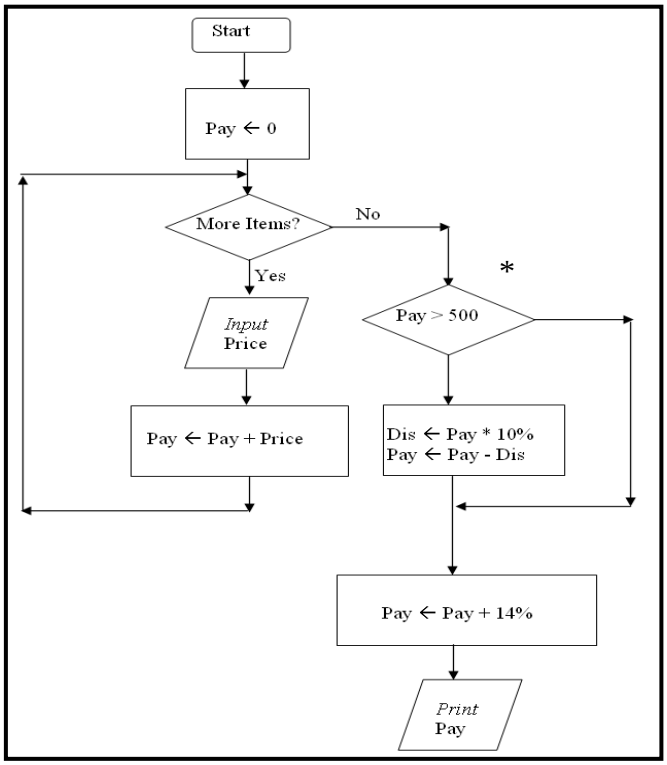
6. Which of the following is the most severe type of error encountered when writing programs? Provide motivation for your answer.

   Syntax error (compiler constantly flashes an error message at you) □

   Run-time error (error that causes the computer to stall when certain types of inputs are entered) □

   Logical error (error that results in incorrect output/answers) □

7. Examine the flowchart below and answer the questions that follow:



(a) Which of the following looping structures would be most appropriate for this flowchart? Provide an explanation for your choice of answer:

- For … Loop □
- While…Loop □
- Do/Repeat …Loop □

(b) The processing flow of control at the Selection structure (*) is missing. Explain what this should be?

PART C

1. If there are three variable, value pairs such as:

   A=10 B = 20 C = 30

   Explain how you would switch their values around so that the variables have the following final values:

   A=30 B=10 C=20

2. A company has information of their employees in three (3) different lists. You have obtained all three (3) lists, which all have different information depending on the purpose of the list. The lists have the following information:

   (a) List 1 : the employee number, employee name, job description and name of department
   (List 1 is ordered by the employee number in ascending order)

   (b) List 2 : the employee name, the employee number, employee address, employee phone number and employee identity number
   (List 2 is arranged alphabetically according to the employee name)

   (c) List 3: the employee number, the employee identity number, employee annual salary,

   Your job is to make a report of those employees whose annual salary is greater than R100 000. The report has to display the employee name, employee address and the job description of the employee.

   Describe how you would solve the problem.

3. You are required to determine the sum of fifty (50) numbers that are input by the user. You are also required to determine the number of numbers that were positive. Using pseudocode/ verbal / written explanation, describe how you would solve the problem

_____
_____
_____
_____
_____
_____
_____

4. You are given the task of determining the smallest number in a set of unequal numbers. What is the least number of comparisons you would have to make in each of the following cases? You are at liberty to ask any questions to clarify the requirements of this task.

   You have:

   (a) 3 numbers _____

   (b) 5 numbers _____

   (c) N numbers _____ (N >=1)

5. You are given the task of searching the telephone directory provided and finding the telephone number for the “University of South Africa”. Provide a concise description of the strategy that you used to achieve a solution for this problem.

Appendix B

The Cognitive Ability Test with exemplars

COGNITIVE ABILITY TEST

30 MINUTES

Please provide the following details:

a. Name: _____

b. Student Number: _____

c. Gender: Male □ Female □

Confidentiality Statement

The results of this test will be used exclusively for the purpose of research and there will be no direct reference to any individual who has provided responses to questions in this test. Responses will be referenced in an aggregate manner, the primary purpose of which is a contribution to the body of knowledge on the teaching and learning of computer programming.

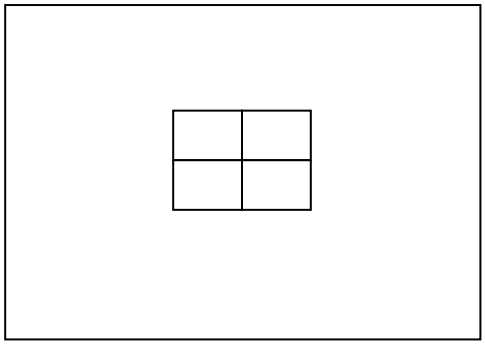
Instructions

You are required to complete the following logical skills test by choosing an appropriate alternative by placing a tick ✓where applicable, or by providing a written response. If the question does not have a definite answer, you may respond by writing the word “impossible”. You will be allocated 30 minutes to complete the test. You are expected to engage in a comprehension of the questions in order to provide the best possible response. You may show all working on the question paper.

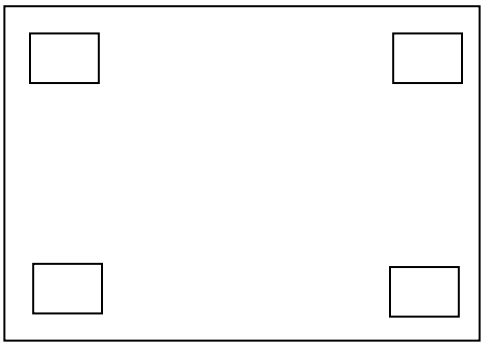
Question One

Two identical four cornered rooms, labeled “A” and “B” are shown below. Each room has 4 identical blocks placed on the floor as shown in the diagram.

A



B



In which of the rooms do the blocks cover a greater surface area.

Room A	☐
Room B	☐
The surface area covered by the blocks is the same in both rooms	☐
Impossible to determine	☐

Question Two

Suppose that you are investigating the running abilities of a horse and a dog. You find that each time the horse takes a step the dog also takes a step. You measure the stride of the horse and find that it is 3 metres long. The horse can run a particular course in 30 seconds. If the dog has a ½ metre stride, how long will it take the dog to complete the course?

Question Three

Suppose that there are four (4) islands, named Bean, Bird, Fish and Snail. The following rules would determine the possibility of navigation between the islands.

- There is a way to fly between Bean Island and Bird Island.
- There is no way to fly between Bird Island and Snail Island.
- There is a way to fly between Bean Island and Fish Island

Is there a way to fly between Fish Island and Snail Island?

Yes	□
No	□
Not enough information	□

Provide an explanation for your answer.

Question Four

Suppose that there are four (4) stores - a barber shop represented by the letter A, a discount store represented by the letter B, a grocery store represented by the letter C and a coffee shop represented by the letter D. These stores have to be arranged side-by-side on the ground floor of a shopping center. Use the place-holders below to illustrate all the possible ways in which the stores can be arranged.

□ □ □ □	□ □ □ □	□ □ □ □
□ □ □ □	□ □ □ □	□ □ □ □
□ □ □ □	□ □ □ □	□ □ □ □
□ □ □ □	□ □ □ □	□ □ □ □
□ □ □ □	□ □ □ □	□ □ □ □
□ □ □ □	□ □ □ □	□ □ □ □
□ □ □ □	□ □ □ □	□ □ □ □
□ □ □ □	□ □ □ □	□ □ □ □
□ □ □ □	□ □ □ □	□ □ □ □
□ □ □ □	□ □ □ □	□ □ □ □

Question Five

There are three (3) blue chips and seven (7) red chips that are placed in a container labelled A. There are four (4) red chips and two (2) blue chips placed in a container labelled B. Which container would you choose to have the best chance of drawing out a blue chip on the first try?

Container A	□
Container B	□
The chance of drawing out a blue chip is the same for both containers	□

Provide an explanation for your answer.

Question Six

In a photograph, a mother is 8cm high and her daughter is 6cm high. If the picture is enlarged so that the daughter is 15cm high, predict the height of the mother in the enlarged photograph.

Question Seven

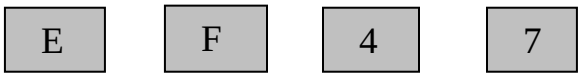
There are a total of 16 birds in an aviary. Six (6) of the birds have long beaks and short tails, two (2) of the birds have short beaks and short tails, two (2) of the birds have long beaks and long tails and six (6) birds have short beaks and long tails. Is there a relationship between the length of the beak and the length of the tail? Provide a motivation for your answer.

Question Eight

Consider the following rule about a set of cards that have letters on one side and numbers on the other side.

“If a card has a vowel on one side, then it has an even number on the other side.”

Look at the cards below and determine which cards need to be turned over to verify if this rule is true?



Write down your response to this question.

Question Nine

You are required to test the truth or falsity of the following hypothesis:

"If a computer has a Pentium processor, then it will be fast."

Examine each of the scenarios depicted below and respond by indicating whether information obtained would be useful in helping you prove/disprove the hypothesis above.

[1] Given computers with Pentium processors, would you need to know if these computers were fast or slow?

YES, this information would be necessary in helping me prove/disprove the hypothesis	☐
NO, this information would not be necessary in helping me prove/disprove the hypothesis	☐

[2] Given computers with no Pentium processors, would you need to know if these computers were fast or slow?

YES, this information would be necessary in helping me prove/disprove the hypothesis	☐
NO, this information would not be necessary in helping me prove/disprove the hypothesis	☐

[3] Given several fast computers, would you need to know if they have Pentium processors?

YES, this information would be necessary in helping me prove/disprove the hypothesis	☐
NO, this information would not be necessary in helping me prove/disprove the hypothesis	☐

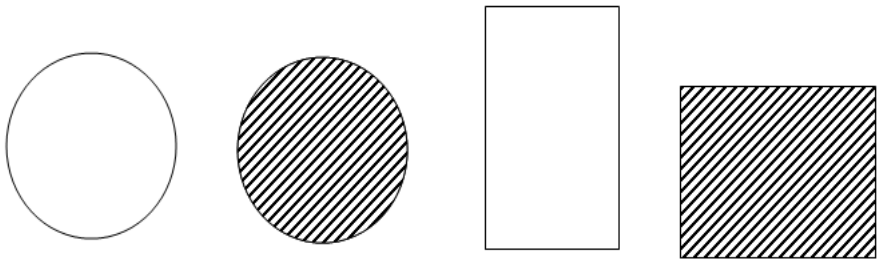
[4] Given several slow computers, would you need to know if they have Pentium processors?

YES, this information would be necessary in helping me prove/disprove the hypothesis	□
NO, this information would not be necessary in helping me prove/disprove the hypothesis	□

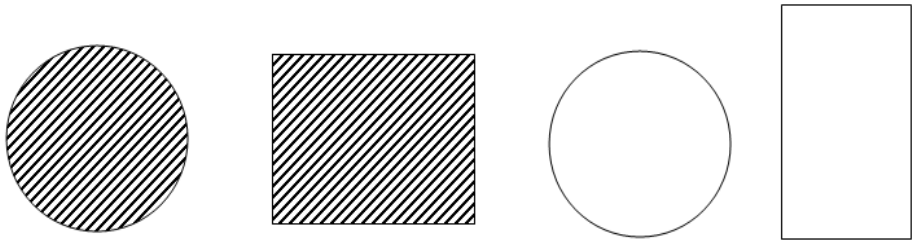
Question Ten

Using an X, strike off those symbols that do not satisfy the following logical rules:

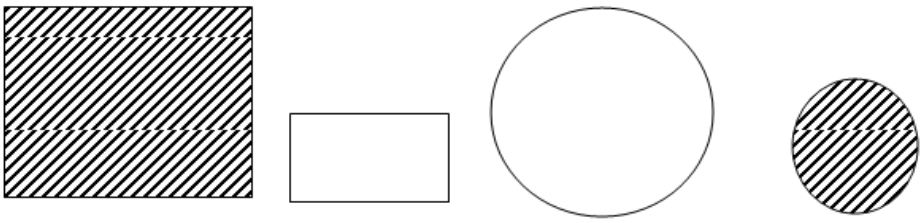
[1] It is round and it is striped



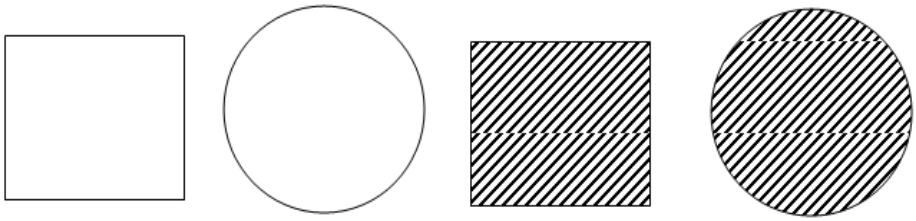
[2] It is round or it is striped



[3] If it is large then it is round

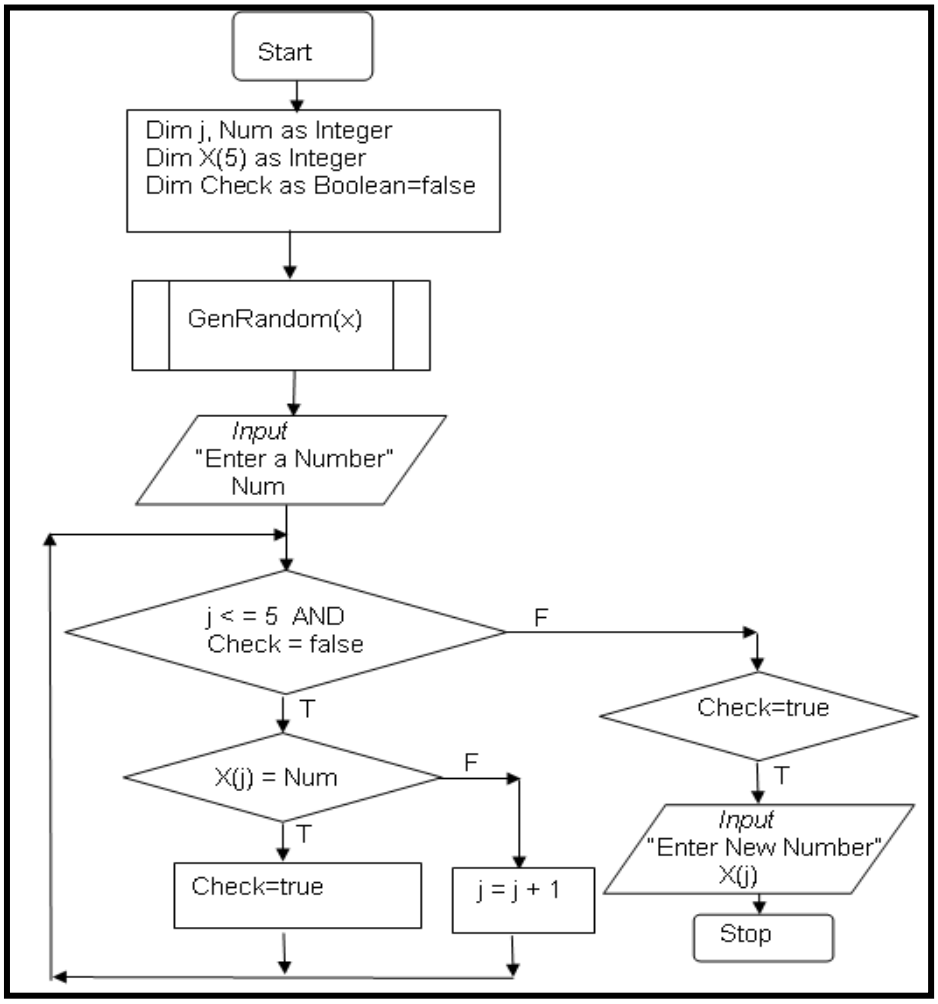


[4] If it is round it is striped and if it is striped it is round



Appendix C

QUESTION ONE



1.1 Identify an example of each of the following: [2 marks]

1.1.1 Flag
Check

1.1.2 **Counter**
J

1.2 The sub-routine "GenRandom(x)" serves the purpose of generating random numbers in the range from 1 to 100 into an Integer array. Write the VB code for this sub-routine. [6 marks]

```
Public Sub GenRandom(ByVal intArray() As Integer)✓
        Dim intI As Integer ✓
        For intI = 0 To intArray.GetUpperBound(0) ✓
            intArray(intI) ✓ = Int(Rnd() * 100) + 1✓
        Next✓
End Sub
```

1.3 Give a brief explanation describing the purpose of the flowchart. [2 marks]

The flowchart is designed to accept user input, search✓ for the number entered and facilitate editing✓ of the original number

1.4 Write the code fragment to implement the flowchart. Your solution should include a reference to the sub-routine written in Question 6.2 above. (Note: A code fragment must entail declaration of all variables as well as the use of an "inputbox" to handle all input and a "messagebox" to handle all output, if required) [8 marks]

```
Dim j, Num As Integer
Dim X(5) As Integer
Dim Check As Boolean
GenRandom(X)
Num = InputBox("Enter a Number")✓
Do While✓ (j <= 5) And (Check = False) ✓
   If X(j) = Num Then✓
      Check = True
   Else                  } ✓
      j = j + 1
   End If
Loop ✓
If Check = True Then ✓
   X(j) = InputBox("Enter a Number") ✓
End If
End Sub
```

Alternative loping structure would be Do..Loop Until ; For…Next with an "Exit For" statement if Check becomes true.

QUESTION TWO (27 marks)

2.1 The following logically incorrect and inefficient sub-routine was written in order to sort an array of integers into ascending order. Rewrite the code so that the sub-routine would be successful in sorting of the array.

```
Public Sub Sort(ByVal intArr As Integer)as Integer
 Dim i, j, maxSize As Integer
 maxSize = intArr.length
 For i = 0 To maxSize
   For j = 1 To maxSize -2
      If (j) < intArr(j + 1) Then
         Dim Temp As Integer
         intArr(i)=Temp
         intArr(j) = intArr(j+1)
         intArr(i + 1) = Temp
      End If
     Next
  Next
```

[6 marks]

```
Public Sub Sort(ByVal intArr As Integer)as Integer
 Dim i, j, maxSize As Integer
 maxSize = intArr.GetUpperBound(0) ✓
 For i = 0 To maxSize
   For j = 0 To maxSize – 1 ✓
    If intArr(j) ✓ >✓ intArr(j + 1) Then
      Dim Temp as Integer
      Temp = intArr(j)
      intArr(j) = intArr(j + 1)      ✓✓
      intArr(j + 1) = Temp
    End If
   Next
 Next
 End Sub
```

OR

```
Public Sub SelectSort(ByVal intArr As Integer())
 Dim i, j, maxSize As Integer
 maxSize = intArr.GetUpperBound(0) ✓
 For i = 0 To maxSize – 1 ✓
  For j = (i + 1) To maxSize ✓
     If intArr(i) > intArr(j) Then ✓
       Dim Temp as Integer
       Temp = intArr(j)
       intArr(j) = intArr(j + 1)      ✓✓
       intArr(j + 1) = Temp
     End If
 Next
Next
End Sub
```

QUESTION THREE

3.1 The global declarations below have been used in an application that does an analysis of sales data for sales employees of a retail outlet.

```
Structure Sales
        Dim Name As String
        Dim TotalSales As Integer
End Structure
Dim Employees(9) As Sales
```

3.1.1 Write a procedure that would populate the array with data. The procedure must prompt the user the user for the name and total sales amount for each employee. The procedure stores this data in the array declared above. [5 marks]

```
Sub GetData(ByVal X() As Sales) ✓
 Dim i As Integer ✓
 For i = 0 To X.GetUpperBound(0) ✓
    X(i).Name = InputBox("Enter Name of Employee")
    X(i).TotalSales = InputBox("Enter Amount of
sales") ✓
 Next✓
 End Sub
```

3.1.2 Assuming that the total sales amount generated by each employee is different, write a function that will return the name of the employee who has achieved the highest sales. [8 marks]

```
Function GetHighest(ByVal X() As Sales) As String ✓✓
 Dim i As Integer
 Dim HighIndex As Integer✓
 For i = 1 To X.GetUpperBound(0) ✓
   If X(i).TotalSales > X(HighIndex).TotalSales Then
✓✓
      HighIndex = I ✓
   End If
 Next
 Return X(HighIndex).Name ✓
End Function
```

3.2 You have been assigned the task of providing a software solution for a businessman who would like to track sales for a period of 6 working weeks. A working week consists of 5 days. The businessman has requested that he be presented with a composite list showing the actual sales figures per day as well as the total sales per week.

3.2.1 Assuming that all sales data are recorded as Integers, provide a declaration for a two dimensional (2D) array that would be adequate to store the daily sales figures and the weekly totals. [2 marks]

```
Dim sales2D(5✓, 5✓) As Integer
```

3.2.2 Assuming that the sales data has been entered into the 2D array, write a sub-routine to compute the total sales recorded for each week. The results of this computation need to be stored in the 2D array. [6 marks]

```
Sub Get_Totals(ByVal x(,) As Integer) ✓
 Dim sum, i, j As Integer
 For i = 0 To x.GetUpperBound(0) ✓
    sum = 0 ✓
    For j = 0 To x.GetUpperBound(1) ✓
      sum += x(i, j) ✓
    Next
    x(i, 5) = sum ✓
  Next
 End Sub
```

……THE END……..

Appendix D

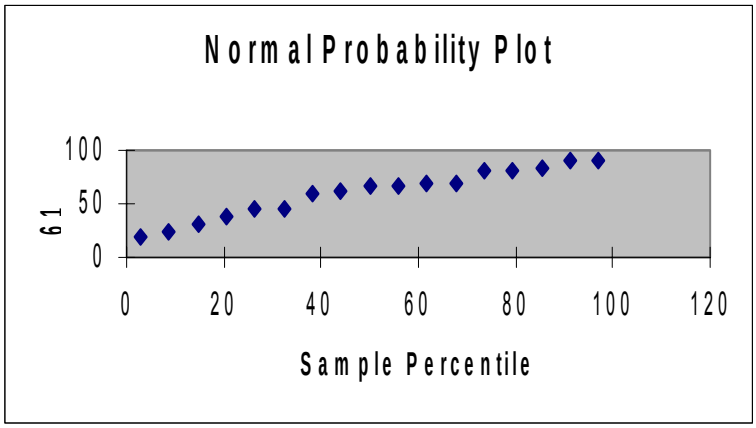
Responses given in the interviews were quantified by attaching a mark (out of 4) for the various categories of programming that were tested. A multiple-correlation with the mark obtained in the interviews was performed with the marks obtained by these students in the examinations. A frequency table shows the data generated from this exercise is presented below:

Student Number 1	Student Exam Mark (%)	Iteration (4)	Selection (4)	Algorithmic Thinking (4)	Array Processing (4)	Total Score (16)
1	**61**	3	3	2	2	10
2	**38**	3	3	3	2	11
3	**67.5**	3	4	3	3	13
4	**70**	3	4	3	3	13
5	**19**	2	3	2	1	8
6	**82**	4	2	3	3	12
7	**32**	2	2	1	1	6
8	**46**	1	2	1	1	5
9	**70**	3	4	2	2	11
10	**62**	3	2	1	2	8
11	**90**	4	4	4	4	16
12	**82.5**	4	4	2	3	13
13	**46**	1	1	1	1	4
14	**80**	4	4	3	3	14
15	**24**	2	3	1	1	7
16	**67**	4	4	4	4	16
17	**90**	4	4	3	3	14
18	**60**	3	3	2	3	11

Notes:
The interviews were analysed according to the headings above
Each category carried a total of 4 marks
The 4 categories totalled 16 marks

The correlation for “selection” and “algorithmic thinking” with the final examination mark was poor. However, there was a strong correlation between performance in “iteration” and “array processing” to the final examination mark. A “snapshot” of the statistical analysis is presented:

EXAM SCORE= 13.638+5.959*ITERATION TASK SCORE+12.517*ARRAY TASK SCORE ($p<0.05$)
The iteration task score and array task score appear to predict the exam score at ($p<0.05$)
The normality test: passed ($p=0.426$)
Constant Variance Test: Passed ($p=0.392$)
Power of performed test with alpha=0.050:0.997



The implication from the data presented suggests that performance in iteration and array processing questions are a reliable predictor of examination performance. However, this line of enquiry could be replicated on a greater scale to confirm the validity of the conclusion drawn.

Appendix E

Part One. Ethical Clearance Certificate

UNIVERSITY OF
KWAZULU-NATAL

Sanjay Ranjeeth
ranjeeths@ukzn.ac.za
Student Number: 205186614

RE: PERMISSION TO CONDUCT RESEARCH

Dear Professor Brian McArthur

I intend conducting research into the computer programming ability of university students. The area of focus is on the cognitive domain and its impact on the mastery of computer programming skills. Hopefully the emergent discourse will have a positive effect from a pedagogical perspective, thereby enriching the learning experience of students from the School of Information Systems and Technology (IS&T) at the University of KwaZulu-Natal(UKZN).

I would like to hereby request your permission in allowing me access to the 2nd year and 3rd year Information Systems students. The focus of these contact sessions would be to ascertain the cognitive abilities of these students in relation to their performance in programming assessment tasks. Every effort would be made to ensure that this research agenda would not cause any disruption to the academic programmes of the university.

DECLARATION

I BRIAN MCARTHUR (DEAN HSS) (full name and capacity) do hereby confirm that I understand the contents of this document and the nature of the research agenda as proposed by Mr S Ranjeeth. I consent to allowing him to engage the 2nd and 3rd year students from the School of IS&T at UKZN as subjects in this research project.

I understand that I am at liberty to withdraw my consent, as stipulated in this document, at my discretion.

SIGNATURE OF AUTHORITY **DATE**

..... 25-01-2007

School of Information Systems and Technology
Private Bag X01, Scottsville 3209, South Africa. Tel: (033) 260 5704 Fax: (033) 260 5648 e-mail: brianm@ukzn.ac.za

Part Two. Student Consent Form

THE EFFECT OF COGNITIVE ABILITY ON A STUDENT'S APTITUDE FOR COMPUTER PROGRAMMING AT A TERTIARY INSTITUTE IN KWAZULU-NATAL

To Whom It May Concern:

The aim of this research is to gauge the impact of one's cognitive ability on the ability to acquire competence in computer programming. The reason for this research effort is that there is significant evidence to suggest that there is a problem with the teaching and learning of computer programming. This statement is made in the light of high failure rates in programming courses at the university. It is hoped that cognisance of the findings of this study will be taken by instructors of programming courses. The investigation will highlight students' perceptions and opinions with the desire of incorporating these into future teaching plans.

It should be pointed out that participation in this study is purely voluntary and you are free to withdraw from the study at any stage and for any reason. Your participation would, however be greatly appreciated. All information will be kept strictly confidential and no information relating to you will be made public. As a strategy, only aggregated data will be presented and any comments made by you will remain anonymous.

In terms of your participation, you will be required to participate in an interview of approximately 30 minutes. In order to ensure accurate capturing of your response, an audio recording of the interview will be made. You will also be required to complete a cognitive ability test that would take approximately 30 minutes. This data capturing exercise forms part of a Masters dissertation at the Durban University of Technology under the supervision of Dr Ramu Naidoo.

It should be pointed out that there is no obligation to answer all the questions and participants should feel free to leave out any questions they do not want to answer.

Sanjay Ranjeeth
c/o School of Information Systems and Technology
University of KwaZulu-Natal
ranjeeths@ukzn.ac.za
Tel: 0332605641

I..... (full names of participant) hereby confirm that I understand the contents of this document as well as the nature of the research project, and I consent to participating in the research project

Signature of Participant _____ Date _____

Appendix F

Part One. "Snapshot" of Interview data

Student Number	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Deep	Surface	Neutral	Deep	Surface	Neutral
1	deep	deep	deep	deep	neutral	deep	deep	deep	deep	deep	deep	neutral	10	0	2	Deep	Deep/Neutral	Deep
2	surface	surface	surface	deep	neutral	surface	surface	surface	surface	surface	surface	neutral	1	9	2	Surface/Neutral	Surface	Surface
3	deep	surface	surface	deep	surface	surface	surface	surface	surface	surface	surface	surface	2	10	0	Surface/Neutral	Surface	Surface
4	surface	surface	neutral	neutral	surface	surface	surface	surface	surface	surface	surface	surface	0	10	2	Surface/Neutral	Surface	Surface
5	deep	deep	deep	deep	deep	deep	deep	deep	deep	deep	surface	neutral	10	1	1	Deep	Deep/Neutral	Deep
6	deep	deep	surface	deep	surface	surface	surface	surface	surface	surface	surface	neutral	3	8	1	Surface/Neutral	Surface	Surface
7	surface	surface	surface	deep	surface	surface	surface	surface	surface	surface	surface	neutral	1	10	1	Surface/Neutral	Surface	Surface
8	deep	deep	deep	deep	deep	deep	deep	deep	deep	deep	surface	neutral	10	1	1	Deep	Deep/Neutral	Deep
9	deep	deep	surface	deep	surface	deep	surface	surface	neutral	surface	surface	surface	4	7	1	Surface/Neutral	Deep/Neutral	Neutral
10	deep	deep	surface	deep	surface	surface	surface	surface	surface	surface	surface	neutral	2	8	1	Surface/Neutral	Surface	Surface
11	deep	deep	deep	deep	deep	deep	deep	deep	deep	deep	deep	deep	12	0	0	Deep	Deep/Neutral	Deep
12	deep	deep	surface	deep	surface	deep	surface	deep	neutral	surface	surface	surface	5	6	1	Surface/Neutral	Deep/Neutral	Neutral
13	neutral	surface	surface	surface	surface	surface	surface	surface	neutral	surface	surface	surface	0	10	2	Surface/Neutral	Surface	Surface
14	deep	deep	neutral	deep	neutral	deep	deep	deep	neutral	neutral	deep	neutral	7	0	5	Deep	Deep/Neutral	Deep
15	surface	surface	surface	deep	surface	deep	surface	surface	neutral	surface	surface	neutral	2	8	2	Surface/Neutral	Surface	Surface
16	surface	deep	neutral	deep	neutral	surface	deep	deep	neutral	deep	surface	neutral	5	3	4	Surface/Neutral	Deep/Neutral	Neutral
17	deep	deep	deep	deep	neutral	deep	deep	deep	neutral	deep	deep	neutral	9	0	3	Deep	Deep/Neutral	Deep
18	surface	neutral	surface	deep	surface	deep	surface	surface	neutral	surface	surface	neutral	2	7	3	Surface/Neutral	Surface	Surface
19	deep	neutral	surface	deep	surface	deep	surface	surface	surface	surface	surface	neutral	3	7	2	Surface/Neutral	Surface	Surface
20	surface	surface	surface	deep	surface	surface	surface	surface	surface	surface	surface	neutral	1	10	1	Surface/Neutral	Surface	Surface
21	deep	deep	surface	deep	deep	deep	deep	deep	neutral	surface	surface	neutral	7	3	2	Deep	Deep/Neutral	Deep
22	surface	surface	surface	deep	surface	neutral	neutral	surface	neutral	surface	surface	neutral	1	7	4	Surface/Neutral	Surface	Surface
23	surface	surface	surface	surface	surface	neutral	surface	surface	surface	surface	surface	surface	0	11	1	Surface/Neutral	Surface	Surface
24	deep	neutral	surface	deep	neutral	deep	deep	deep	neutral	neutral	surface	surface	5	3	4	Surface/Neutral	Deep/Neutral	Neutral
25	surface	surface	surface	surface	surface	deep	surface	surface	surface	surface	surface	surface	1	11	0	Surface/Neutral	Surface	Surface
26	deep	neutral	neutral	deep	surface	deep	deep	deep	neutral	deep	surface	neutral	6	2	4	Deep	Deep/Neutral	Deep
27	surface	deep	surface	deep	surface	surface	surface	deep	neutral	surface	surface	surface	3	8	1	Surface/Neutral	Surface	Surface
28	deep	deep	surface	deep	deep	deep	deep	deep	deep	deep	surface	neutral	9	2	1	Deep	Deep/Neutral	Deep
29	deep	deep	surface	deep	surface	deep	deep	deep	surface	deep	surface	deep	8	4	0	Deep	Deep/Neutral	Deep
30	surface	deep	surface	deep	surface	surface	surface	deep	surface	surface	surface	neutral	3	8	1	Surface/Neutral	Surface	Surface
31	deep	deep	surface	deep	surface	surface	surface	surface	neutral	surface	surface	neutral	3	7	2	Surface/Neutral	Surface	Surface
32	surface	deep	surface	deep	surface	deep	surface	deep	surface	surface	surface	surface	4	8	0	Surface/Neutral	Surface	Surface
33	deep	deep	neutral	deep	surface	deep	surface	deep	neutral	deep	deep	deep	8	2	2	Deep	Deep/Neutral	Deep
34	deep	deep	neutral	deep	deep	deep	deep	deep	neutral	deep	deep	neutral	9	0	3	Deep	Deep/Neutral	Deep
35	deep	deep	neutral	deep	deep	deep	deep	deep	neutral	deep	deep	neutral	9	0	3	Deep	Deep/Neutral	Deep
																13	18	4

Part Two. Cognitive Test Results

Student No	Score	Maturity Level
1	11	Level Two
2	7	Level One
3	4	Level One
4	12	Level Two
5	7	Level One
6	8	Level Two
7	11	Level Two
8	13	Level Two
9	10	Level Two
10	9	Level Two
11	10	Level Two
12	3	Level One
13	13	Level Two
14	8	Level Two
15	7	Level One
16	16	Level Three
17	4	Level One
18	8	Level Two
19	3	Level One
20	4	Level One
21	12	Level Two
22	22	Level Three
23	6	Level One
24	10	Level Two
25	7	Level One
26	6	Level One
27	4	Level One
28	5	Level One
29	12	Level Two
30	18	Level Three
31	18	Level Three
32	9	Level Two
33	22	Level Three
34	4	Level One
35	11	Level Two

Student No	Score	Maturity Level
36	9	Level Two
37	9	Level Two
38	10	Level Two
39	8	Level Two
40	12	Level Two
41	3	Level One
42	13	Level Two
43	5	Level One
44	8	Level Two
45	11	Level Two
46	4	Level One
47	3	Level One
48	7	Level One
49	12	Level Two
50	7	Level One
51	11	Level Two
52	19	Level Three
53	6	Level One
54	8	Level Two
55	3	Level One
56	20	Level Three
57	14	Level Two
58	7	Level One
59	6	Level One
60	4	Level One
61	22	Level Three
62	9	Level Two
63	8	Level Two
64	6	Level One
65	10	Level Two
66	12	Level Two
67	16	Level Three
Average	**9.49254**	
Median	**9**	
Std Dev	**4.9094**	

Scores	*Frequency*
0..4	12
5	2
6	5
7	7
8	7
9	5
10	5
11	5
12	6
13	3
14	1
15	0
16	2
17	0
18	2
19	1
20	1
21	0
22	3

Part Three. Error and Cognitive Level Classification

	Student No	Score	Maturity Level	Arbitrary	Executive	Structural	Total Errors
1	12	3	Level One	2	2	4	8
2	19	3	Level One	1	3	5	9
3	41	3	Level One	2	4	5	11
4	47	3	Level One	1	2	6	9
5	20	4	Level One	1	3	4	8
6	27	4	Level One	2	2	5	9
7	34	4	Level One	2	1	3	6
8	46	4	Level One	1	4	4	9
9	60	4	Level One	1	3	5	9
10	28	5	Level One	3	2	5	10
11	43	5	Level One	4	1	2	7
12	23	6	Level One	2	3	4	9
13	26	6	Level One	1	3	3	7
14	53	6	Level One	1	4	4	9
15	59	6	Level One	2	2	2	6
16	64	6	Level One	4	3	4	11
17	2	7	Level One	3	1	3	7
18	50	7	Level One	2	2	2	6
19	58	7	Level One	3	3	2	8
20	6	8	Level Two	1	2	3	6
21	14	8	Level Two	1	1	2	4
22	44	8	Level Two	2	2	2	6
23	54	8	Level Two	1	1	2	4
24	32	9	Level Two	0	2	3	5
25	36	9	Level Two	0	0	1	1
26	37	9	Level Two	2	0	1	3
27	62	9	Level Two	1	1	2	4
28	24	10	Level Two	3	0	1	4
29	38	10	Level Two	2	2	2	6
30	65	10	Level Two	1	0	1	2
31	35	11	Level Two	1	1	1	3
32	45	11	Level Two	2	1	1	4
33	51	11	Level Two	0	0	2	2
34	4	12	Level Two	0	1	1	2
35	21	12	Level Two	1	2	2	5
36	40	12	Level Two	1	0	1	2
37	8	13	Level Two	2	0	0	2
38	13	13	Level Two	0	1	0	1
39	42	13	Level Two	1	2	1	4
40	57	14	Level Two	0	1	0	1
41	16	16	Level Three	2	1	1	4
42	67	16	Level Three	3	2	0	5
43	30	18	Level Three	4	1	0	5
44	22	22	Level Three	3	1	0	4
45	33	22	Level Three	3	0	0	3
46	61	22	Level Three	4	0	0	4