An Evaluation of Programming Assistance Tools to Support the Learning of IT Programming: A Case Study in South African Secondary Schools

by

Melisa Koorsse

Submitted in fulfilment of the requirements for the degree of Philosophiae Doctor in Computer Science at the Nelson Mandela Metropolitan University

Promoter: Prof C.B. Cilliers
Co-Promoter: Prof. A.P. Calitz

January 2012

Declaration

I, Melisa Koorsse (199201064), hereby declare that the thesis for my qualification to be awarded is my own work and that it has not previously been submitted for assessment or completion of any postgraduate qualification to another University or for another qualification.

M. Koorsse

Summary

Worldwide, there is a decline in interest in the computer science profession and in the subject at secondary school level. Novice programmers struggle to understand introductory programming concepts and this difficulty of learning to program is contributing to the lack of interest in the field of computer science. Information Technology (IT) learners in South African secondary schools are novice programmers, introduced to introductory programming concepts in the subject which also includes topics on hardware and system software, e-communication, social and ethical issues, spreadsheets and databases. The difficulties faced by IT learners are worsened by the lack of suitably qualified teachers, a saturated learning programme that allocates very little time to the understanding of complex programming concepts and limited class time where practical examples can be implemented with the support of the IT teacher.

This research proposes that IT learners could be supported by a *programming assistance tool* (PAT). A PAT is a software program that can be used by novice programmers to learn how to program and/or improve their understanding of programming concepts. PATs use different techniques to assist novice programmers. The main objective of this research was to determine whether the use of a PAT impacted IT learners' understanding of programming concepts and motivation towards programming.

The literature study and feedback from IT learners and teachers were used to identify novice programming difficulties and IT learner programming difficulties, respectively. Selection criteria were derived from the programming difficulties identified. The selection criteria were grouped into three categories, namely, programming concepts, programming knowledge and programming skills. Existing PATs were evaluated using the selection criteria and three PATs, namely, *RoboMind*, *Scratch* and *B#*, were selected as suitable for use by IT learners. *RoboMind* was adapted in this research study, allowing it to support the Delphi programming language. The three PATs were evaluated by participating IT learners at four schools.

The findings of this research provided no conclusive evidence that IT learners who used a PAT had a significantly better understanding of programming concepts and motivation towards programming than learners who did not use a PAT. IT learner feedback was used to identify the strengths and shortcomings of the three PATs and to provide recommendations for the development of PATs specifically to support IT learners.

This research study has provided several theoretical and practical contributions, including the research design, selection criteria, adaptations to *RoboMind* and the evaluation of the three PATs. In addition, IT teachers and learners have been made aware of PATs and the support that can be provided by these PATs. IT teachers have also been provided with a means of selecting PATs applicable to the IT curriculum. All the research contributions have formed the basis for future work, such as improving and extending *RoboMind's* functionality and support of programming concepts, the refinement of the selection criteria and, ultimately, the development of a new PAT, specifically designed to support IT learner understanding of programming concepts and motivation towards programming.

Acknowledgements

This research could not have been completed without the contributions of many different people, who have supported and assisted in this research in different ways. I would like to extend my appreciation to the following:

I would like to thank God for the perseverance and wisdom that He has bestowed upon me during this research, and throughout my life. The following words have been a constant source of encouragement:

> "*I can do all things through Christ who strengthens me.*" (Phillipians 4:13)

My promoters, Charmain Cilliers and André Calitz, for their invaluable guidance and continuous support for the duration of this research. Their advice and suggestions were always constructive and improved the quality of the work.

The participating schools and learners for their co-operation during this study. A special thank you to the IT teachers at the participating schools for their time and co-operation with regards to the administration of questionnaires and class tests. Their contribution has been invaluable to this research study.

The Department of Computing Sciences for allowing me to conduct this research and for providing ongoing research support.

Prof. Danie Venter and John Cullen for their time and assistance with statistical analysis and language editing, respectively.

Clayton Burger for giving his time to provide feedback and technical support.

Finally, I would like to thank my family, in particular my mother, for unconditional support and encouragement throughout a challenging time.

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

Dedications

This thesis is dedicated to my grandmother
Dorothy Enith Cunningham

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Background

The Information and Communications Technology (ICT) industry has become a major industry internationally. ICT is important for organisations that want to adapt existing business to new business opportunities (Cumps, Viaene and Dedene, 2010). ICT related technical positions such as Computer Systems Analyst, Technical Writer and Computer Programmer all rank within the top 30 professions (Strieber, 2011). The reason for the high rankings can be attributed to an increase in the number of companies developing web, mobile applications and cloud computing applications. Software engineering has been identified as the top profession in America in 2011. Evaluated using five criteria (Strieber, 2011), the software engineering profession offers a better than average income, a comfortable working environment, few physical demands, comparatively low stress and good hiring opportunities. Consequently, Computer Science remains a qualification in great demand in the United States (NACE, 2011). In South Africa, programming skills obtained through Computer Science and Information Systems degree programmes are just as sought after by industry (Calitz, 2010).

Despite the demand for ICT qualifications, the number of learners choosing a career in Computer Science is decreasing (Calitz, 2010; Gardner and Feng, 2010; Porta, Maillet and Gil, 2010; Wilson, Sudol, Stephenson and Stehlik, 2010). This crisis is shared by countries around the world, including the United States (Wilson *et al.*, 2010), European countries (Porta *et al.*, 2010) and South Africa (Calitz, 2010; Rogerson and Scott, 2010). In the U.S., the number of enrolments in Computer Science bachelor programs has decreased by 30% over the last decade (Egan, 2010). Research studies in Europe and North America indicate that, although the number of learners pursuing careers in the fields of science and technology is decreasing, the total number of enrolments in tertiary education institutions has increased (Porta *et al.*, 2010).

The apparent lack of interest in the fields of science and technology has resulted in a decrease in the number of software developers available. The shortage of software developers has increased the costs of software development. Governments in several countries around the world, including the U.S. (Wilson *et al.*, 2010), European countries (Porta *et al.*, 2010) and South Africa (Medium Term Strategic Framework, 2009), are working to promote interest in computing professions. The importance of equipping school learners with technological skills and knowledge for future success has also been recognised (Wilson *et al.*, 2010).

Computer Studies subjects at schools worldwide aim to develop the technological skills and knowledge of learners (Appendix H). However, these subjects are in crisis due to lack of qualified teachers, lack of quality instructional materials, and the difficulty of covering all the subject topics in adequate detail in the limited time available (Wilson *et al.*, 2010; Havenga and Mentz, 2009). In addition, the difficulties experienced when learning to program is resulting in high attrition rates, a decline in interest to continue with a career in computing and a general negative impression of programming amongst learners (Garner, 2007; Gomes and Mendes, 2007).

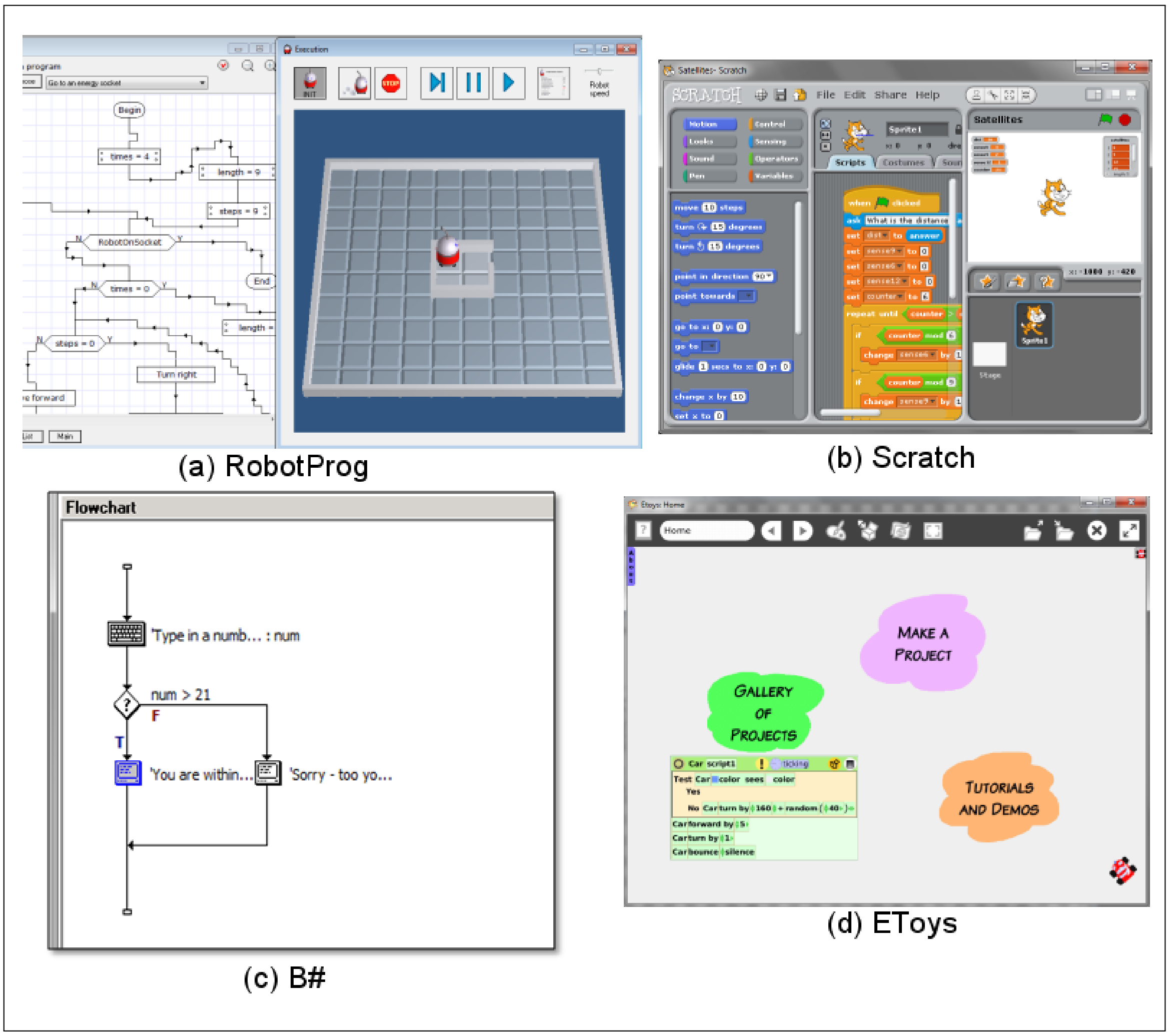
There is consensus that programming is a difficult skill to learn[1]. The problem solving and logical cognitive processes that expert or professional programmers possess (Baldwin and Kuljis, 2000; Al-Imamy, Alizadeh and Nour, 2006) are essential when developing software that is fast, scalable and can be adapted to technological advances. However, it takes an average of 10 years for a novice[2] to progress to the level of expert programmer (Robins, Rountree and Rountree, 2003). Novice programmers typically need concurrently to learn how to understand and solve a problem, formulate a solution in a structured form (algorithm) and write the algorithm in a programming language (Vickers, 2009). Inability to plan solutions (Rongas, Kaarna and Kalvianen, 2004), lack of understanding of programming concepts due to the abstract nature of these concepts (Lahtinen *et al.*, 2005) and a lack of understanding of how a computer works (Levy, Ben-Ari and Uronen, 2001) are some of the reasons identified why programming is difficult.

The situation in South Africa is no different. The Information Technology (IT) subject offered in South African secondary schools includes introductory programming knowledge and skills in the subject learning outcomes (Department of Education, 2003). South African IT learners, as novice programmers, are struggling to understand and apply programming concepts (Havenga and Mentz, 2009). IT teachers struggle to address the

[1] Baldwin and Kuljis (2000); Gayo-Avello and Fernández-Cuervo (2003); Lahtinen, Ala-Mutka and Järvinen (2005); Garner (2007); Teague and Roe (2008); Shuhidan, Hamilton and D'Souza (2009)

[2] A person learning to program regardless of programming language and with no previous programming experience.

programming difficulties experienced by their IT learners due to limited contact time with learners and an overloaded subject curriculum.



(a) RobotProg

(b) Scratch

(c) B#

(d) EToys

Figure 1.1: Examples of PATs

Research in the teaching of introductory programming has suggested the use of *programming assistance tools* (PATs) designed specifically for novice programmers (Figure 1.1). PATs are developed with the goals: to enhance comprehension of algorithms and computer programs, assist with code debugging and/or assess programming knowledge and skills (Kelleher and Pausch, 2005). Graphical environments and animation are techniques that can be used to make programming interesting and enhance a user's understanding of programming concepts (Bryant, Weiss, Orr and Yerion, 2011; Pears, Seidman, Malmi, Mannila, Adams, Bennedsen, Devlin and Paterson, 2007). Visualisation techniques have also been incorporated into some PATs to improve the conceptual understanding of programming concepts (Bryant *et al.*, 2011; Baldwin and Kuljis, 2000).

In South African secondary schools, IT learners are not usually provided with PATs. IT teachers seem to be unaware of the range of tools that can support the learners in teaching and learning (Section 4.2). PATs also have educational deficiencies and do not support all of the content in the IT subject programming syllabus (Section 5.4).

1.2 Research Relevance

IT is one of 29 subjects included in the National Curriculum Statement (NCS) of the South African Department of Education (Department of Education, 2003) and is offered at secondary school level for learners in Grades 10, 11 and 12. Although IT is a designated subject contributing to the requirements of learners who want to enrol for a Bachelors degree programme at a Higher Education institution, IT is not a prerequisite subject for any computing related degree programme (NMMU, 2011; SUN, 2011; UCT, 2011).

IT, in South African secondary schools, has earned a reputation as a difficult subject (Havenga and Mentz, 2009) and this impacts learners' decisions to take IT as a subject. Many learners that do attempt the subject change to another, "easier" subject before Grade 12 (the final year) or remain in the subject class but lack the motivation and interest in the programming content to achieve their potential. The lack of interest and motivation in the subject and the negative impression created amongst learners seems to affect negatively the number of learners who decide to pursue further education in a computing discipline.

Table 1.1: Learning Outcome Time Allocation (Department of Education, 2008)

Learning Outcome (LO)	Weighting
LO1 : Hardware and System Software	20%
LO2 : e-Communication	10%
LO3 : Social and Ethical Issues	10%
LO4 : Programming and Software Development	60%
	100%

IT at school has four learning outcomes (Table 1.1) as specified in the IT National Curriculum Statement (Department of Education, 2003). LO4 (Programming and Software Development) has the highest and most substantial weighting of the IT subject and is the most critical learning area. Consequently, learners who are unable to understand and apply programming concepts are unlikely to perform well in the subject.

IT learners in South African secondary schools are faced with the same difficulties as novice programmers around the world (Section 1.1). The difficulties faced by IT learners are aggravated by a learning environment particular to secondary schools. Short (approximately 45 minutes), daily lessons make it difficult both to teach and practise complex programming concepts and skills during school hours (Chapter 4). There is consequently inadequate time available for IT teachers to address the specific programming difficulties of individual IT learners as well as accurately assess learner understanding of concepts during class time. In Grade 10, IT class sizes can range from 20 to 35 learners in a class. The numbers of learners continuing IT in the Grade 11 and Grade 12 classes are less due to attrition.

IT learners lack support in the form of resources to supplement the prescribed IT subject textbook. One such supporting resource is the use of PATs that assist with the development of the understanding of programming concepts and skills during self-study. PATs support self-study (Al-Imamy *et al.*, 2006) and promote active interaction (Baldwin and Kuljis, 2000). The problem that exists is that PATs are not used as IT teachers and IT learners are generally not aware of them (Pears *et al.*, 2007). Furthermore, many of the PATs available are unsuitable for IT learners to use in a classroom learning environment in South African secondary schools. One of the reasons for this situation is that very few of the PATs support the prescribed Delphi (Pascal) programming language. Another problem is that many of the PATs assist in the development of only a subset of the programming skills required by programmers (Rongas *et al.*, 2004). This support is insufficient to improve IT learners' overall understanding of programming concepts.

PATs may employ different techniques to represent code and assist users to construct programs (Kelleher and Pausch, 2005). Code may be represented using text, pictures, flowcharts or animation. PATs may visualise structures or execution of code, animate algorithms, construct algorithms and visualisations graphically, provide editing and syntax support or generate code from templates (Pears *et al.*, 2007). The techniques employed by a PAT are as a result of the programming skills or knowledge that the PAT aims to develop in a novice programmer.

1.3 Research Outline

The lack of support provided to IT learners learning introductory programming in South African schools has been highlighted. The specific problem to be addressed in this investigation (Sections 1.3.1 and 1.3.2) gives rise to a number of research objectives (Section 1.3.3) and research questions (Section 1.3.4).

1.3.1 Problem Statement

The problem statement for this research is the following:

> *IT learners lack programming learning resource support to develop their understanding of programming concepts independently, resulting in a lack of interest and motivation in IT as a subject and in computing as a career.*

South African secondary school IT learners, as novice programmers, struggle to understand and correctly apply programming concepts. The ratio of teacher to students in an average IT class is high considering the practical nature of the programming learning outcome with which learners need assistance. Furthermore, IT learners lack support and assistance to develop programming skills and knowledge during self-study. In particular, there is a lack of support for the Delphi programming language, which is used in most schools offering IT as a subject.

Novice programming research has resulted in the development of a variety of PATs to support the development of programming skills. IT teachers and learners are not aware of the different PATs and how to use PATs to most effectively support the teaching and learning of programming concepts. Furthermore, many of the PATs have deficiencies, particularly with regard to the IT curriculum, which make them unsuitable for use by IT learners in South African secondary schools.

1.3.2 Thesis Statement

The primary aim of this research is to evaluate the use of PATs by IT learners in South African secondary schools. In particular, the objective is to evaluate the support provided by PATs in developing IT learner understanding of programming concepts and the effect of PATs on IT learner motivation towards programming.

The thesis statement that guides this research in achieving its goals is therefore:

> *Programming assistance tools (PATs) can support IT learner development of programming skills and influence motivation to learn programming.*

A number of research objectives are derived from the thesis statement.

1.3.3 Research Objectives

The primary and secondary objectives of this research are related to providing IT learners with existing PATs that are suitable or can be made suitable to support the achievement of the IT subject programming learning outcomes in South African secondary schools.

The primary objectives (POs) of this research are:

PO1: To identify existing introductory programming assistance tools (PATs) that can be used to support learner understanding of programming concepts included in the IT subject curriculum implemented in a case study of South African secondary schools (Chapter 5).

PO2: To evaluate the impact of the selected programming assistance tools (PATs) on a novice programmer's understanding of programming concepts included in the IT subject curriculum implemented in a case study of South African secondary schools (Chapter 7 and 8).

The following secondary objectives (SOs) will be researched in order to achieve the primary objectives (Table 1.2):

SO1.1: Formulate selection criteria for determining the suitability of PATs to support the achievement of programming learning outcomes in the IT subject curriculum (Chapter 4).

SO1.2: Adapt selected PATs to make them suitable to support the achievement of programming learning outcomes in the IT subject curriculum implemented in South African secondary schools (Chapter 6).

SO2.1: Evaluate the impact of the proposed PATs on IT learner understanding of programming concepts (Chapter 7).

SO2.2: Evaluate the influence of the proposed PATs on IT learner motivation towards programming (Chapter 8).

The following research questions (RQ) will be answered to guide the research in order to achieve the research objectives (Table 1.2):

RQ1: What programming difficulties and skills do PATs need to address and develop, respectively?

Table 1.2: Research objectives, questions and methods

Primary Objective	**Secondary Objective**	**Research Question**	**Research Method**	**Chapter**
PO1	SO1.1	RQ1	Literature study: novice programming	3
		RQ2	Interviews with IT teachers	4
			Questionnaires to IT learners	
	SO1.2	RQ3	Literature study to identify existing PATs	5
		RQ4	Adaptation to PATs	6
PO2	SO2.1	RQ5	Questionnaires to IT learners (Perceived difficulty of IT Questionnaire)	7
			End-of-year summative assessment marks	
			Multiple choice class tests	
		RQ7	Questionnaires to IT learners (Evaluation of PATs)	
	SO2.2	RQ6	Questionnaires to IT learners (Motivated Strategies for Learning	8
		RQ8	Questionnaires to IT learners (Evaluation of PATs)	

RQ2: What factors may influence the use of PATs in SA secondary school learning environments?

RQ3: What PATs exist that would be suitable for use in SA secondary schools?

RQ4: How can the selected PATs be adapted for use by IT learners in SA secondary schools to support the understanding of programming concepts?

RQ5: What impact do the different PATs have on IT learner understanding of programming concepts?

RQ6: What impact do the PATs have on IT learner motivation towards programming?

RQ7: What techniques should PATs implement in order to assist IT learner understanding of programming concepts?

RQ8: What techniques should PATs implement in order to motivate IT learners to learn programming concepts and to use a PAT?

The data collection methods that are used to answer each of the research questions include surveys, class tests and a review of relevant literature (Table 1.2). A study of related work in the field of introductory programming identifies difficulties experienced by novice programmers in general, as well as PATs suitable for use by IT learners. IT teacher and learner questionnaire responses provide feedback regarding the novice programming difficulties experienced by IT learners in South African secondary schools. Questionnaires are also used to evaluate IT learner motivation towards programming and the IT subject (Section 2.5.1). Multiple choice class tests (Section 2.5.2) are used to evaluate IT learner understanding of specific programming concepts. The research methods, questionnaires administered to IT learners and teachers and multiple choice tests administered to IT learners are presented in more detail in the research design and methods chapter (Chapter 2).

1.4 Research Methodology

In any research study, the research plan must suit the purposes of the research (Cohen, Manion and Morrison, 2007). The adapted research process "onion" (Figure 1.2) outlines the main issues that need to be considered before starting the research study (Saunders, Lewis and Thornhill, 2006).

The research philosophy or paradigm is a way of looking at the world and is composed of philosophical assumptions that guide and direct thinking and action (Mertens, 2004). The post-positivism paradigm will be adopted by this research to evaluate quantitatively the impact of the proposed PATs.

The post-positivism paradigm is a philosophy associated with educational research (Phillips and Burbules, 2000) and has been used to determine the impact of teaching methods (Levy *et al.*, 2001; Cohen *et al.*, 2007). The post-positivism paradigm is mainly quantitative, however, qualitative methods can also be used (Mertens, 2004; Cohen *et al.*, 2007). The research will follow a deductive approach using an experimental research strategy.

Quasi-experimental methods are commonly used in the evaluation of educational programs in cases when it is not possible or practical to assign participants randomly (Gribbons and Herman, 1997). The quasi-experimental method is used to determine if there is a difference in IT learner motivation (SO2.2) and the perceived difficulty of programming concepts (SO2.1) between learners that use a PAT and learners that do not. The research uses a case study involving IT learners from a number of schools using Delphi as the programming language. Qualitative methods are used to gather information on IT learner opinions related to the proposed PATs (RQ7 and RQ8).

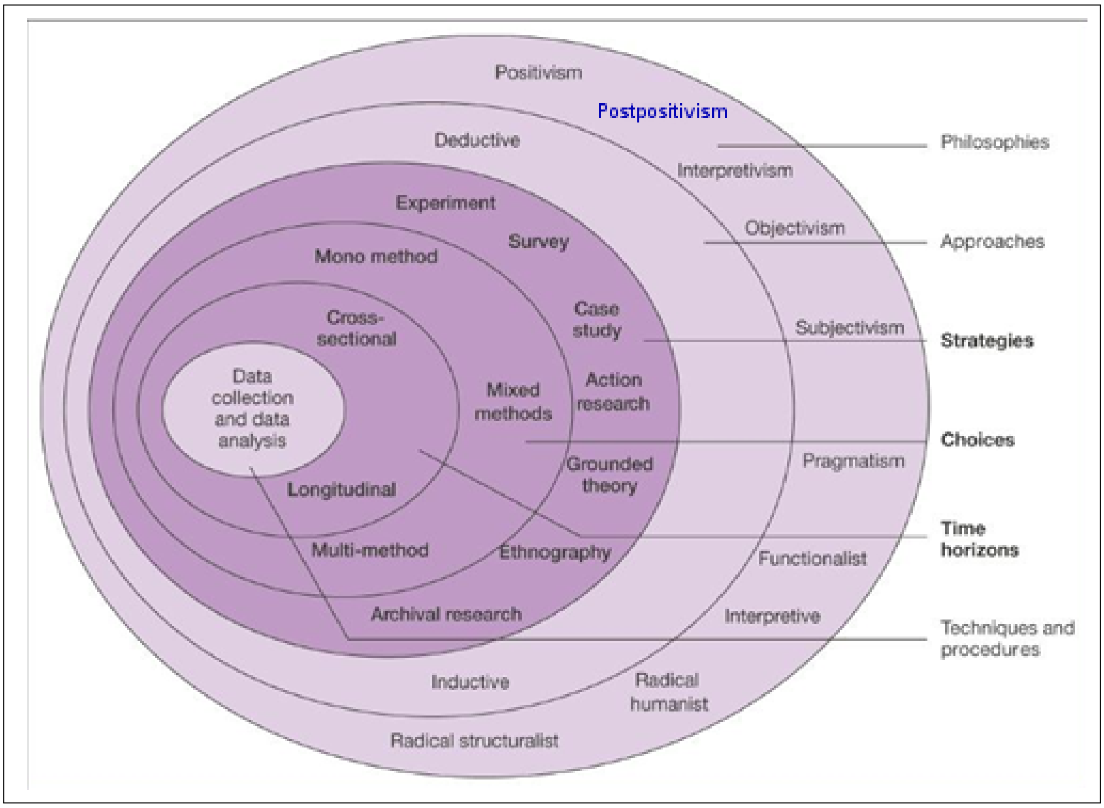


Figure 1.2: Adapted Research Process Onion

Data collection methods include a literature study together with questionnaire feedback from South African IT learners and teachers (Table 1.2). The data collected from these two methods are used to identify existing PATs and criteria to select PATs (SO1.1) in order to support the learning of IT programming concepts. The proposed PATs are adapted, if necessary and if possible, for use by learners (SO1.2).

The research methodology employed in this study is primarily quantitative in nature, although qualitative analysis methods are also used. Qualitative research methods are used at the start of the study to identify factors influencing the performance of IT learners in South African secondary schools. Quantitative data analysis techniques are used to determine the impact of the proposed PATs on learner perceptions of the difficulty of different programming concepts and the impact on learner motivation with regard to programming. The research design is discussed in detail in Chapter 2.

1.5 Scope and Limitations

The non-equivalent groups design (NEGD) is common in education research for the selection of groups for the quasi-experimental research strategy. In the NEGD, two comparable groups are selected for the research (Trochim, 2006). Groups are selected in such a way that they are as similar as possible to compare them fairly. A limitation of using NEGD is that the groups are not equivalent due to lack of random assignment. Any prior differences between the groups may thus affect the outcome of the research.

The context of the research includes schools in the Eastern Cape Province, specifically Port Elizabeth. Schools in this province teach Delphi as the programming language. Some aspects of the research, especially with regard to the PATs, may thus not be generalisable for the teaching of all IT learners in South Africa as some provinces use Java as the programming language. Learners, using the proposed PATs, should be able to achieve the learning outcomes of the IT subject regardless of whether Delphi or Java is used as the programming language, but this will not be verified as part of this investigation.

The number of schools in the sample population is less than that required to represent accurately the theoretical population, which is all schools in SA teaching Delphi. The study will require IT learners and teachers, as participants, to consent to participation in the study (Appendix H). The sample size is consequently affected by the willingness of schools, IT teachers and learners to participate in the research for the entire duration of the investigation. To increase the chance of a random sample, the sample size is thus the size of each participating class, which is on average less than 30 (Cohen *et al.*, 2007).

The research study is designed to ensure, as far as possible, that the use of the PATs is the only difference that the control and treatment groups may experience, besides the fact that the classes themselves are different. However, changes in teacher or learning environments at the school are beyond the control of the researcher. Any changes are noted and catered for as best possible in the analysis of the research data.

Adaptations to the PATs are limited to changes that will make the programming language of the PAT similar to the Delphi programming language. The quasi-experimental research strategy is then used to evaluate the impact of the PATs in participating schools.

The research will not evaluate the use of the proposed PATs for all programming concepts taught to IT learners for the duration of the IT subject to minimise the disruption of the IT teacher's planned work schedule. One of the selected PATs is provided to the participating treatment group Grade 10 and Grade 11 IT learners in each of the participating schools to use in their own time. IT learner knowledge of certain programming concepts are assessed using class tests, after different concepts have been taught, in the course of the year.

1.6 Conclusion and Thesis Structure

IT learners in South African secondary schools face many difficulties. IT learners struggle to understand and successfully apply introductory programming concepts. Practical programming time in class where learners are able to apply their programming knowledge in the presence of the teacher, is limited. Consequently, IT learners require support and assistance to develop their programming knowledge and skills independently. PATs could support IT learners, however, IT teachers and learners are not aware of the range of PATs in existence.

The aim of this study is to identify and evaluate PATs that would be suitable for use by IT learners. The study evaluates the support PATs provide to IT learners with respect to the understanding of programming concepts and the influence of PATs on IT learner motivation towards programming. The results of this study indicate whether PATs can improve IT learner understanding of programming concepts and motivation towards programming. The findings of the study also identify different techniques used by the PATs that motivate learners and assist with the understanding of programming concepts. Based on the feedback from learners, techniques that a PAT should employ are suggested.

The research is using a case study of four IT schools in the Port Elizabeth area which use Delphi as the programming language. One reason for selecting these schools is to make access to participants more convenient. The influence of school environments and teachers has also resulted in each of the participating schools receiving a different PAT. The impact of the PAT on IT learner understanding of programming concepts and motivation towards programming is evaluated for each PAT separately per school. The evaluation of the PATs may thus not be generalisable to the larger IT learner populations. Certain parts of the questionnaire feedback, related to IT learner opinions of IT programming, are combined, increasing the sample size to make the results generalisable.

The research strategy used to answer the research questions and achieve the research objectives of this research study, taking into consideration the scopes and limitations outlined, is discussed in more detail in Chapter 2 (Research Design and Methods). This discussion includes identifying the different research methods, data collection and analysis methods that will be used.

Thereafter, the chapters of the thesis (Figure 1.3) are structured to address progressively the research questions and objectives. Chapter 3 (Introductory Programming) provides a study of related work on the topic of novice programming research. It is important to identify reasons why programming is difficult and identify recommendations for assisting

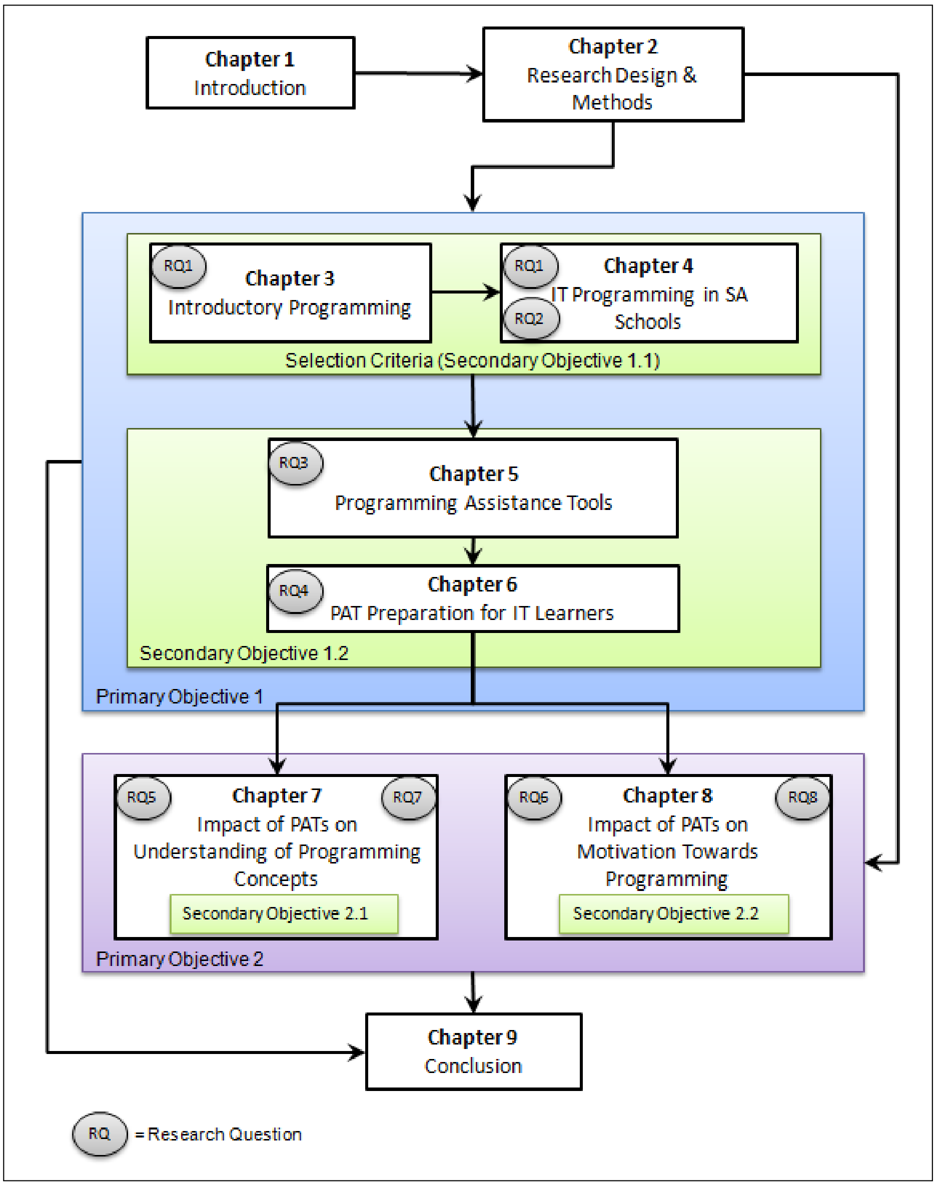


Figure 1.3: Chapter outline

novice programmers in introductory programming courses (RQ1). Chapter 3 identifies specific novice programming difficulties from literature that are included in the selection criteria for PATs, formulated in Chapter 4.

Chapter 4 (IT Programming in South African Schools) first focuses on IT programming in South African schools by gathering feedback from IT teachers participating in this research study (RQ2). IT learner feedback, identifying difficult programming concepts and learning preferences, is also presented (RQ1). The feedback from IT teachers and learners, together with the programming difficulties identified in Chapter 3, are used to formulate and present selection criteria for proposed PATs (SO1.1).

Chapter 5 (Programming Assistance Tools) discusses specific PATs identified from novice programming research literature (RQ3). The selection criteria (Chapter 4) are used to evaluate the PATs. Chapter 5 concludes by identifying PATs that are suitable for evaluation by IT learners in South African secondary schools.

Chapter 6 (PAT Preparation for IT Learners) describes any adaptations made to the PATs identified in Chapter 5 (RQ4). These adaptations are implemented before the PATs are provided to IT learners for use and evaluation (SO1.2). Chapter 6 also discusses issues where the implementation of programming concepts in the PATs differs from the implementation in Delphi and no adaptation is possible.

At this point in the thesis, PATs suitable for South African secondary schools have been selected, thus achieving Primary Objective 1. The selected PATs are then evaluated in the participating schools in order to address Primary Objective 2.

Chapter 7 (Impact of PATs on the Understanding of Programming Concepts) evaluates the impact of the PATs on IT learner understanding of programming concepts (SO2.1). The impact is evaluated by comparing the difficulty of IT (Grade 11 learners) perceived by the control and treatment group, multiple-choice class test assessment and end-of-year summative assessment results (RQ5). IT learner questionnaire feedback is used to identify techniques that support the learning of programming concepts (RQ7).

Chapter 8 (Impact of PATs on Motivation towards Programming) evaluates the impact of the PATs on IT learner motivation towards programming as well as motivation to use the different PATs selected (SO2.2). The impact on motivation towards programming is evaluated by comparing control and treatment groups (RQ6). IT learner feedback with respect to the usefulness and shortcomings of the PATs used, identify techniques that motivate IT learners to use the PATs (RQ8).

Chapter 9 (Conclusion) presents a summary of the research results and conclusions drawn from the findings of the research. The research objectives and questions are revisited and major contributions of the research and future research opportunities are identified.

Chapter 2

Research Design and Methods

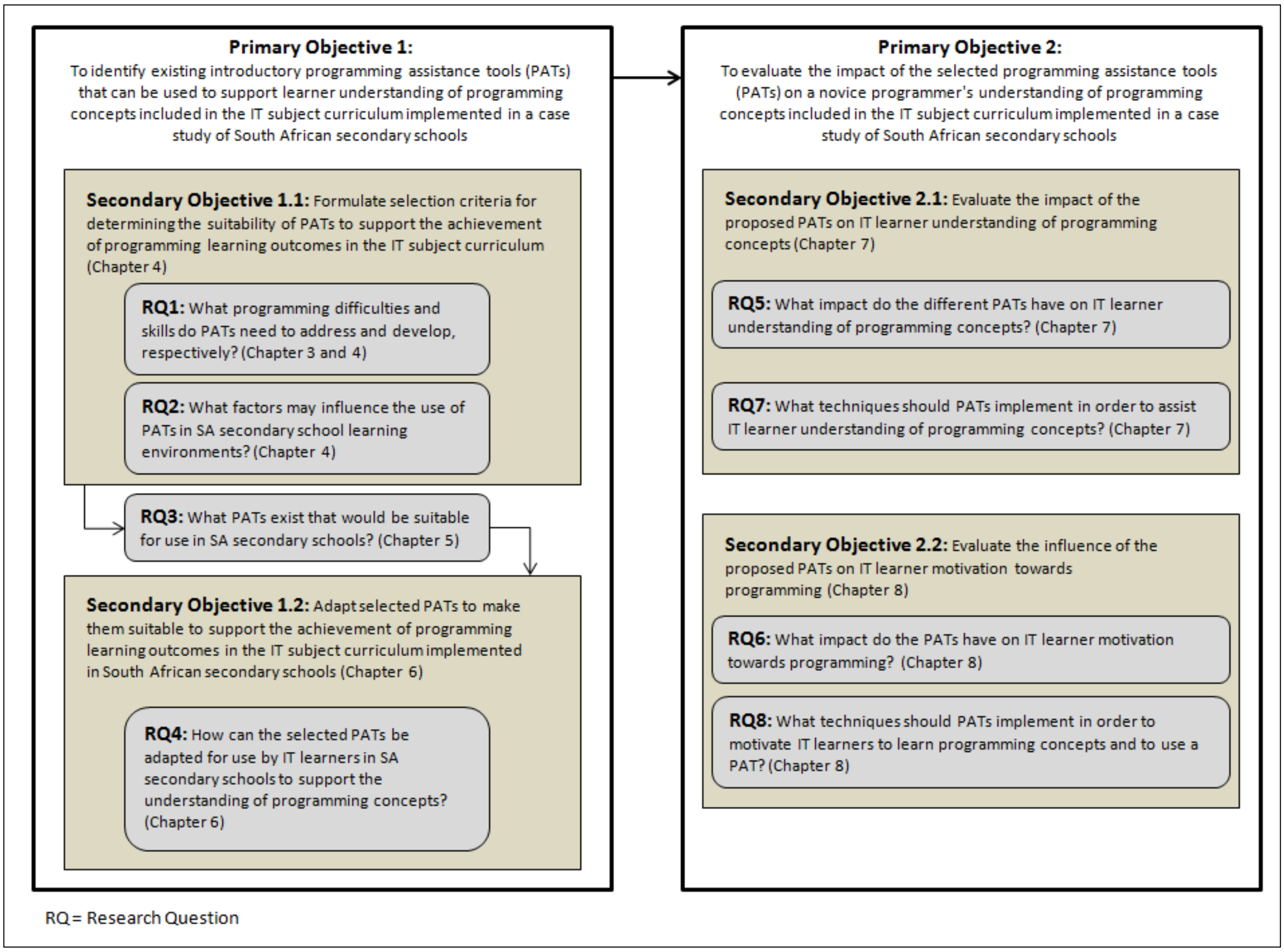
2.1 Introduction

IT learners in South African secondary schools, as novice programmers, struggle to develop their programming skills. Several factors related to the secondary school teaching environment contribute to this difficulty. Short daily lessons and the IT teacher as the only tutor are just two of the factors identified (Section 1.2).

This research investigates whether the use of PATs impacts an IT learner's understanding of introductory programming concepts. In addition, the impact of the use of PATs by IT learners on their motivation towards programming and the IT subject (Section 1.4), is evaluated. Appropriate research methods are identified to answer the research questions and achieve the stated research objectives (Section 2.2). Research hypotheses are presented (Section 2.3), followed by a description of the participant selection methods (Section 2.4), methods used to collect data from IT learners and teachers (Section 2.5) and data analysis methods (Section 2.6). The process contributes to a comprehensive evaluation strategy to assess the impact of the use of PATs in a secondary school environment.

2.2 Research Strategy

The research strategy used to achieve the research objectives is applied in two phases, namely the identification of suitable PATs for IT learners (Primary Objective 1) and the evaluation of the impact of the PATs identified on IT learner understanding of programming concepts and their motivation towards programming (Primary Objective 2). In this work, the term, *suitable PATs*, refers to PATs that support IT learner development of their understanding of programming concepts that are included in the IT curriculum.



Primary Objective 1: To identify existing introductory programming assistance tools (PATs) that can be used to support learner understanding of programming concepts included in the IT subject curriculum implemented in a case study of South African secondary schools

Secondary Objective 1.1: Formulate selection criteria for determining the suitability of PATs to support the achievement of programming learning outcomes in the IT subject curriculum (Chapter 4)

RQ1: What programming difficulties and skills do PATs need to address and develop, respectively? (Chapter 3 and 4)

RQ2: What factors may influence the use of PATs in SA secondary school learning environments? (Chapter 4)

RQ3: What PATs exist that would be suitable for use in SA secondary schools? (Chapter 5)

Secondary Objective 1.2: Adapt selected PATs to make them suitable to support the achievement of programming learning outcomes in the IT subject curriculum implemented in South African secondary schools (Chapter 6)

RQ4: How can the selected PATs be adapted for use by IT learners in SA secondary schools to support the understanding of programming concepts? (Chapter 6)

Primary Objective 2: To evaluate the impact of the selected programming assistance tools (PATs) on a novice programmer's understanding of programming concepts included in the IT subject curriculum implemented in a case study of South African secondary schools

Secondary Objective 2.1: Evaluate the impact of the proposed PATs on IT learner understanding of programming concepts (Chapter 7)

RQ5: What impact do the different PATs have on IT learner understanding of programming concepts? (Chapter 7)

RQ7: What techniques should PATs implement in order to assist IT learner understanding of programming concepts? (Chapter 7)

Secondary Objective 2.2: Evaluate the influence of the proposed PATs on IT learner motivation towards programming (Chapter 8)

RQ6: What impact do the PATs have on IT learner motivation towards programming? (Chapter 8)

RQ8: What techniques should PATs implement in order to motivate IT learners to learn programming concepts and to use a PAT? (Chapter 8)

RQ = Research Question

Figure 2.1: Relationship between the research objectives and questions

The identification of suitable PATs in the first phase of the study to achieve Primary Objective 1 (Figure 2.1) requires an understanding of the difficulties faced by novice programmers in general and IT learners in particular. An understanding of the difficulties is used to evaluate existing PATs and determine their suitability for IT learners (Section 2.2.1). In the second phase of the study, in order to achieve Primary Objective 2 (Figure 2.1), the impact of the selected PATs on IT learner understanding of programming concepts and motivation towards programming, is assessed (Section 2.2.2). During this phase, the selected PATs are provided to IT learners to use.

Data for both phases of the study is collected using several questionnaire instruments (Section 2.5). Only Grade 10 and Grade 11 IT learners participate in the research study. Grade 12 learners were not included to avoid any disruption in their final year that could negatively affect their final assessment. The research strategy for each phase and the research questions and objectives that each of these two phases aim to achieve are discussed further in this section.

2.2.1 Identification of Suitable PATs

PATs should be able to assist novice programmers with most, if not all, programming concepts considered difficult to understand (Rongas *et al.*, 2004). The first phase of the research study (Figure 2.2) identifies PATs that can support IT learner programming skills and understanding of programming concepts (Primary Objective 1). Primary Objective 1 is achieved by completing several steps which aim to identify the difficulties related to introductory programming and programming in South African schools and to identify PATs suitable for use in South African secondary schools.

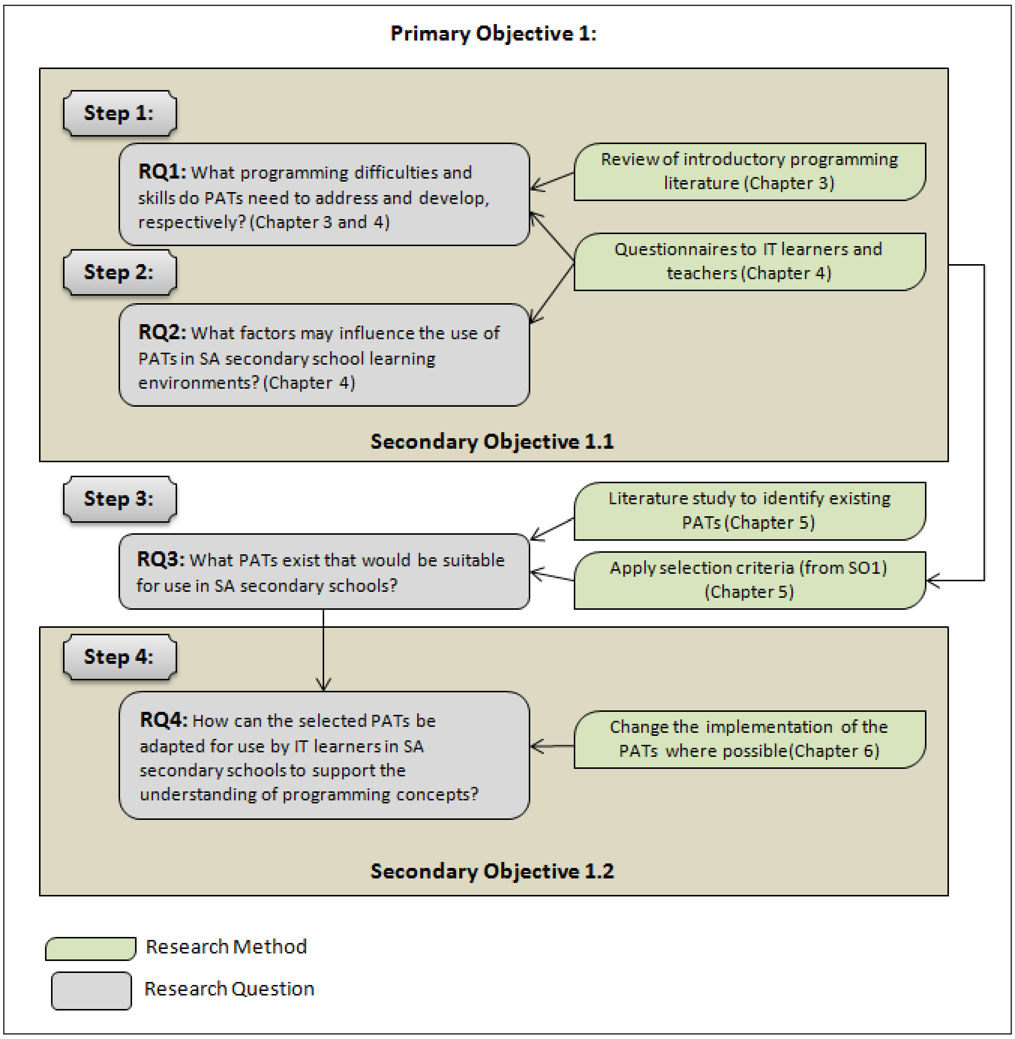

Primary Objective 1:
Step 1:
RQ1: What programming difficulties and skills do PATs need to address and develop, respectively? (Chapter 3 and 4)
Review of introductory programming literature (Chapter 3)
Questionnaires to IT learners and teachers (Chapter 4)
Step 2:
RQ2: What factors may influence the use of PATs in SA secondary school learning environments? (Chapter 4)
Secondary Objective 1.1
Step 3:
Literature study to identify existing PATs (Chapter 5)
RQ3: What PATs exist that would be suitable for use in SA secondary schools?
Apply selection criteria (from SO1) (Chapter 5)
Step 4:
RQ4: How can the selected PATs be adapted for use by IT learners in SA secondary schools to support the understanding of programming concepts?
Change the implementation of the PATs where possible(Chapter 6)
Secondary Objective 1.2
Research Method
Research Question

Figure 2.2: Research strategy to achieve Primary Objective 1

The first step answers RQ1 (Figure 2.2) by identifying programming concepts with which novice programmers and, specifically, IT learners have difficulty. A review of introductory programming literature (Chapter 3) addresses the former and questionnaires to IT learners and teachers (Chapter 4) address the latter.

In the second step, questionnaire responses from IT learners and teachers inform the response to RQ2 (Figure 2.2). The programming concepts and learning environment factors identified from RQ1 and RQ2 are used to formulate selection criteria.

A survey of related research and literature, as well as a general search of PATs that are freely available for download from the Internet, are used to identify suitable PATs (Chapter 5) in the third step (Figure 2.2). From the PATs identified, only those that would be suitable for use by IT learners in South African secondary schools (Section 2.4) are selected. Selection criteria are used to determine the suitability of PATs to support the achievement of programming learning outcomes in the IT subject curriculum (Secondary Objective 1.1).

In the fourth and final step, the PATs are adapted, if necessary and if possible, for use by IT learners in South African secondary schools (Secondary Objective 1.2, Chapter 6). Adaptations to the selected PATs to make them more suitable for use by IT learners are implemented in order to answer RQ4. After the adaptations are implemented, the PATs are ready for use by IT learners and Primary Objective 1 is achieved. The selected PATs are evaluated in the second phase of the research study.

2.2.2 Evaluation of Proposed PATs

The second phase of the research study evaluates the impact of the PATs (Primary Objective 2) that have been identified and adapted in the first phase of the study (Figure 2.3). A between-groups quasi-experimental approach (Figure 2.4) is used to evaluate the impact of the selected PATs on IT learner understanding of programming concepts (Secondary Objective 2.1) and on IT learner motivation towards programming (Secondary Objective 2.2). Several questionnaire instruments as well as class assessments are used to measure IT learner understanding of programming concepts and motivation towards programming (Figure 2.3).

An experimental type research design is suitable to evaluate the application of the combination of the proposed PATs in natural settings - learning IT programming in a school. Experimental research design propositions are of the form:

> *If the model is applied, a specific outcome will result and if the model is not applied, the specific outcome does not result.*

Experimental research propositions are ideal to test the proposed PATs where the desired outcome would be improved understanding of programming concepts. However, the key

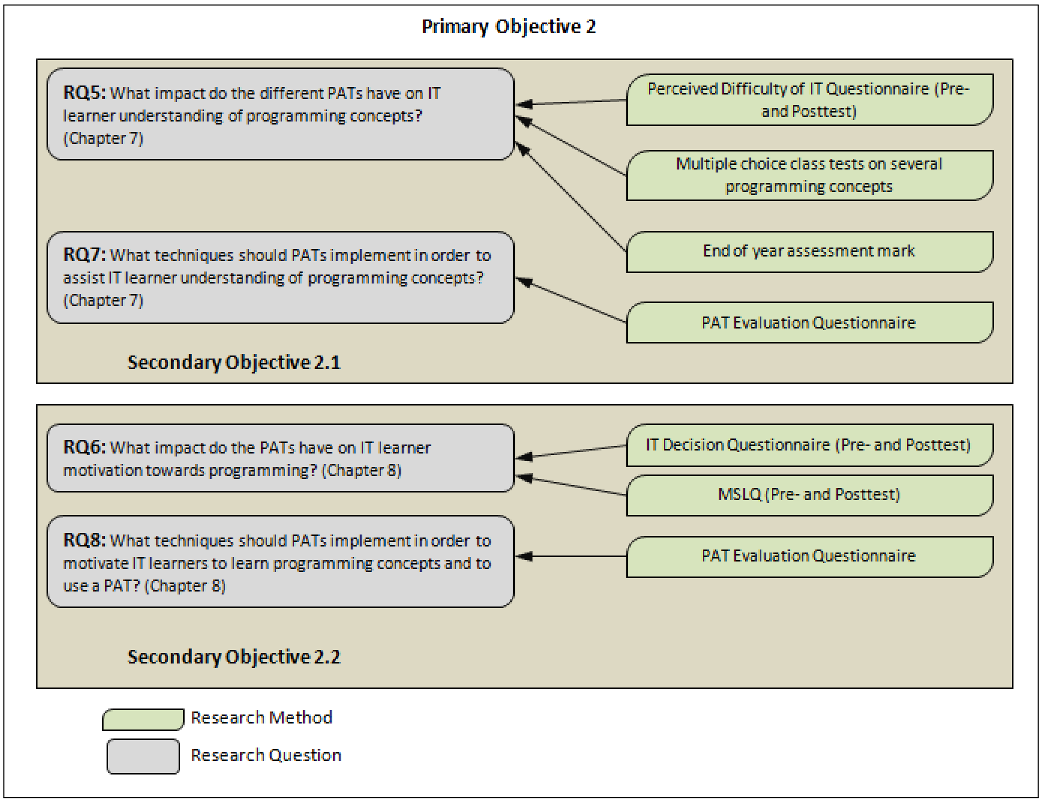

Primary Objective 2
RQ5: What impact do the different PATs have on IT learner understanding of programming concepts? (Chapter 7)
RQ7: What techniques should PATs implement in order to assist IT learner understanding of programming concepts? (Chapter 7)
Perceived Difficulty of IT Questionnaire (Pre- and Posttest)
Multiple choice class tests on several programming concepts
End of year assessment mark
PAT Evaluation Questionnaire
Secondary Objective 2.1
RQ6: What impact do the PATs have on IT learner motivation towards programming? (Chapter 8)
RQ8: What techniques should PATs implement in order to motivate IT learners to learn programming concepts and to use a PAT? (Chapter 8)
IT Decision Questionnaire (Pre- and Posttest)
MSLQ (Pre- and Posttest)
PAT Evaluation Questionnaire
Secondary Objective 2.2
Research Method
Research Question

Figure 2.3: Research Strategy to achieve Primary Objective 2

to success of the experimental approach is that the proposed PATs, in this case, are the only reason why the specific outcome results or does not result. This is achieved in the experimental research design by creating two equivalent groups - a control group not using the proposed PATs and a treatment group where the proposed PATs are used.

In a true experimental design, the participants of each group should be equivalent. Equivalence of the groups is achieved by randomly assigning participants to the two groups (Cohen *et al.*, 2007; Gribbons and Herman, 1997; Kenny, 1975; Trochim, 2006). However, participating learners across the different schools could not be randomly assigned to control and treatment groups as there could be no guarantee that control group learners, in the same class as treatment group learners, did not also use the treatment. Similarly, there could be no way of knowing if treatment group learners in the same class but using different PATs did not use the PAT assigned to other learners. Learners in one school could also not be compared to learners in another school as different teachers teaching each class in the control and treatment groups might result in bias in the data collected to evaluate the PATs.

The quasi-experimental research design is thus appropriate to this investigation (Gribbons and Herman, 1997; Kenny, 1975) as existing IT classes within the schools are used. Each school's participating learners receive the same PAT and the control and treatment group classes, allocated over two consecutive academic years, are compared within each school. The non-equivalent groups design (NEGD) of the quasi-experimental research approach (Table 2.1), is used. The NEGD is structured similarly to pretest-posttest randomised experiments but lacks random participant assignment to the control and treatment groups (Kenny, 1975). O1 represents the pretests administered to the treatment and control groups before treatment, X (Table 2.1). O2 represents the posttests administered after the treatment period.

Table 2.1: Non-equivalent groups design of quasi-experimental approach

Treatment	O1	X	O2
Control	O1		O2

The proposed PATs are used by several IT classes in their respective schools. The treatment group learners are introduced to and provided with the proposed PATs to use during the school academic year. The effect of the teacher's style of teaching on the results is minimised by making the control group the teachers' IT class of the academic year preceding the treatment group IT learners.

The research spans two academic years to collect the data required from the control and treatment groups. A full academic year is used for each group as two separate classes per grade from each school provides a larger number of learners to participate in the study rather than splitting one academic class into control and treatment groups. Separating the control and treatment groups by academic year also prevents control group participants from inadvertently having access to the treatment.

Table 2.2: Experimental research design for this study

Treatment				O1	X	O2
Control	O1		O2			
	year α			year α+1		

Table 2.2 illustrates how the design is adjusted for this research study. The control and treatment groups are IT classes of two different academic years. The control group pre- and posttests are thus administered before the treatment group pretests are administered at the beginning of the next academic year.

An experimental research strategy (Figure 2.4) is used to predict the expected relationship between the control and treatment groups with respect to IT learner understanding of programming concepts (RQ5). The *Perceived Difficulty of Programming* questionnaire

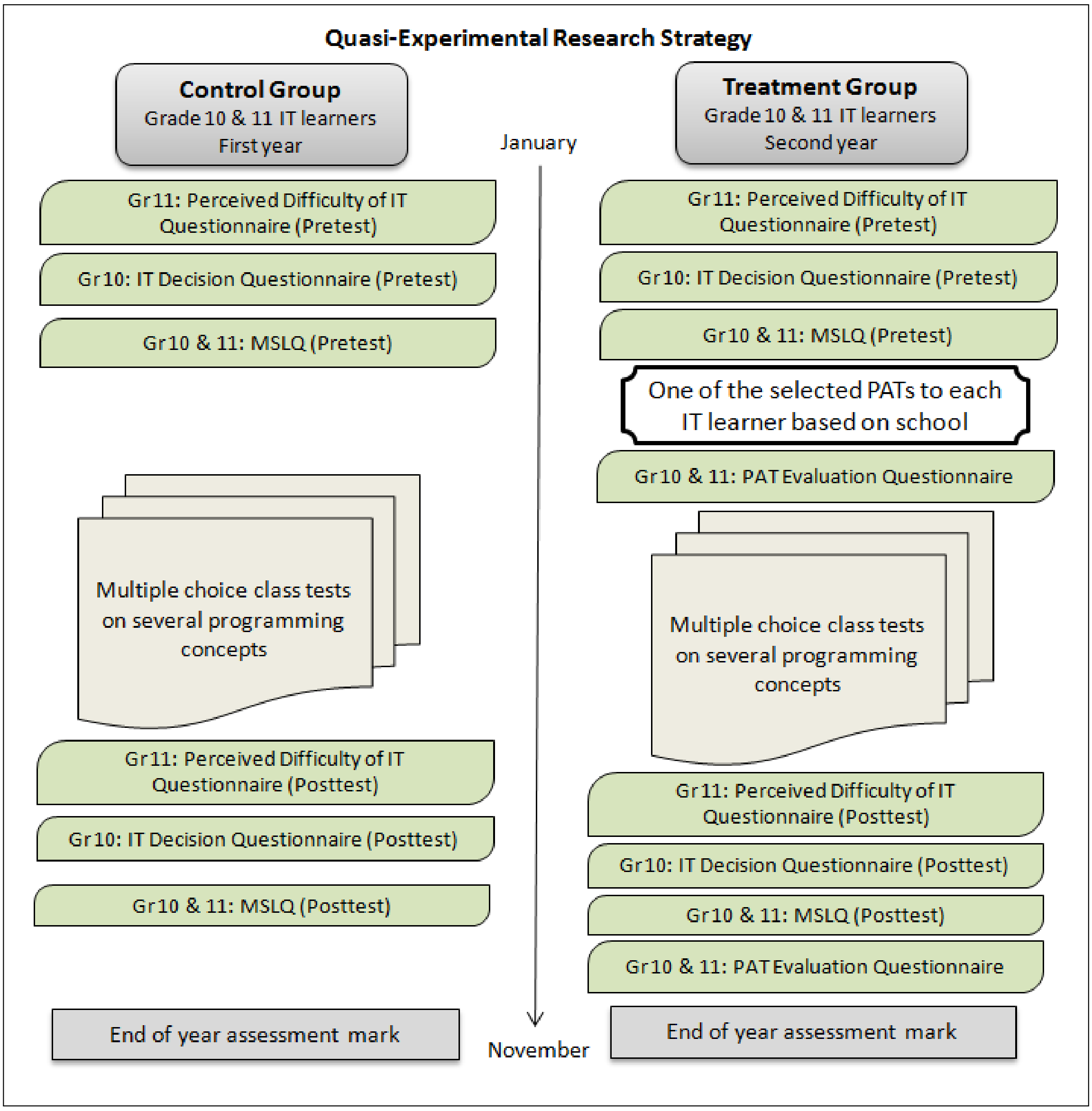

Quasi-Experimental Research Strategy
Control Group
Grade 10 & 11 IT learners
First year
Treatment Group
Grade 10 & 11 IT learners
Second year
January
Gr11: Perceived Difficulty of IT Questionnaire (Pretest)
Gr10: IT Decision Questionnaire (Pretest)
Gr10 & 11: MSLQ (Pretest)
One of the selected PATs to each IT learner based on school
Gr10 & 11: PAT Evaluation Questionnaire
Multiple choice class tests on several programming concepts
Gr11: Perceived Difficulty of IT Questionnaire (Posttest)
Gr10: IT Decision Questionnaire (Posttest)
Gr10 & 11: MSLQ (Posttest)
Gr10 & 11: PAT Evaluation Questionnaire
End of year assessment mark
November

Figure 2.4: Pre- and posttests administered during the quasi-experimental approach

(Section 2.5.1, Appendix C), together with multiple choice tests to assess concept knowledge are used for the evaluation (Section 2.5.2). An experimental research strategy is also used to determine the impact of the proposed PATs on IT learner motivation towards programming (RQ6). The *Motivated Strategies for Learning Questionnaire* is used for this evaluation (Section 2.5.1). The findings and analysis of questionnaire results (Section 2.5.1) are also used to identify the techniques used by the PATs to assist IT learner understanding of (RQ7) and motivation for learning programming concepts (RQ8).

Quantitative hypotheses (Section 2.3) are used to compare the relationship between the control and treatment groups with respect to IT learner's understanding of programming concepts and motivation towards programming (Creswell, 2009). The procedure used to select and obtain participants for this research, is then discussed (Section 2.4). This is followed by a section on data collection (Section 2.5), providing detailed descriptions of each of the questionnaire instruments used and their purpose in the research study. The methods that are used for the analysis of the data collected are presented (Section 2.6) and risks to the study (Section 2.7) due to the research strategy and methods used, are identified before this chapter is concluded (Section 2.8).

2.3 Research Hypotheses

The main hypothesis tested for this research related to Primary Objective 2 is the following:

$H_{0,0}$: There is no difference in the performance/experience between control and treatment groups

$H_{0,1}$: There is a difference in the performance/experience between control and treatment groups

The main hypothesis is tested using two sub-hypotheses. The first sub-hypothesis (H_1) aims to answer RQ5 by evaluating the impact the different PATs have on IT learner understanding of programming concepts. The second sub-hypothesis (H_2) aims to answer RQ6 by evaluating the impact the different PATs have on IT learner motivation towards programming. The hypotheses are tested at the 95% significance level.

H_1: Programming concept knowledge:

$H_{1,0}$: There is no difference between the assessment means of the control and treatment groups (μ_1=μ_2, μ_1 is the control group assessment mean, μ_2 is the treatment group assessment mean)

$H_{1,1}$: There is a difference between the assessment means of the control and treatment groups ($\mu_1 \neq \mu_2$)

The mean referred to is an assessment of the concept knowledge shown by the control and treatment groups using the *Perceived Difficulty of Programming Questionnaire* (Section 2.5.1.3) as well as the end-of-year IT summative assessment mark obtained by learners.

H_2: Motivation toward IT Programming:

$H_{2,0}$: There is no difference between the motivational strategy means of the control and treatment groups (ω_1=ω_2, ω_1 is the control group motivational strategy scores mean, ω_2 is the treatment group motivational strategy scores mean)

$H_{2,1}$: There is a difference between the motivational strategy means of the control and treatment groups ($\omega_1 \neq \omega_2$)

The motivational strategy scores are obtained from the *Motivated Strategies for Learning Questionnaire* (MSLQ) discussed in Section 2.6.1.

2.4 Participant Selection

The selection and number of schools and IT teachers that participate in the experiment is affected by the willingness of the teacher, learners and school to participate. IT teachers of schools in the sampling frame were approached and requested to participate in the research. The selections of the treatment and control groups or classes and their willingness to participate thus determine the sample size, n where:

$$n = \min(\text{participants }_{year(\alpha)}, \text{participants }_{year(\alpha+1)})$$

Grade 10 and Grade 11 IT learners in the consenting schools consent to participate in the research study following all required ethical consideration (Ref: H10-Sci-CSS-001). All participants in the study, namely learners, teachers and schools, are informed that participation in the study is voluntary. Participants may withdraw from the study at any time after consenting to participate. Those not participating are not disadvantaged in any way. There is a risk that due to the fact that participation is voluntary and parental/guardian consent is required, the number of participants could be less than the minimum required for a sample normal distribution. Attrition of IT learners in the class may also reduce the number of participants over the course of the study. Commonly used inferential statistical methods for experimental data analysis may thus not be applicable; alternative methods will have to be used (Section 2.7).

2.5 Data Collection

Several data collection methods are employed to gather qualitative and quantitative data for this research study. The main method of data collection for this research study is the use of questionnaire instruments (Section 2.5.1). Several different questionnaires are completed by IT learner participants in the study. Additional data relating to IT learner knowledge of programming concepts is obtained from multiple-choice class tests written by IT learners (Section 2.5.2). Final IT end-of-year summative assessment marks are obtained from the IT teachers for participating learners in Grade 10 and 11.

2.5.1 Questionnaires

A questionnaire is a useful instrument for collecting survey information (Cohen *et al.*, 2007). It can provide structured, often numerical data and can be administered without the researcher having to be present. A questionnaire is therefore the ideal survey instrument for collecting information from IT learners. The large number of participants would make interviews impractical.

All questionnaires are self-administered without the researcher being present. Questionnaires administered to IT teachers are aimed at gathering information on the teacher's views of IT in South African schools (Section 2.5.1.1). The questionnaires provided to IT learners are used to determine reasons for selecting IT as a subject (Section 2.5.1.2), learner perception's of different programming concepts and skills (Section 2.5.2.3), learner motivation in the subject (Section 2.5.2.4), learning preferences of IT learners (Section 2.5.2.5) and IT learner evaluation of the PATs received (Section 2.5.2.6).

2.5.1.1 IT Teacher Questionnaires

The aim of the questionnaires to IT teachers is to gain insight into the difficulties faced by IT teachers in South African schools with regards to the factors that influence the teaching of introductory programming concepts in schools. The questionnaires to IT teachers gather data to address RQ1: *What programming difficulties and skills do PATs need to address and develop, respectively?* and RQ2: *What factors may influence the use of PATs in SA secondary school learning environments?*

Two questionnaires are administered to IT teachers at the beginning of the research study. One of the questionnaires is an open-ended questionnaire which aims to gather information regarding the current IT teaching environments and the IT teacher's opinions regarding the difficulties faced by IT teachers (Appendix A.1). The purpose of this questionnaire is to gather data that can be used to qualitatively address RQ2. The questionnaire consists of 12 questions that investigate the IT teacher's views on the current IT curriculum, the differences between teaching IT and other subjects, the challenges of teaching IT, the suitability of current textbooks and supporting materials and whether or not the amount of time allocated in the school timetable to IT and programming is adequate for presenting programming concepts to learners in sufficient amount of detail.

The second questionnaire (Appendix A.2) consists of items that aim to identify programming concepts that IT teachers find difficult to teach to learners, teaching styles used by teachers to introduce programming concepts, the manner in which IT teachers manage programming exercises, methods of debugging taught to learners and whether or not IT teachers have used PATs to teach programming concepts. The data collected using this questionnaire is used to answer RQ1.

Qualitative analysis results (Section 4.2) of the questionnaire data collected are used to gain further insight into the current teaching environment with regards to the IT subject in South African secondary schools. The questionnaire aims to acquire IT teacher opinions relating to the challenges faced when teaching IT with respect to the curriculum and teaching environments. The questionnaire also aims to gather feedback on which concepts are easy and which are difficult for IT learners to understand. The questionnaire responses are incorporated in the selection criteria formulated (Section 4.4) to identify PATs that would be suitable for IT learners.

2.5.1.2 IT Decision Questionnaire

The aim of this questionnaire (Appendix B) is to identify the reasons why Information Technology (IT) was chosen as a subject by the learners. IT learner responses to the questionnaire identify IT learner perceptions and expectations of the IT subject and programming in particular. The results of this questionnaire administered to Grade 10 learners contribute to an understanding of the IT learning environment and, in particular, the expectations of IT learners. The feedback is used to answer RQ2: *What factors may influence the use of PATs in SA secondary school learning environments?* This questionnaire is adapted from questions used by Biggers, Brauer and Yilmaz (2008) in a similar study to investigate university student perceptions of Computer Science (CS) and the reasons why students decided to major in CS.

The questionnaire is administered to both control and treatment Grade 10 groups as a pretest at the beginning of the academic year. Grade 10 learners, in the first term of their Grade 10 year, would have received no or very little programming instruction up until the time the questionnaire is administered and would thus not be able to provide meaningful feedback on the difficulty of programming. It was therefore decided to determine Grade 10 learners' reasons for selecting IT as a subject.

Learners are also asked to rate the following two questions, "I think IT is a difficult subject when compared to other school subjects" and "I think I can do well in IT", using a 7-point semantic differential scale where 1 is "Strongly disagree" and 7 is "Strongly agree". These two questions are aimed at determining a learner's perceived difficulty of the subject. At the end of the Grade 10 year, a follow-up (posttest) questionnaire is administered to learners in the control and treatment groups (Appendix B). IT learners rate their perceived difficulty of the subject again and, in addition, learners are asked to motivate whether they would repeat their decision to select IT as a subject. The results from these items are used to determine whether or not the use of the PATs influenced learners' perceptions of the difficulty of programming and their decision to take IT as a subject. These results are presented in the evaluation of the impact of PATs on learner motivation towards programming (RQ6).

2.5.1.3 Perceived Difficulty of Programming Questionnaire

The *Perceived Difficulty of Programming Questionnaire* (Appendix C) is aimed at identifying programming concepts and skills with which IT learners have difficulty. This questionnaire serves two purposes in this research study. The first purpose is to answer RQ1: *What programming difficulties and skills do PATs need to address and develop, respectively?*

The second purpose of this questionnaire is to determine whether the proposed PATs influence learner perceptions of the difficulty of specific programming concepts and skills, ability to debug code and ability to problem solve and plan solutions. This is achieved by comparing questionnaire results from the control and treatment groups. The results are used to answer RQ5: *What impact do the different PATs have on IT learner understanding of programming concepts?*

The first item in the questionnaire requires IT learners to rate the difficulty of programming concepts that are included in the IT subject content and the difficulty of certain programming skills which IT learners are required to develop if they are to achieve the IT learning outcomes. Participants are required to rate each concept or skill appearing in a provided list using a 7-point semantic differential scale where 1 is "Extremely easy" and 7 is "Extremely difficult". Participants can provide a 0 (zero) rating if the concept or skill has not been taught yet. Learners are also asked to identify which one of the 22 programming concepts in the list they consider to be the most difficult (item 2).

The remaining 18 items in the questionnaire are aimed at determining the degree to which IT learners are able to understand simple programming concepts (items 3-6), apply simple programming solutions to other problems or exercises (items 7-10), debug or find errors in their code, problem solve (items 11-15) and plan solutions to problems (items 16-20). Learners are required to rate items on a 7-point semantic differential scale where 1 is "Strongly disagree" and 7 is "Strongly agree". This questionnaire is administered as a pre- and posttest to Grade 11 learners in the control and treatment groups of the experimental research strategy to identify concepts that IT learners have difficulty understanding and to evaluate the effectiveness of the proposed PATs.

Questionnaire data collected from the control group pretest are used to gain insight into the concepts that South African IT learners find difficult, thus addressing RQ1. These results contribute to the selection criteria to identify suitable PATs that may be able to address any problems or difficulties identified (Section 4.3). Control and treatment group pre- and posttest data are analysed to evaluate the impact of the PATs on IT learner understanding of programming concepts (RQ5, Chapter 7).

2.5.1.4 Motivated Strategies for Learning Questionnaire (MSLQ)

The *Motivated Strategies for Learning Questionnaire* (MSLQ), developed by Pintrich, Smith, Garcia and McKeachie (1991), was designed to assess college students' motivational orientations and use of learning strategies in college courses (Artino and Anthony, 2005; Chen, 2002). The final version was presented in 1991 after numerous statistical tests were completed to confirm the reliability and validity of the instrument, including confirmatory factor analysis, internal consistency estimates of reliability (Cronbach's alpha) and zero-order correlations between the different cognitive and motivational scales (Artino and Anthony, 2005). The MSLQ is administered to IT learners during the research study to determine the impact of PATs on IT learner motivation towards programming (RQ6).

The MSLQ was designed to be used by researchers as an instrument for investigating the nature of learner motivation and use of learning strategies. Educators and learners would be able to use the instrument as a means of assessing motivation and study skills in a given course. Scores from the MSLQ have also been used for empirical research to evaluate the effects that instructional interventions including various educational technologies or differing course structures, have on motivational orientation and learning strategies (Artino and Anthony, 2005).

The MSLQ is a self-reporting instrument consisting of 81 items. Learners rate the items on a 7-point semantic scale where 1 represents "Not at all true of me" and 7 represents "Very true of me". The 81 items are divided into two sections: motivational (31 items) and learning strategies (50 items). Items are also further divided into 15 subscales of which six are within the motivational section and nine are within the learning strategies section.

Persmission was obtained to use the motivational section of the MSLQ instrument in this research study to determine the motivational orientation of IT learners. The learning strategies section is not administered to learners as only the motivation of learners in the IT subject is of interest.

The motivational section of the MSLQ instrument assesses learners' goals and value beliefs for a subject, their belief about their skill to succeed and their anxiety about tests. The six subscales that are within the motivational section of the MSLQ are intrinsic goal orientation, extrinsic goal orientation, task value, control of learning beliefs, self-efficacy for learning and performance, and test anxiety (Table 2.3).

Table 2.3: MSLQ Motivational Section Subscales and Components

Motivational Subscale
1. Value Components
a. Intrinsic Goal Orientation
b. Extrinsic Goal Orientation
c. Task Value
2. Expectancy Components
a. Control Beliefs
b. Self-Efficacy for Learning and Performance
3. Affective Components
a. Test Anxiety

The MSLQ instrument is administered to Grade 10 and 11 learners in the control and treatment groups of the experimental research strategy to determine the impact of the proposed PATs on learner motivation. The instrument is administered as a pre- and posttest for both the treatment and the control groups.

2.5.1.5 Visual, Aural, Read/Write, Kinaesthetic (VARK) Questionnaire

The *Visual, Aural, Read/Write, Kinaesthetic (VARK) Questionnaire* was designed by Neil Fleming (Fleming and Baume, 2006; Fleming, 2010) to identify preferred modes of learning. Fleming designed the questionnaire based on his own prior experiences and observations working with students and teachers at Lincoln University. Fleming claims that modal preferences influence a person's behaviours, including learning. A "Younger" version of the questionnaire is a modified version of the main questionnaire for people aged between 12 and 18 and uses wording more appropriate for people in this age group (Fleming, 2010). The VARK Questionnaire is included as a questionnaire in this research to determine the learning preferences of IT learners. Permission to use the VARK questionnaire was obtained.

Table 2.4: VARK learning preferences

Modal Preference	Description	Teaching Strategies
Visual	Seeing or observing	Drawings, charts, symbolic representations, diagrams, demonstrations,video materials
Aural	Auditory or hearing	Explanations or verbal presentations, discussions, chats, verbal instructions
Read/Write	Reading and writing	Information in text format, interaction with textual material, reading and/or writing instructions
Kinaesthetic	Hands-on experience	Physical experience, real or simulated practice, touching, performing activity, lessons that emphasise doing and manipulation of objects

A person's learning preference is the manner in which information about which they are attempting to learn is most efficiently and effectively processed, stored and recalled (Wehrwein, Lujan and DiCarlo, 2007). Table 2.4 outlines the learning preferences and approaches that can be used to help learners (Bednarik and Fränti, 2004; Wehrwein *et al.*, 2007) by presenting information in a manner aligned with their particular preference or combination of preferences. The results of the VARK questionnaire can indicate that a person has a strong preference for one of the four learning modalities and is thus unimodal or has several preferred learning preferences and is thus multimodal. A person with multimodal learning preferences can have balanced preferences between two, three or all four learning modalities (Wehrwein *et al.*, 2007).

The VARK questionnaire administered to Grade 10 and Grade 11 learners in this study is the "Younger Version". The questionnaire consists of 16 multiple-answer questions presenting everyday situations. Participants can indicate one or more answers for each item. The results of the VARK Questionnaire identify the learning preference modes that should be used to present information to IT learners.

2.5.1.6 PAT Evaluation Questionnaire

The purpose of this questionnaire (Appendix E) is to obtain IT learner feedback regarding the PATs. The *PAT Evaluation Questionnaire* is used to obtain IT learner perceptions of the usefulness of the PAT with regards to the understanding of programming concepts.

The ease of use and learnability of the PAT are evaluated and open-ended questions are used for learners to provide opinions about what they liked and disliked about the PAT.

The *PAT Evaluation Questionnaire* is administered to treatment group learners as two versions. Each version is administered at a different point in time during the treatment group evaluation. The first version of the questionnaire is administered up to three months after the PAT is received by treatment group learners. The second version of the questionnaire is administered at the end of the study together with the posttest questionnaires.

The reason for the first version is, firstly, to determine early in the year whether IT learners are using the PAT. Early detection of a large number of learners not using the PAT would allow time for alternative methods to be used in order to promote use. The second reason for the first version of the questionnaire is that, if learners have used the PAT for a short time but have discontinued using it, learners would still be able to evaluate their use of the PAT accurately. Accurate evaluation of the PAT may not be possible if a considerable amount of time has elapsed between the administration of the questionnaire and use of the PAT.

The reason for the second version of the questionnaire is that the first version will be administered before all the programming concepts have been covered in class. Learners may not have used the PAT to practise the implementation of all the programming concepts supported. The second version of the questionnaire administered at the end of the year, only evaluates the programming concepts supported by the PATs (Section 6.2). If the first version of the questionnaire is not administered to treatment group learners owing to reasons beyond the control of the research, only the first version of the questionnaire is administered to treatment group learners, but at the end of the year.

Both versions of the questionnaire require IT learners to indicate whether the PAT was installed and how often the PAT was used. Learners also evaluate the usefulness of the PAT in assisting with the understanding of programming concepts. Only programming concepts supported by the PAT are evaluated. The feedback regarding the programming concepts is used to answer RQ5: *What impact do the different PATs have on IT learner understanding of programming concepts?*

The first version of the questionnaire requires IT learners to rate whether they agree or disagree that the PAT is easy to use, quick to learn to use, if they require assistance to use the PAT and are confident using the PAT. These four questions are derived from the *System Usability Scale (SUS) Questionnaire* (Brooke, 1996), developed to provide a global view of subjective assessments of usability. The questions are rated on a five-point semantic differential scale where 1 represents “Strongly disagree” and 5 represents “Strongly agree”. Results from these questions are used to evaluate the impact of the PAT on IT learner motivation towards programming and are thus used to answer RQ6: *What impact do the PATs have on IT learner motivation towards programming?*

Open-ended questions in the first version are used to obtain feedback regarding what learners liked and disliked about the PAT. Learners are also asked to explain how they would like the PAT improved. If learners are not using the PAT, they are requested to provide reasons explaining why they are not using the PAT. The responses to the open-ended questions are analysed in order to answer RQ5 and RQ6. Learner comments related to the techniques used by the PATs are used to address RQ7: *What techniques should PATs implement in order to assist IT learner understanding of programming concepts?* and RQ8: *What techniques should PATs implement in order to motivate IT learners to learn programming concepts and to use a PAT?*

2.5.2 Class Tests

One of the aims of the research study is to identify PATs that can be used to assist IT learner understanding of programming concepts. Assessments in the form of class tests, examinations (theory and applied) and assessment tasks are completed by IT learners throughout the academic year. These assessments are prepared and marked by the IT teacher. The final paper in Grade 10, Grade 11 and Grade 12 is prepared by the Department of Education. The nature of the questions and the manner in which the concepts are assessed are beyond scope of this research study. However, the end-of-year summative assessment marks for Grade 10 and Grade 11 learners are used to determine the impact of PATs on IT learner understanding of programming concepts (Chapter 7).

Several short class tests (Appendix D) prepared by the researcher are also administered to the treatment and control groups. The class tests are in the form of multiple-choice questions and are similar to the test prepared by Lister, Adams, Fitzgerald, W.Fone, Hamer, Lindholm, McCartney, Moström, Sanders, Seppälä, Simon and Thomas (2004) used in their multi-national study of the reading and tracing skills of novice programmers.

The class tests are administered to Grade 10 and Grade 11 learners in the control and treatment groups of the experimental research strategy. The tests are administered after the relevant programming concept has been completed by the class as a normal class test during one of the IT lessons. All learners in the class write the test, however, only the answer sheets of learners who have consented to participation in the research study are analysed and included in the evaluation of the impact of the PATs.

Identical class tests are administered to the control and treatment groups in each respective academic year. The multiple choice tests assess IT learner knowledge and understanding of specific programming concepts. The programming concept knowledge assessed by the multiple choice class tests are presented in Section 7.2.3.

2.6 Data Analysis

Qualitative data analysis is used to report on the results from the questionnaires administered to IT teachers. The results of the questionnaire are used to describe the situation in South African secondary schools. Qualitative data analysis, specifically content analysis (Cohen *et al.*, 2007), is used to analyse IT learner feedback for open-ended questions describing what features they liked, disliked and would like to have improved, with respect to the PATs (Chapters 7 and 8).

Quantitative data analysis methods, namely descriptive and inferential data analysis methods (Cohen *et al.*, 2007), are used to analyse the questionnaires and class tests administered to IT learners participating in this research study. Descriptive data analysis methods such as the calculation of the sample mean (μ) and standard deviation (s), as well as frequency distribution of responses, are used to report semantic differential scale results. Descriptive statistics are also used to describe the relationship between different variables such as control or treatment group and grade with respect to questionnaire responses in the presentation of results in Chapters 7 and 8.

Most of the questionnaires administered contain items that use a 5- or 7-point semantic differential scale. All semantic differential scale data referred to in this work are analysed as interval ratio data as only the extreme values have descriptions (e.g. “Strongly disagree” and “Strongly agree”) on the questionnaires.

Inferential statistics are used to test the research hypotheses. This includes determining whether the differences in the results of the control and treatment groups are statistically and/or practically significant. Statistical significance is determined at the 95% confidence interval. Analysis of covariance (ANCOVA) methods are used to examine the differences between the control and treatment groups where pre- and posttest data are available. Data collected from the *Perceived Difficulty of Programming Questionnaire*, MSLQ and

Grade 11 IT summative assessment marks are evaluated using this analysis. Analysis of variance (ANOVA) methods are used to determine signficant differences between control and treatment groups where only one dependent variable is available. Multiple choice class results are evaluated using this analysis. One sample t-tests are used to determine the difference between mean scores and the neutral value of rating scale data. *PAT Evaluation Questionnaire* results are evaluated using this analysis. Cohen's d is calculated only for statistically significant results, as a measure of practical significance.

2.7 Risks and Limitations

Several risks are identified with respect to the evaluation phase due to the fact that the research is dependent on the co-operation and support of schools, IT teachers and IT learners. One risk to the study is a sample size smaller than the minimum required for a normal distribution and thus the use of conventional inferential methods. This risk is managed by identifying appropriate analysis techniques that can be used for smaller sample sizes. The data from different schools are grouped together for certain of the descriptive and inferential data analysis performed.

Another risk is that there is no control over how the questionnaires and multiple choice tests are administered to the participating IT learners. IT learners are relied upon to co-operate by returning questionnaires. If questionnaires are not returned, the data for certain IT learners would be incomplete. Consequently, the learner may have to be excluded from certain of the data analyses which would further reduce the sample size.

There is no control over the research in the schools, thus any changes in the teaching environment need to be managed. The impact of any changes in the teaching environment would be monitored and the effect on the research results evaluated, specifically the correlation between the use of the PAT and the effect on IT learner understanding of programming concepts and motivation. Analysis of data from such a school would be considered carefully and the possible impact of this situation noted in the analysis discussion.

2.8 Conclusion

IT learners in South African secondary schools, as novice programmers, need assistance with introductory programming concepts (Chapter 1). This research evaluates the impact of using PATs on IT learner understanding of programming concepts, as well as IT learner motivation towards programming. The primary research objectives (Chapter 1) are achieved using two phases. The purpose of the first phase is to identify PATs that are suitable for use by IT learners (Primary Research Objective 1, as well as Secondary Research Objectives 1.1 and 1.2). Selection criteria (Section 4.4) are formulated from a review of related work in the field of introductory programming and novice programmer literature (Chapter 3), together with feedback from IT teachers and learners in South African secondary schools (Chapter 4).

Questionnaires are administered to IT teachers aimed at identifying difficult concepts as well as methods used to teach IT programming. IT learners' perceptions of IT programming are determined using the *Perceived Difficulty of Programming Questionnaire*. Selection criteria are used to evaluate the suitability of the PATs (Section 5.3) and, where possible, selected PATs are adapted to meet the selection criteria (Chapter 6).

The purpose of the second phase of the study is to evaluate the PATs identified in the first phase (Primary Research Objective 2 and Secondary Research Objective 2.1 and 2.2). A quasi-experimental research strategy is used for the evaluation to cater for the fact that learners cannot be randomly assigned to the control and treatment groups. The IT learner questionnaire results for the *Perceived Difficulty of Programming Questionnaire* and *Motivated Strategies for Learning Questionnaire* are used as pre- and posttest data in the experimental design to test the two secondary research hypotheses (Section 2.3). This is achieved by comparing the control and treatment groups to evaluate the impact of the PATs on learner motivation and perceived difficulty of programming (Phase 2 to achieve Primary Research Objective 2). In addition to the questionnaires, multiple choice class tests to assess learner knowledge of several IT concepts are administered to the control and treatment groups. The multiple choice test results are also used to evaluate the impact of the PATs on learner understanding of programming concepts.

Results are presented indicating whether or not providing IT learners with PATs supports their study of programming concepts (Chapter 7) and their motivation towards programming (Chapter 8). IT learner feedback is used to formulate guidelines describing the particular presentation techniques that a PAT should implement to support novice programmers (Chapter 9).

Chapter 3

Introductory Programming

3.1 Introduction

"Learning to program is difficult."

This statement by Shuhidan *et al.* (2009) is supported by the vast amount of research (Pears *et al.*, 2007) worldwide dedicated to the area of novice programming. Many learners are interested in programming (Gayo-Avello and Fernández-Cuervo, 2003; Robins *et al.*, 2003) but the poor results of students in high school and especially in higher education introductory programming courses (Lister *et al.*, 2004) have prompted much of the research undertaken by various groups. Learners, struggling to understand programming concepts, become frustrated, negative and stressful and, in many cases, this leads to failure in and eventual withdrawal from the subject (Garner, 2007; Haataja, Suhonen, Sutinen and Torvinen, 2001; Robins *et al.*, 2003; Shuhidan *et al.*, 2009).

Learners continue to struggle to cope with introductory programming despite the large amount of research contributions over the last two decades aimed at assisting novice programmers to understand complex programming concepts. These contributions have had a limited effect on classroom practice (Pears *et al.*, 2007; Shuhidan *et al.*, 2009). The large amount of active research (Havenga and Mentz, 2009; Pears *et al.*, 2007; Shuhidan *et al.*, 2009) continuing to take place in this field is indicative of the fact that further initiatives to address the difficulties experienced by novice programmers, are required.

Appropriate PATs to assist novice programmers in South African secondary schools cannot be proposed before the difficulties experienced by novice programmers are identified. The purpose of this chapter is therefore to address RQ1: *What programming difficulties and skills do PATs need to address and develop, respectively?*

This chapter provides a discussion of general novice programming difficulties identified in literature while Chapter 4 addresses RQ1 further in the context of South African secondary schools. The difficulties identified in these chapters, are used to formulate selection criteria for PATs that can address these difficulties, thus achieving Secondary Objective 1.1: *Formulate selection criteria for determining the suitability of PATs to support the achievement of programming learning outcomes in the IT subject curriculum.*

The results of novice programming research that attempt to identify the reasons why novice programmers struggle with programming concepts and introductory programming courses are explored (Section 3.2). The suggested solutions to overcome the difficulties faced by novice programmers and the success of these solutions are also discussed (Section 3.3).

3.2 Difficulties of Learning to Program

Novice programmer dissatisfaction and frustration, poor assessment results and a decline of interest in computer science courses indicate that difficulties are experienced by novice programmers with introductory programming content. Research studies have been done to identify why these difficulties are experienced by students (Gomes and Mendes, 2007). The difficulties that learners experience while learning to program are attributed to a lack of programming skills and abilities (Section 3.2.1), style of teaching approach (Section 3.2.2), programming language and environment used (Section 3.2.3) and specific types of programming concepts (Section 3.2.4).

3.2.1 Programming Skills and Knowledge

Programming is a complex activity requiring the learner to learn non-trivial new concepts, facts and skills (Baldwin and Kuljis, 2000). Figure 3.1 summarises the relationship between the different types of knowledge required by a novice programmer and code comprehension and generation.

Novice programmers need to understand that code generation involves three sequential steps (Figure 3.1):

1. Studying a given problem statement or set of requirements and deciding on the best programming strategy to use.
2. Producing an algorithm to solve the problem. The algorithm will often be formulated using pseudocode.

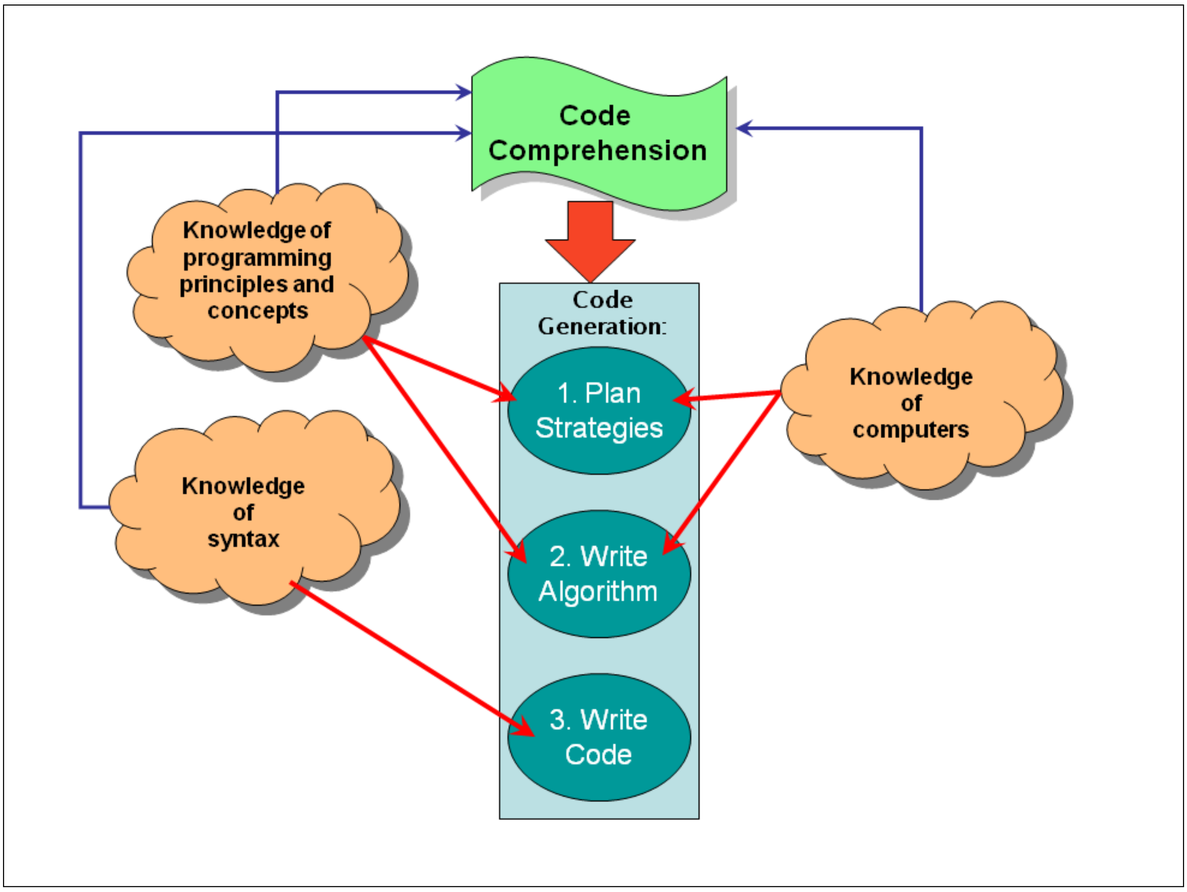


Figure 3.1: Knowledge and skills required by a programmer

3. Translating the algorithm into the code of the programming language being used. Testing and changing the program code until the program meets the original set of requirements and thus solve the problem.

A programmer requires certain abilities and knowledge in order to achieve the steps described and to formulate the solution. The aspects which influence programming ability are the application of knowledge and strategies and the relationship between code comprehension and code generation (de Raadt, 2008; Robins *et al.*, 2003).

Programming knowledge and *strategies* are distinct but related (de Raadt, 2008). *Programming knowledge* relates to the body of information describing programming concepts and principles (for example: how a *for*-loop works or the purpose of a variable), knowledge of computers (for example: how an event is generated) and programming language knowledge or *syntax*[1]. If a programmer is not aware of concepts such as loops, data structures or objects, it would be difficult for the programmer to plan a suitable solution. Knowledge of the syntax is required to implement a solution in a particular programming

[1] *Syntax* refers to the way in which keywords, names and symbols can be combined to create expressions and statements in a language without consideration to their meaning (Seaton, 2007; Slonneger and Kurtz, 1995)

language (Pears *et al.*, 2007). Regardless of how well the solution is designed, if the syntax is incorrect, the program will not run successfully. Knowledge in all three of these areas is required for a novice programmer to be successful in programming.

Programming strategy is the way in which programming knowledge is applied to solve a particular problem (de Raadt, 2008; Robins *et al.*, 2003). Programming strategies are more abstract than programming knowledge and are thus applicable regardless of programming language (de Raadt, 2008).

Programming strategies are required for the first step of code generation (Figure 3.1), while different programming knowledge areas are required for different steps of the code generation process. Novice programmers who lack either programming knowledge or strategies will struggle with programming. For example, novice programmers may understand the syntax of a certain concept but struggle to use the concept in context or transfer their knowledge to solve similar problems (Lahtinen *et al.*, 2005; de Raadt, 2008), thus lacking programming strategy. If the syntax is correct, the program will compile even though the program may not solve the problem (Al-Imamy *et al.*, 2006). Alternatively, a program that uses the correct programming strategies will not compile if there are errors in the syntax.

Teachers need to develop both a learner's programming knowledge and programming strategies. In general, however, more time is spent teaching programming language knowledge than programming strategy (Al-Imamy *et al.*, 2006; Pears *et al.*, 2007). Novice programmers often combine Steps 1 and 2 of the code generation process (Figure 3.1) as they attempt to solve the algorithm in a particular programming language (Garner, 2007; Rongas *et al.*, 2004). Lack of planning or lack of algorithm design leads to difficulty in solving a given problem (Shuhidan *et al.*, 2009). A novice programmer should also have adequate knowledge in all three programming knowledge areas to comprehend and generate code successfully (Figure 3.1).

Several researchers (Heines and Schedlbauer, 2007; Lahtinen *et al.*, 2005; Lister *et al.*, 2004; Robins *et al.*, 2003) agree that many novice programmers lack knowledge of abstract programming concepts. This deficiency results in a limited surface knowledge of programs and a fragile grasp of basic programming principles and concepts (Lahtinen *et al.*, 2005; Lister *et al.*, 2004). Novice programmers are unable to identify the connection between the techniques and the problems or transfer their knowledge from one problem to another (Heines and Schedlbauer, 2007; Hooper, Carr, Davis, Millard, White and Wills, 2007).

Another reason, highlighted in similar studies, for the difficulty novice programmers have in programming successfully is that they lack knowledge of a computer and thus struggle to understand how a computer sequentially executes program code (Levy *et al.*, 2001; Rongas *et al.*, 2004; Lahtinen *et al.*, 2005). Novice programmers consequently lack a viable mental model of how a computer works (Levy *et al.*, 2001; Kumar, 2006) which contributes to their poor knowledge of programming (Kumar, 2006). One result of this misunderstanding is that novice programmers may use the correct instructions and syntax but the instructions are written in an incorrect order (Rongas *et al.*, 2004).

A problem related to the lack of comprehension of sequential execution of program code is that many novice programmers have difficulty reading and understanding code (Ala-Mutka, 2003; Rongas *et al.*, 2004). Program comprehension is described as a "cognitively complex skill" (Bednarik and Tukiainen, 2006; Lister *et al.*, 2004). Research results have identified that novice programmers struggle to follow the execution of code or understand how the state of a variable changes from one line to another (Lahtinen *et al.*, 2005; Vainio and Sajaniemi, 2007). Novice programmers reading code should be able to identify knowledge, such as concepts used in the solution and the strategy applied to produce the solution (de Raadt, 2008). Novice programmers therefore require an adequate knowledge of all three programming areas to be able to comprehend code (Figure 3.1).

A correlation has been shown to exist between code comprehension and code generation (de Raadt, 2008; Lister, Simon, Thompson, Whalley and Prasad, 2006). Novice programmers who are unable to read and understand code are unable to write similar code (Lister *et al.*, 2006). This is a problem as new concepts are explained to novice programmers using existing practical examples. If novice programmers are unable to read and understand code solutions, they will be unable to build their knowledge of programming concepts and strategies to solve real-world problems (Lister *et al.*, 2004, 2006). The ability to read and understand code is moreover important for finding bugs or logical errors in code (Lister *et al.*, 2006). Debugging is a *programming skill* that is important in order to program successfully (Bednarik and Tukiainen, 2006).

An explanation for the inability of novice programmers to transfer their knowledge of problem solutions to similar problems is that they lack program comprehension and have a fragile knowledge of programming principles. Their knowledge is not deep enough to understand how the principles can be applied to new or larger problems that are more complex. Many researchers have indicated that novice programmers lack problem solving ability (Heines and Schedlbauer, 2007; Lahtinen *et al.*, 2005; Lister *et al.*, 2004; Robins *et al.*, 2003). However, novice programmers may be capable of problem solving but be unable to express solutions in a programming language or in manner that a computer would understand. This is supported by the finding that novice programmers have a non-viable mental model of programming concepts which results in misconceptions and difficulties when trying to solve programming problems (Ma, Ferguson, Roper, Ross and Wood, 2008).

Code comprehension, code generation and problem solving ability are important *programming skills* required by novice programmers in order to become successful programmers. However, these high-order skills require knowledge of programming principles and concepts, syntax and computers together with programming strategies (Figure 3.1). Novice programmers in an introductory programming course learn the knowledge and skills required from their teacher. The teaching approach is important to consider as it may contribute to the difficulties experienced by novice programmers.

3.2.2 Teaching Approach

A novice programming student (learner at school or student at university) is exposed to programming through the teacher presenting the subject. The programming language knowledge, programming concepts, syntax, programming environment and problem solving strategies are all introduced to the students by the teacher and by other resources, for example, textbooks.

Teachers may overemphasise trying to teach the students *how to*, for example, how to use a *for*-loop or variable in a particular problem. Students may be unable to transfer what they have learnt as their knowledge of the underlying skills and concepts is limited (Baldwin and Kuljis, 2000). Overemphasising *why* will provide students with a wider knowledge base of the underlying principles. However, this large amount of theory should be balanced with the practical experience of applying the principles, concepts and skills to problems.

There is a shortage of qualified IT or Computer Science teachers who are able to teach programming (Haataja *et al.*, 2001; Havenga and Mentz, 2009). Many times teachers are enthusiastic about the subject and are dedicated to assisting students. However, teachers face many challenges imposed by the time available in the academic year (Al-Imamy *et al.*, 2006) and subject content prescribed by the curriculum (Kölling and Rosenberg, 2002). Teachers may also lack experience teaching programming concepts that were not always included in the subject content outline (Kölling and Rosenberg, 2001, 2002). The lack of experience would result in the teachers not being able to assist the student to develop a proper mental model or correct knowledge.

Different people create different mental models of concepts, especially abstract concepts. If the teacher is not able to explain a concept in a way that a particular student would understand, that student may develop a poor or incorrect model of the concept. An incorrect mental model would affect the student's understanding of the concepts and result in the student becoming frustrated and losing confidence when material that is more complex is taught (Shuhidan *et al.*, 2009).

Each student has a preferred learning style (Lahtinen *et al.*, 2005; Rongas *et al.*, 2004) (Appendix H). Students also have differing abilities, learning speeds and attitudes or motivations towards the subject (Lahtinen *et al.*, 2005). In large classes, it is extremely difficult for a teacher to explain a programming concept in a way that everyone in the class can understand. Large class sizes make individual student contact very difficult (Rongas *et al.*, 2004).

Teachers have been identified as assisting students to develop misconceptions and misunderstandings (Baldwin and Kuljis, 2000). This is due to the fact that teachers are criticized for failing to develop student understanding of key concepts and skills such as program comprehension. It is claimed that teachers focus more on programming language syntax. Often, it is difficult for the teacher to go beyond syntax coverage. The teacher is struggling to ensure that all students understand the syntax which is difficult when using a one-size-fits-all teaching approach to teach students of different backgrounds and learning abilities (Al-Imamy *et al.*, 2006). There is simply not enough class time for the teacher to cover everything.

Introductory programming teachers in school learning environments are expected to address individual learner difficulties and learning abilities while structuring short daily lessons in such a way that the time is utilised effectively. Adequate time should be allocated for theoretical explanations as well as practical implementation of concepts in class, to allow the teacher to be able to assist learners. No matter how effective the teaching approach employed is, it may still not be able to address or avoid contributing to the difficulties faced by individual learners due to the school learning environment.

The method used by the teacher to introduce programming concepts to learners can be an influential factor contributing to the difficulties experienced by novice programmers (Kunkle, 2010). PATs may be able to assist teachers to improve learner understanding by helping IT learners to develop proper mental models of abstract programming concepts, cater for different learning styles and promote self-learning and exploration by learners.

3.2.3 Programming Language and Environment

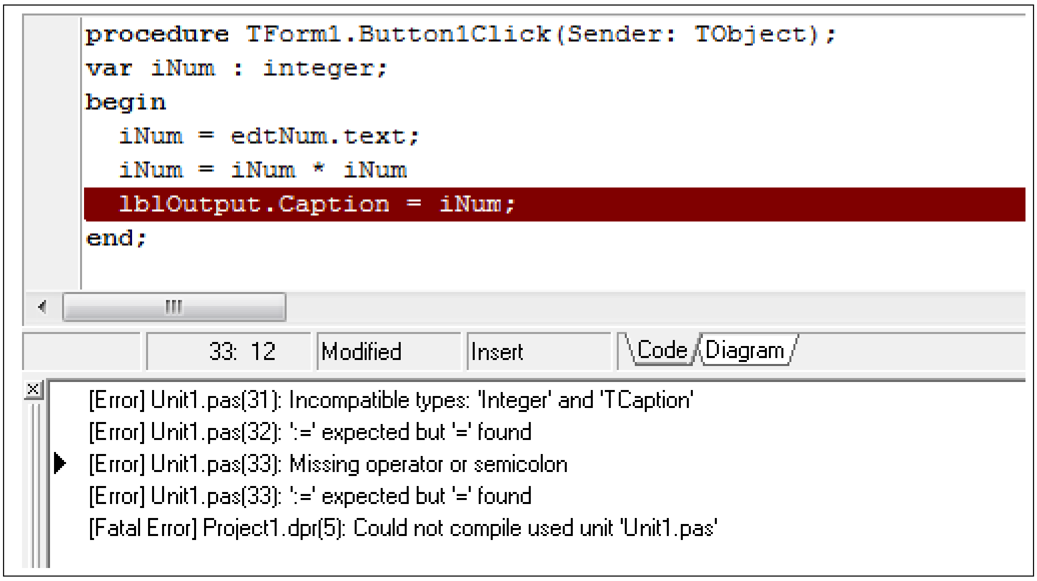
The programming language and development environment used are the software tools that allow novice programmers to experience programming. The most appropriate programming language for teaching introductory programming has been debated extensively (Mannila and de Raadt, 2006; Pendergast, 2006; Pears *et al.*, 2007). The language is chosen based on factors such as relevance in industry, the availability and cost of programming development environments (for educational purposes), educational aspects of the language and preference of faculties or other controlling institutions (Pears *et al.*, 2007). The choice of language is important as the programming languages and the development environments used to develop programs can contribute to the difficulties experienced by novice programmers (Pears *et al.*, 2007).

Certain programming languages are too complex to explain or use to teach some of the programming concepts taught to novice programmers (Kölling, 1999). A student may have difficulty understanding the initial explanation or definition of a concept. Further explanation of the concept using a code example should not confuse the student any further. This would be the case if the implementation of the concept as code is complicated.

Similarly, programming development environments for writing code and compiling and running programs can be confusing to students (Kölling, 1999). A professional programming development environment may overwhelm the students by presenting them with functionality and interfaces not needed by novice programmers (Kölling, 1999; Levy *et al.*, 2001; Pendergast, 2006). Furthermore, the compiler messages provided by professional programming development environments are directed at professional programmers (Rongas *et al.*, 2004).

Program compilers ensure that programs are syntactically and semantically correct before executing. If an error is found in the code, a compiler message is displayed to the programmer describing the error and providing the location of the error in the code. Most compiler messages are too complicated and low-level and do not explain sufficiently well to a novice programmer which syntactical or semantic errors have occurred.

Figure 3.2 is a screenshot of compiler messages provided for code that is not syntactically correct. Missing semi-colon messages are associated with the code line following the line where the semi-colon should be. Vague error messages such as the identification of incompatible types do not provide programmers with information on how the error can be corrected. In professional programming development environments, novice programmers struggle to understand compiler error messages intended for professional programmers



```
procedure TForm1.Button1Click(Sender: TObject);
var iNum : integer;
begin
  iNum = edtNum.text;
  iNum = iNum * iNum
  lblOutput.Caption = iNum;
end;
```

```
[Error] Unit1.pas(31): Incompatible types: 'Integer' and 'TCaption'
[Error] Unit1.pas(32): ':=' expected but '=' found
[Error] Unit1.pas(33): Missing operator or semicolon
[Error] Unit1.pas(33): ':=' expected but '=' found
[Fatal Error] Project1.dpr(5): Could not compile used unit 'Unit1.pas'
```

Figure 3.2: Screenshot of Delphi compiler error messages

with a better understanding of the programming environment and more programming expertise.

Programming languages and development environments can contribute to the difficulties faced by novice programmers. However, certain programming concepts are more difficult for novice programmers to understand, regardless of the programming language or development environment.

3.2.4 Programming Concepts

Novice programmers struggle to understand abstract concepts and data types with no real-life metaphor presented in introductory programming courses (Hu, 2008). Accurate mental models of these concepts need to be developed if the student is to succeed.

Several programming concepts which novice programmers struggle to understand, have been identified, namely:

- *if*-statements and logical operations (Haataja *et al.*, 2001)
- Control structures, namely sequence, repetition and selection (Brewer, 2009; Gayo-Avello and Fernández-Cuervo, 2003)
- Nested loops (Shuhidan *et al.*, 2009)
- Abstract data types such as arrays (Haataja *et al.*, 2001; Lahtinen *et al.*, 2005)

- Procedures and functions (Gayo-Avello and Fernández-Cuervo, 2003; Haataja *et al.*, 2001)
- Debugging errors in code (Lahtinen *et al.*, 2005)
- Object-oriented programming (OOP) (Cooper, Dann and Pausch, 2003; Jones, Boyle and Pickard, 2003; Kölling, 1999; Kunkle, 2010)
- Recursion (Haataja *et al.*, 2001; Lahtinen *et al.*, 2005; Shuhidan *et al.*, 2009)

Recursion is the only difficult concept identified which is not included in the IT syllabus in South African secondary schools (Department of Education, 2008).

The reason why novice programmers may struggle with *if*-statements is related to their understanding of booleans and logical operators used when the condition is tested (Tew, McCracken and Guzdial, 2005). This would also be applicable to repetition control structures and nested loops where statements are repeated while a certain condition is met. The difficulties experienced with arrays could be associated with an incorrect understanding of the use of the array index (Tew *et al.*, 2005). The difficulty associated with procedures and functions is that novice programmers would need to understand larger entities of the program in order to divide the functionality of the code into, and be able to call, procedures and functions (Lahtinen *et al.*, 2005).

Natural language transfer is a factor that may be the cause of difficulties associated with control structures where the English meaning of a word is confused with its meaning in a particular programming language (Bonar and Soloway, 1989). For example, *while* is considered a continually active test in natural language (Bonar and Soloway, 1989). Novice programmers may also have difficulty interpreting an assignment statement such as *a := a + 1;* as this differs from what they have been taught in algebra (Putnam, Sleeman, Baxter and Kuspa, 1989).

Another difficult programming concept for novice programmers is object-oriented programming (OOP) (Cooper *et al.*, 2003; Jones *et al.*, 2003; Kölling, 1999; Kunkle, 2010). There is a debate how best to introduce OOP to novice programmers. The most common method of instruction in introductory programming is to start with simple concepts and programs and gradually advance to more difficult concepts and complex exercises (Cooper *et al.*, 2003). The complexities of understanding the different aspects of OOP such as objects, classes, instantiation and inheritance are too complex and abstract for a novice when starting to program (Hu, 2008; Kölling, 1999). A novice programmer learning object instantiation and method calls before any other concepts such as *for*-loops and *if*-statements, is likened to teaching a child sentences before words (Hu, 2008). However, when OOP is taught last, many teachers find OOP difficult to teach due to the paradigm shift from structured methods (Cooper *et al.*, 2003; Kölling, 1999).

3.3 Approaches to Address Programming Difficulties

A vast amount of research has been done to identify the difficulties novice programmers face (Al-Imamy *et al.*, 2006; de Raadt, 2008; Gayo-Avello and Fernández-Cuervo, 2003; Lister *et al.*, 2004; Kunkle, 2010; Robins *et al.*, 2003). In addition to identifying and explaining the difficulties, approaches to overcome these problems are suggested in many of the research studies. The approaches are aimed at making learning to program easier for novice programmers. Several ways of overcoming novice programming difficulties are presented in this section.

The *lack of problem solving ability* has also been identified as an important problem affecting novice programmers (Section 3.2.1). A solution suggested to improve problem solving is the use of a customised PAT (Al-Imamy *et al.*, 2006) which allows novices to focus on planning a solution without having to worry about syntax. Problem solving ability can also be developed and enhanced by using PATs that can guide students when writing programming constructs. Common errors can be avoided and students will be able to learn faster and with more confidence.

It is accepted that novice programmers require *syntactic and conceptual knowledge together with programming strategies* (Al-Imamy *et al.*, 2006; de Raadt, 2008). It is suggested that more emphasis be placed on design and creative thinking or problem solving skills (Al-Imamy *et al.*, 2006). Students should also be assisted in the learning process by being provided with templates to guide and scaffold the building of knowledge. The learning process should become more language independent - a learner should be able to demonstrate understanding of concepts by being able to implement a concept in any programming language. This is often not possible in many classroom environments but the deeper understanding of programming concepts and principles is important.

Novice programmers must be able to *develop a proper mental model of programming principles and concepts* if they are to apply these principles and concepts correctly to different programming problems. It is therefore important that novice programmers understand the basic programming concepts (Lahtinen *et al.*, 2005) and that the correct definitions and explanations are provided (Baldwin and Kuljis, 2000; Hu, 2008). For example, without a proper understanding of the flow of control in a program, students will be unable to plan and program solutions effectively, especially as the exercises become more complex.

Theoretical knowledge of programming concepts alone is meaningless for a programmer. Programming is a practical skill and *students must be able to apply their knowledge.* Exercises and examples are therefore very important in a novice programmer's learning process. Simple examples focusing on one or two concepts allow students to build their initial understanding of a concept (Lahtinen *et al.*, 2005). Thereafter, the student should practise generating code by doing many practical exercises. This will assist the students to increase their knowledge, understanding and use of strategies (Rongas *et al.*, 2004).

An important aspect of using exercises and examples to improve learning is that students need to *be able to read and understand the code* (Lopez, Whalley, Robbins and Lister, 2008). Students are encouraged to work through as many examples as possible to build their knowledge and understanding of the code gradually (Rongas *et al.*, 2004). PATs that indicate flow of execution through code can assist novice programmers to improve their tracing skills (Ala-Mutka, 2003).

A student's understanding of programming can be influenced by collaboration or by working alone. Survey results in the research done by Lahtinen *et al.* (2005) showed that *students prefer working and studying alone.* Working alone on exercises was more useful to students than lectures or group practical sessions. This is supported by Baldwin and Kuljis (2000) who state that students need to teach themselves. This can be accomplished with the assistance of computer programming assistance or learning tools.

Alternatively, *collaboration amongst students is encouraged* by Al-Imamy *et al.* (2006). Students are able to discuss and solve problems as a team. Collaboration may be beneficial if all students actively participate in the collaboration. Williams, Wiebe, Yang, Ferzli and Miller (2002) have researched the benefits of pair programming. Students involved in pair-programming during practical sessions were more likely to succeed in the course, performed better on programming projects, were more self-sufficient and less reliant on teaching staff.

The approaches discussed above have resulted from research investigating the difficulties of learning to program. The large amount of ongoing research in the field of novice programming, as well as a lack of consensus on certain issues, is an indication that these approaches may not assist with all the difficulties with which novice programmers are faced. An approach that may not be successful in improving learners' understanding or overall assessment marks, could, instead, ensure that learners remain confident that they can succeed and that they are motivated to continue to improve their understanding of programming principles and knowledge. Motivation is a contributing factor to the success and building of knowledge in programming courses (Fidge and Teague, 2009).

3.4 Conclusion

This chapter investigates the difficulty of introductory programming in related research studies in order to address RQ1: *What programming difficulties and skills do PATs need to address and develop, respectively?* and achieve Secondary Objective 1.1: *Formulate selection criteria for determining the suitability of PATs to support the achievement of programming learning outcomes in the IT subject curriculum.* This chapter has shown that researchers agree that novice programmers find it difficult to learn to program. Despite various proposals, there is also no clear or easy solution to make learning to program easier for novice programmers to understand.

Table 3.1: Programming knowledge and skills required to generate code

Programming Knowledge/Skill	**Code Generation**
Knowledge of programming principles	Step 1: Planning and identifying strategies
Knowledge of computers	Step 2: Writing the algorithm
Knowledge of syntax	Step 3: Implementing algorithm in specific programming language

Specific *programming knowledge* and *programming skills* are required by novice programmers in order to be successful. In order to generate code successfully, a programmer should be able to comprehend code. A programmer will not be able to understand a segment of code unless there is knowledge of programming principles and concepts, syntax and the computer (Table 3.1). A program is generated in three main steps: planning or identifying the strategies required to solve the problem, writing the algorithm and then generating the actual code for the algorithm in a specific programming language. Syntax of a particular programming language is required for the final step and knowledge of programming principles and knowledge of how the computer will execute the solution, are required for the first two. Programmers that lack knowledge or ability in any one or more of these areas will have difficulty with programming.

Table 3.2: Criteria for selecting PATs identified from introductory programming literature

Category	Criteria Item
Programming knowledge	Assists with the understanding of code execution
	Assists with the learning of programming language syntax
	Assists with developing knowledge of programming principles and concepts
	Constructivist to promote self-study
	Programming environment suitable for novice programmers
	Assists with the application of programming knowledge (programming strategies)
Programming skills	Provides simple error messages to assist with debugging
	Promotes problem solving
	Develops code comprehension

In order for PATs to support the novice programmer, PATs should assist learners to develop programming knowledge in three areas, namely, computer execution of program code, programming language syntax and programming principles and concepts (Table 3.1). Addressing these three areas together with methods of improving code comprehension, should implicitly assist users to improve problem solving ability. PATs should also support the development of programming strategies in terms of applying programming knowledge.

PATs selected should promote self-study (Table 3.2) for the learner to improve knowledge in spite of the difficulties related to the teaching approach and programming language and development environment. PATs appropriate for use by IT learners should promote debugging by identifying errors in code or solution creation and provide simple error messages that users can understand. The specific programming language addressed by PATs identified in this research is restricted to Delphi. However, the programming environment, that is, the interface provided by the PAT, should not confuse learners.

There are programming concepts that are more difficult for novice programmers to understand. The following *programming concepts* identified as difficult by research (Section 3.2.4) are included in the Information Technology subject framework for South African secondary schools (Department of Education, 2008):

- *if*-statements
- Nested loops
- Control structures

- Abstract data types, including arrays
- Procedures and functions
- OOP

Novice programmers have difficulty developing an accurate mental model of these concepts due to their abstract nature. The selection of PATs to assist IT learners as novice programmers should assist learners to develop a clear understanding of these concepts.

This chapter identifies selection criteria to address general novice programming difficulties categorised as *programming knowledge* and *programming skills* required by novice programmers (RQ1). Specific *programming concepts* that PATs should address have also been identified. The difficulties identified in this chapter are combined with the specific problems and challenges faced by South African IT teachers and learners (Chapter 4) and used in the formulation of the selection criteria for the PATs (Chapter 4) to achieve Secondary Objective 1.1. Thereafter, PATs that meet these selection criteria are identified (Chapter 5).

Chapter 4

Selection Criteria for PATs to Support IT Programming

4.1 Introduction

Programming is difficult for novice programmers regardless of whether they are IT learners at high school or computing science students at university. Literature shows that novice programmer difficulties can be attributed to a lack of knowledge of programming concepts, programming language syntax and how a computer executes code (Section 3.2.1) as well as the teaching approach (Section 3.2.2) and programming development environment (Section 3.2.3). RQ1: *What programming difficulties and skills do PATs need to address and develop, respectively?* has been partially addressed by identifying the introductory programming difficulties from related research (Section 3.2).

One of the aims of this research study is to provide suitable PATs for South African IT learners. RQ1 is addressed further by identifying the difficulties in programming concepts, skills and knowledge included in the IT subject curriculum with which IT learners need assistance (Section 4.2 and 4.3). Feedback from IT teachers and learners is also used to answer RQ2: *What factors may influence the use of PATs in SA secondary school learning environments?*. Feedback from IT teachers on how the IT subject content, particularly programming concepts, is taught and managed in South African secondary schools is provided to address RQ2 (Section 4.2). IT learner questionnaire (Appendix C) responses are used to identify specific programming concepts, knowledge and skills with which IT learners have difficulty (Section 4.3.1).

The programming difficulties experienced by novice programmers in general (Section 3.2) contribute to the selection criteria (Section 3.4), which are combined with selection criteria derived from IT teacher and learner feedback regarding the programming difficulties experienced by IT learners (Section 4.4). Selection criteria in this chapter are categorised according to the three categories identified in Chapter 3. *Programming concepts* criteria include concepts IT learners are required to know as defined by the IT curriculum, *programming knowledge* criteria assist IT learner understanding of programming concepts and principles and *programming skills* criteria assist IT learner ability to perform programming related skills such as tracing code execution. The results of Chapter 3 together with the findings of this chapter achieve Secondary Objective 1.1: *Formulate selection criteria for determining the suitability of PATs to support the achievement of programming learning outcomes in the IT subject curriculum.*

The selection criteria formulated (Section 4.4) are used to evaluate PATs suitable for use by IT learners in South African secondary schools (Section 5.4). Based on the evaluation using the selection criteria, PATs are selected for use by treatment group IT learners in the experimental phase of this research study (Section 2.2.2).

Participants in this research study - IT teachers and learners - are sampled from four (67%) secondary schools in Port Elizabeth, South Africa, which offer IT as a subject and consented to participate in the research. The research was restricted to schools in the Port Elizabeth area to allow easy access to participants during the research study for the administration of tests and questionnaires.

4.2 IT Teacher Feedback

This section presents IT teacher experiences with regard to teaching the IT subject and programming in particular. The teaching approaches used by IT teachers (Section 4.2.1) of the South African secondary schools participating in this research study and the factors affecting the teaching of IT in the school (Section 4.2.2) are outlined. The IT teacher feedback is used to contribute to the selection criteria for the evaluation and identification of PATs (Section 4.2.3) that are used by IT learners in the experimental phase of the research study (Section 2.2.2).

4.2.1 Teaching IT in South African Secondary Schools

IT teachers at the four secondary schools participating in this research study completed a questionnaire (Appendix A.1) consisting of open-ended questions designed to gather information regarding the current IT teaching environments and the participating IT teacher's opinions regarding the difficulties faced by IT teachers. The purpose of the questionnaire is to identify any challenges faced by IT teachers with regard to the IT subject content and the teaching environments in the participating schools.

Qualitative data analysis methods are used to identify common statements and themes in the responses of IT teachers. Responses to the different questions are combined to provide overall feedback regarding the challenges in the IT teaching environment and methods employed by IT teachers to address these challenges (Table 4.1). Only statements confirmed by two or more IT teachers have been included.

Table 4.1: Summary of IT teacher feedback related to the South African school teaching environment

Theme	**Statement**	n
Positive	Smaller IT classes allow more individual contact time.	4
	Practical nature of subject (not just theory).	2
	Talented, stronger, more motivated learners take IT.	2
Negative	Lessons are too short to practise programming effectively.	4
	Learners who are struggling are asked to attend extra lessons.	3
	IT is on par with Mathematics & Science and more difficult than other subjects.	3
	IT only starts in Grade 10, unlike other subjects of similar difficulty.	2

Positive feedback from IT teachers regarding the IT subject (Table 4.1) indicate that IT class sizes are generally smaller than the classes for other subjects thus IT teachers can provide their learners with more individual contact time (n=4). The practical nature of the subject allows IT teachers to make lessons interesting for learners as the content is not only theoretical (n=2). Another positive point identified by IT teachers (n=2) is that a greater number of talented learners tend to take IT.

Negative points identified by IT teachers are that the IT subject content is similar in difficulty to subjects such as Mathematics and Science (n=3). However, unlike these subjects, IT is only started in Grade 10 (n=2), allowing less time to develop a proper understanding of the subject content as IT learners are not exposed to programming content in the preceding grades. Another negative point identified by IT teachers is that time is a challenge for the learning of IT content, particularly programming, in South African secondary schools. Forty-five minute lessons are too short to practise programming content effectively (n=4) - one teacher indicated that learners "*take a while to get into coding mode*". The IT teacher has to assume that IT learners do not have access to the programming development environment outside of the allocated IT subject time at school and thus the majority of work needs to be done in class. This is difficult for weaker learners as they require more time than is available during school to understand concepts and practise exercises.

IT teachers encourage IT learners to practise coding longer than the allocated class time per day as regular, consistent practice is important to develop programming skills and knowledge. However, this is difficult for IT learners as not all IT learners may have the Delphi development environment installed on their privately owned PC's (n=1). Borland Delphi is one of the development environments used in SA secondary schools. Borland Delphi is a commercial software package that must be purchased, which is not feasible for all IT learners. Other non-commercial Delphi development environments are available but lack certain of the graphical components that Borland Delphi offers and are then not suitable for school IT subject use. Access to the programming development environment or a privately owned computer is not a prerequisite for an IT learner to take IT as a subject. Extra lessons are arranged by IT teachers to allow IT learners more access to the Delphi programming environment and provide IT learners with more individual time to assist with difficulties (n=3), but not all learners can or want to attend.

4.2.2 Teaching IT Programming Content

The IT teachers participating in this research study also completed a questionnaire (Appendix A.2) designed to gather information regarding the methods used to teach programming concepts to their learners. All questions require IT teachers to provide a rating using a 7-point semantic differential scale.

IT teachers provided a rating of the difficulty of programming concepts and skills in terms of perceived learner understanding. Table 4.2 indicates programming concepts that IT teachers believed were difficult for IT learners to understand. Concepts were rated from 1 (extremely easy) to 7 (extremely difficult). Programming concepts are categorised as *difficult concepts* if 50% (n=2) or more of the sample rated the concept as difficult (5-7 rating).

IT teachers rated objects and classes as the most difficult programming concept for IT learners to understand. Correct use of parameters had the second highest mean rating even though the mean ratings for procedures and functions are slightly above neutral. Parameters are used to pass values to procedures and functions.

IT teachers also rated the difficulty of programming skills in terms of learner understanding. All the IT teachers (n=4) rated problem solving, planning of solutions and algorithms as difficult (ratings of 5-7). An algorithm is defined as a finite set of steps for solving

Table 4.2: IT teacher feedback: Difficulty of Programming Concepts (n=4)

	Programming Concept	**Mean**	**SD**
Difficult Concepts	Objects & classes	6.3	0.5
	Planning (skill)	6.0	1.2
	Problem solving (skill)	5.5	0.6
	Correct use of parameters	5.5	1.3
	Algorithms (skill)	5.3	0.5
	Two-dimensional arrays	5.3	1.7
	Debugging (skill)	5.0	1.0
	String handling	5.0	1.6
	One-dimensional arrays	4.8	1.3
	Accessing a database	4.8	1.3
	Case-statements (Delphi)	4.5	1.3
	while-loops	4.5	1.3
	repeat-loops	4.5	1.3
	Procedures	4.5	1.3
	Functions	4.5	1.3
	SQL statements	4.0	1.2
	File handling	3.5	0.6
	Variables	3.3	1.5
	for-loops	3.0	1.2
	if-statements	3.0	0.8
	Input	2.5	0.6
	Output	2.3	0.5

a problem or computing a result (Vickers, 2009). An algorithm would be formulated as a set of code instructions after the problem solving step (Section 3.2.1). Debugging was rated as a difficult programming skill by 50% of the IT teachers (n=2).

The remaining items in the questionnaire (Appendix A.2), require IT teachers to rate their agreement with statements related to the method used to administer programming exercises to learners as well as IT learners' debugging skills. 75% of the teachers indicated that most of the programming exercises provided to learners are for completion during class time. The IT teachers provide their learners with solutions to exercises (n=4), but only check exercises solutions if requested by the learner (n=3).

All the IT teachers (n=4) indicated that learners have difficulty applying programming concepts, that they have learnt, to different problems and exercises. 75% of the teachers disagreed with the statement that there is enough time in class to ensure that each learner understands the concepts satisfactorily.

Two of the IT teachers agreed with the statement that learners who struggle have difficulty with the programming language syntax, compared to one teacher who disagreed and one neutral response (rating of 4). Most of the IT teachers (*n*=3) indicated that learners do not understand compiler error messages or how to use compiler error messages to assist them to identify syntax errors. Only one teacher indicated that learners are taught how to use breakpoints to debug code for errors. Creating breakpoints within the code is a feature available within the programming environment that allows programmers to step through code execution.

None of the IT teachers currently provides or recommends a PAT to their learners to use. The teachers indicated that they would like to provide a PAT to their learners (*n*=4), although two of the teachers feel that time spent introducing a new tool to their learners in the class may be a negative factor.

4.2.3 Selection Criteria from IT Teacher Feedback

Selection criteria have been identified to address points arising from the IT teacher feedback. The criteria are also categorised as *Programming Knowledge* (PK) or *Programming Skill* (PS) to identify whether the criteria address a point related to understanding of programming concepts or to improving programming skills, respectively.

Constructivist to promote self-study (PK): As short lessons result in less time in class to practise programming with the assistance of the IT teacher, learners should be able to use the PAT without the assistance of the teacher to build their knowledge and explore different programming concepts (Areias and Mendes, 2007).

Develops knowledge of programming principles and concepts (PK): PATs that provide an explanation of programming concepts and principles can improve learner understanding of concepts during self-study.

Assists with the application of programming knowledge (PK): Learners struggle to apply programming knowledge and can be assisted if the PAT provides support in the application of programming knowledge to different problems and exercises. An example would be support in the form of scaffolding that assists the learner to plan a code or non-code solution (Wood, Bruner and Ross, 1976). When the scaffolding is removed, the learner should be able to perform the task independently (Kunkle, 2010).

Assist with learning syntax knowledge (PK): IT teachers indicated that struggling learners have difficulty with programming syntax. Improving syntax knowledge is important for novice programmers to produce a successful code solution (Section 3.2.1). In this research study, PATs should assist IT learners with the learning of

Delphi programming language syntax, particularly as learners may not have access to Delphi beyond the classroom.

Feedback regarding errors (PS): PATs will benefit IT learners who struggle to identify syntax errors, by providing feedback regarding errors in the code or solution. Error feedback will assist learners to develop debugging skills.

Provides simple error messages (PS): If feedback is provided, simple error messages should be used that are easy to understand and help to identify the error in order to improve an IT learner's debugging skills.

Promotes problem solving and planning (PS): IT teachers identified problem solving and planning of solutions as skills that IT learners have difficulty understanding and applying. The PAT should help to promote these skills.

Feedback to guide solution creation (PS): IT learners need assistance to improve their ability to create algorithms. The PAT should assist learners to convert a planned solution to a code solution that can be executed.

In addition to the programming knowledge and skills criteria listed, the ranking of programming concepts on difficulty as a result of the feedback from IT teachers (Table 4.2) will be evaluated with difficult programming concepts identified from literature (Section 3.4), as well as programming concept results from IT learners (Section 4.3.2). The evaluation of programming concepts will result in a ranked list of programming concepts (descending order of difficulty) that PATs should be able to implement (Section 4.4).

4.3 IT Learner Feedback

IT learners participating in the research study as part of the Grade 11 control group were provided with the *Perceived Difficulty of Programming Questionnaire* (Appendix C) at the start of the first year of the study. The aim of the questionnaire is to identify factors contributing to the difficulty of IT programming from an IT learner perspective (Section 4.3.1).

The IT learners participating in the study are from the Grade 11 IT classes of four schools in the Port Elizabeth area. The participating learner responses from each school are combined for the results in this section. The sample size is n=45 learners.

4.3.1 Perceived Difficulty of IT

The *Perceived Difficulty of Programming Questionnaire* (Appendix C) was completed at the beginning of the Grade 11 year, thus learners were rating their perceptions of the difficulty of programming concepts and skills learnt in Grade 10. Learners are required to rate the difficulty of 22 different programming concepts and skills (Table 4.3) derived from the IT Learning Programme Guidelines (Department of Education, 2008). Items are rated on a 7-point semantic differential scale where 1 is "Extremely easy" and 7 is "Extremely difficult". Three of the programming concepts (accessing a database, SQL statements and objects and classes) have been omitted from Table 4.3 as IT learners had not yet covered these concepts at the time of administering the questionnaire. Table 4.3 indicates the mean value of the ratings provided by IT learners. None of the mean ratings were above the neutral value of four.

Table 4.3: Difficulty of programming concepts and skills as rated by Grade 11 learners (*Perceived Difficulty of Programming Questionnaire*) in descending order of frequency.

	Programming Concept/Skill	**Frequency** n=38	**Mean**	**SD**	n	**Teacher Mean**
Difficult	Procedures	13	3.65	1.40	37	4.5
	Debugging (skill)	5	3.47	1.70	43	5.0
	repeat-loops	4	3.11	1.79	44	4.5
	Planning (skill)	3	3.47	1.52	32	6.0
	Functions	2	3.40	1.38	30	4.5
	One-dimensional arrays	2	2.66	1.75	35	4.8
	for-loops	2	2.61	1.74	44	3.0
	String handling	2	2.44	1.45	45	5.0
	Two-dimensional arrays	1	3.35	1.50	17	5.3
	while-loops	1	2.93	1.71	45	4.5
	Problem solving (skill)	1	2.86	1.25	44	5.5
	Correct use of parameters	1	2.67	1.57	33	5.5
	File handling	1	2.42	1.35	24	3.5
	Algorithms (skill)	0	3.13	1.40	40	5.3
	case-statements	0	2.64	1.42	44	4.5
	if-statements	0	1.98	1.22	43	3.0
	Output	0	1.67	1.21	45	2.3
	Variables	0	1.59	1.19	44	3.3
	Input	0	1.56	1.22	45	2.5

Participants are also required explicitly to rank the concept that they perceive as the most difficult. The frequency count in Table 4.3 indicates the number of learners that specifically identified a concept as most difficult. *Procedures* was identified as the most difficult concept to understand by 34% of the learners (n=13) (Table 4.3). Learners also identified the programming skills debugging of code (n=4) and planning (n=3) as difficult. Procedures, functions, planning, debugging and algorithms were generally rated higher when compared to other items with mean values above 3. Although algorithms has the fifth highest mean value rating, none of the learners explicitly identified algorithms as difficult.

Two of the programming concepts, namely file handling and *for*-loops, categorised as difficult based on the results of IT learner feedback, were not identified as difficult concepts in the results from IT teachers (Table 4.2). *Case*-statements and algorithms were also identified as a difficult programming concept and skill, respectively, by IT teachers but not by IT learners.

Strings are implemented as an array of characters in Delphi (Kerman, 2002). IT learners apply the same knowledge to strings as to arrays, in order to read specific characters within a string. IT learners are also required to develop knowledge and an understanding of string handling procedures and functions. The use of string handling procedures and functions develops learner understanding of procedure and function calls as well as passing the correct parameter arguments. Even though string handling requires programming knowledge of one-dimensional arrays, procedures and functions, it is rated as one of the easier programming concepts by IT learners based on the mean rating, only more difficult than input, output, variables and *if*-statements (Table 4.3). This is in contradiction to the IT teacher feedback. IT teachers rated string handling as the fourth most difficult concept for learners to understand.

The remaining questionnaire items aim to determine the IT learner perceptions of their ability to understand simple code exercises and solutions, apply their understanding of simple solutions to more complex problems, debug errors in code and plan solutions to programming problems. Participants are required to rate their responses using a 7-point semantic differential scale where 1 is "Strongly disagree" and 7 is "Strongly agree".

Learners have a positive perception of their ability to understand simple examples (μ=6.22, s=0.82, n=45, $t(44)$=17.31, p<0.01) and apply knowledge to more complex examples (μ=5.39, s=1.06, n=45, $t(44)$=8.78, p<0.01), as a one-sample t-test indicates responses are statistically significant from the neutral value (4). IT learners' perceptions of their ability to understand simple exercises is statistically significantly higher ($t(44)$=5.93, p<0.01, d=0.76) than learners' perceived ability to apply programming concepts and understanding to more complex examples. Learners' positive perception of their ability to apply knowledge to more complex examples contradicts IT teacher feedback indicating that IT learners have difficulty applying programming knowledge to different problems and exercises (Section 4.2.2).

IT learner perceptions of their ability to debug errors in code were also evaluated. Two debugging items are related to the use of compiler error messages displayed by the programming development environment. A dependent samples t-test indicates that there is a statistically significant difference ($t(41)$=-2.27, p=0.029, d=0.31) between the use of compiler error messages (μ=5.31, s=1.87, n=43) and the understanding of compiler messages (μ=4.76, s=1.65, n=44). IT learners try to use the compiler error messages to find errors in code; however, learners may not always understand and correctly interpret the error indicated by the compiler error message.

The results for debugging also indicate that learners use output statements to display variable values at critical points in the program (μ=4.82, s=2.25, n=44). The mean rating is statistically significantly higher than the neutral value, 4 ($t(43)$=2.41, p=0.02). The use of breakpoints, a function supported by professional programming environments used to step through code during runtime and evaluate flow of execution and variable values manually, was given the lowest rating for the questions related to debugging (μ=3.67, s=2.21, n=43). Although the mean indicates breakpoints are not used, the result is not statistically significantly lower than the neutral value ($t(42)$=-0.97, p=0.34). The ability to trace the flow of code execution, either using breakpoints or output messages, is a skill that is important for creating a successful code solution (Section 2.3.1). Learners indicated that test input is used to check whether a solution is correct (μ=5.64, s=1.52, n=45, $t(44)$=7.23, p<0.01).

One of the factors identified as influencing a novice programmer's ability to program successfully (Section 3.2.1) is the ability to plan solutions (Fidge and Teague, 2009; Rongas *et al.*, 2004). The remaining items in the questionnaire to Grade 11 IT learners determine the extent to which IT learners plan solutions before programming. One sample t-tests are used to indicate responses that are statistically significant (p<0.05) from the neutral value (4).

Learners indicated that they are able to understand what is required for most problems or exercises (μ=5.69, s=1.18, $t(44)$=9.57, p<0.01). Learners also indicated confidence in their ability to produce a code solution for programming questions (μ=5.44, s=1.25, $t(44)$=7.73, p<0.01). Forty-seven percent of the learners (n=21) agreed with the statement that they work out some form of solution to the problem before coding, while 20% disagreed (μ=4.53, s=1.59, $t(44)$=2.25, p=0.029). The results for the last two items related to planning indicate that learners do not write non-code solutions (μ=2.95, s=1.55, $t(44)$=-4.52, p<0.01) nor do learners use comments to explain their code (μ=2.95, s=1.77, $t(44)$=-3.96, p<0.01).

4.3.2 Selection Criteria from IT Learner Feedback

Selection criteria for PATs, based on the feedback from IT learners, are presented. The selection criteria derived from the IT learner feedback address the difficulties associated with the understanding of *Programming Knowledge* (PK) and improvement of *Programming Skills* (PS).

Assists with understanding of code execution (PK): Assistance with code execution could allow learners to learn how to trace a program and evaluate the state of variables at different points during the execution of the code. These criteria would assist learners to improve debugging skills as well as algorithm and code comprehension.

Provides simple error messages (PS): Results indicated that learners' understanding of compiler error messages rated lower than their use of compiler error messages. Simple error messages would allow learners who use error messages to understand the error during debugging. Debugging is a skill learners identified as difficult.

Develops code comprehension (PS): Improving a learner's ability to read and understand code (Section 3.2.1) could help learners to evaluate their own code solutions and assist with debugging. Code comprehension would also assist learner understanding of common algorithms and how to adapt algorithms to different problems.

Promotes problem solving and planning (PS): Planning was identified as the third most difficult programming concept. Results indicated that learners do not create non-code solutions to assist with planning a solution. The PAT should thus assist learners to develop a program solution. This includes providing feedback during code execution to guide learners to identify any problems.

Feedback to guide solution creation (PS): The purpose of this criterion is to assist learners to convert a program solution (non-code) into a code solution, thus improving their ability to create code algorithms.

The programming concepts identified by IT learners will be evaluated together with the IT programming concepts (Section 4.2.2) and programming concepts identified as difficult to understand from literature, to form a set of selection criteria (Section 4.4).

4.4 Selection Criteria

The questionnaire responses from IT teachers and learners together with the results from the introductory programming literature study (Section 3.3) are used to formulate the selection criteria (Table 4.4) used to select PATs that can assist IT learners to understand programming concepts, improve programming knowledge and develop programming skills.

Table 4.4: Selection criteria for PATs

Category	**Criteria**	**Literature**	**Teacher**	**Learner**	**Weighting**
Concepts	Two-dimensional arrays	✓	✓	✓	18
	String handling	✓	✓	✓	17
	One-dimensional arrays	✓	✓	✓	16
	Procedures	✓	✓	✓	15
	Functions	✓	✓	✓	14
	repeat-loops	✓	✓	✓	13
	while-loops	✓	✓	✓	12
	Objects & classes	✓	✓		11
	for-loops	✓		✓	10
	if-statements	✓			9
	Correct use of parameters		✓	✓	8
	SQL statements		✓		7
	Accessing a database		✓		6
	case-statements		✓		5
	File handling			✓	4
	Variables				3
	Input (getting information from the user)				2
	Output (displaying information to the user)				1
Knowledge	Assists with the learning of the Delphi programming language syntax	✓	✓		5
	Assists with developing knowledge of programming principles & concepts	✓	✓		4
	Constructivist to promote self-study	✓	✓		3
	Assists with the application of programming knowledge	✓	✓		2
	Assists with the understanding of code execution	✓		✓	1
Skills	Promotes problem solving and planning	✓	✓	✓	5
	Provides simple error messages to assist with debugging	✓	✓	✓	4
	Develops code comprehension	✓		✓	3
	Feedback to guide solution creation		✓	✓	2
	Feedback regarding errors		✓		1

Programming concepts that are included in the IT subject framework for South African secondary schools (Department of Education, 2008) are included as programming concept criteria that PATs should address (Table 4.4). The programming concepts have been ranked in descending order of difficulty. The ordering of the selection criteria has been derived from considering common identification by literature, IT teachers and IT learners.

Selection criteria identified by literature are ranked higher than criteria that are not, as criteria identified in literature are supported by research studies. Programming concepts that are identified by the same groups, that is, literature, IT teachers and IT learners, are ranked based on the IT teacher ranking of programming concepts in order of difficulty (Table 4.2). Planning, problem solving, debugging and algorithms have been excluded from the list of programming concepts as they have been addressed by the selection criteria to improve programming skills (Table 4.4).

All of the programming concepts identified in the literature study, excepting for *if*-statements, are confirmed by IT teacher and/or IT learner results as difficult concepts. Correct use of parameters is the only programming concept identified as difficult by both IT teachers and learners, but not explicitly identified in the literature study (Section 3.3).

The programming knowledge criteria items are all derived from literature and are supported by the IT teacher results for all except one of the criteria items. The criteria that PATs should assist with the understanding of code execution is supported by the IT learner results. Programming skills criteria items to promote problem solving and planning as well as the use of simple error messages to assist with debugging are identified by literature and confirmed by the IT teacher and IT learner results (Table 4.4).

PATs selected for this research study should support the Delphi programming language. The criteria item to assist with the learning of programming language syntax explicitly requires PATs to support Delphi. A PAT is considered to address programming knowledge and programming skill selection criteria items if the PAT provides an explanation addressing the criteria explicitly and/or the criterion is implemented practically, to improve learner understanding and skills. A PAT is evaluated as meeting a programming concept selection criterion if the PAT provides assistance to support the understanding of the concept.

All the selection criteria are allocated a priority weighting based on the ranking of the criteria item within each category. The priority weighting is used to calculate a score for a PAT based on the number and priority of the selection criteria the PAT addresses. The first of the five programming knowledge and programming skills selection criteria items have a weighting of five and the last a weighting of one. The first programming concept criteria item has a weighting of 18 and the weighting is decremented by one for each criterion thereafter. No allowance has been made for equivalent weightings. Although Table 4.4 only indicates if concepts were identified as difficult by literature, IT teachers and IT learners, the mean ratings from IT teacher feedback have been used to rank “equivalent” criteria items. The experience of IT teachers has been considered an important factor in the decision to use the IT teacher rankings instead of the IT learner rankings.

4.5 Conclusion

The main purpose of this chapter is to formulate the selection criteria to identify PATs that are suitable for use by IT learners in South African secondary schools (Secondary Objective 1.1). This objective is achieved by addressing RQ1 and RQ2.

In addressing RQ1: *What programming difficulties and skills do PATs need to address and develop, respectively?* the questionnaire responses from IT teachers and learners have identified specific programming concepts that IT learners have difficulty understanding. The PATs should assist learners with the understanding of as many of the programming concepts included in the IT subject curriculum as possible with particular focus on the difficult concepts. Important skills with which IT learners have difficulty and that should be supported by PATs are problem solving, planning and debugging of errors. IT learners also need assistance to develop a code solution to a problem (algorithm).

The responses from IT teachers identify factors that make the learning of IT programming concepts difficult for IT learners and that may influence the use of PATs. The feedback is used to answer RQ2: *What factors may influence the use of PATs in SA secondary school learning environments?* PATs will benefit IT learners by assisting learners to improve programming knowledge, develop programming skills and improve understanding of programming concepts during self-study, due to the short class lessons with the teacher present. IT learners may also not have access to the Delphi programming environment out of class time, thus a PAT that assists learners to implement programming examples using code that is the same or similar to the Delphi programming language would be beneficial.

The IT teacher and learner feedback have been combined with the criteria identified from the literature study discussed in Chapter 3 to formulate selection criteria for PATs to support IT learners (Table 9.1). The selection criteria are grouped into three categories:

Programming concepts assisting learner understanding and use of specific programming concepts such as *if*-statements and loops,

Programming knowledge assisting learners to improve their knowledge of programming principles, execution of code and programming language syntax

Programming skills to develop IT learner programming skills such as debugging, problem solving and planning.

The selection criteria within the programming knowledge and programming skills categories are ranked in descending order of priority, and the programming concept criteria are ranked in descending order of difficulty, making identification of suitable PATs easier. The identification and formulation of the selection criteria has achieved Secondary Objective 1.1: *Formulate selection criteria for determining the suitability of PATs to support the achievement of programming learning outcomes in the IT subject curriculum.*

The selection criteria can be used to evalute PATs suitable for a specific category, namely, programming knowledge, programming skills or programming concepts or to identify PATs taht satisfy all three categories. PATs are evaluated by identifiying if a specific criterion is met by the PAT or not. A more detailed evaluation of a PAT would evaluate how a particular criterion item is met. For example, whether the PAT provides minimal support for a criterion item or whether the criterion is explicitly addressed by the PAT. The PAT can be scored based on the number of criteria met and the priority fo the criteria items using the weightings (Table 4.4). The PAT score is calculated as:

$$Score_{PAT} = \sum c_i w_i$$

where c_i=1 if criterion item i is met, otherwise c_i=0 and w_i is the priority weighting assigned to criteria item, i. PATs can receive four scores, one for each category and one overall score combining the scores of the three categories.

Chapter 5 identifies several PATs developed to assist novice programmers. The selection criteria (Table 9.1) are used to select PATs that are suitable to improve South African IT learner understanding of programming concepts.

Chapter 5

Programming Assistance Tool Selection

5.1 Introduction

The difficulties faced by novice programmers with regard to programming can be attributed to various ffactors (Section 3.3). Research has been done to identify these factors and to suggest ways of overcoming the difficulties. Research studies tend to focus on one difficulty and suggest a method, either tool or technique, for overcoming that particular difficulty.

The selection criteria have been formulated to select suitable PATs for IT learners (Section 4.4). The evaluation of PATs using the selection criteria can rank the suitability of PATs based on the priority and number of selection criteria that the PAT addresses. Existing PATs need to be identified in order to answer RQ3: *What PATs exist that would be suitable for use in SA secondary schools?*. The result of this chapter is thus the achievement of Primary Objective 1: *To identify existing introductory programming assistance tools (PATs) that can be used to support learner understanding of programming concepts included in the IT subject curriculum implemented in a case study of South African secondary schools.*

The techniques used by PATs to support novice programmer understanding of programming concepts, together with IT learner feedback regarding learning preferences, are presented (Section 5.2), before identifying specific PATs from different research studies in the field of novice programming (Section 5.3). The formulated selection criteria (Section 4.4) are used to determine the suitability of each of the PATs identified (Section 5.4). The chapter concludes by identifying three PATs suitable for use in South African secondary schools that are used to address Primary Objective 2: *To evaluate the impact of the selected programming assistance tools (PATs) on a novice programmer's understanding of* *programming concepts included in the IT subject curriculum implemented in a case study of South African secondary schools.*

5.2 Techniques Used by PATs

The purpose of a PAT is to assist novice programmer users to develop their programming skills and knowledge (Section 5.2.1). PATs use different techniques, such as animation or graphical representations, to support user understanding of different programming knowledge and skills (Section 5.2.2). The VARK questionnaire (Section 2.5.1) has been administered to IT learners to determine the learning preferences that should be addressed by PATs (Section 5.2.3).

5.2.1 Definition of a PAT

PATs are specifically designed for use by novice programmers. PATs can use visualisation techniques to make the programming environment far simpler to use than professional programming environments. PATs can also visualise programming concepts in order to support novice programming understanding of abstract concepts. Interactive microworlds can make programming more interesting and applicable (Pears *et al.*, 2007).

Rongas *et al.* (2004) states that the ideal PAT would be able to support:

- problem solving,
- algorithm design,
- data structure design,
- assist with the learning of syntax for a particular programming language,
- partial compiling for fast checking of output and operation of a code block,
- administrative properties to support the teacher, and
- communication properties to support team work.

A combination of several systems would fulfill all requirements but would be too complicated to use (Rongas *et al.*, 2004).

An advantage of using a PAT is that it can assist a novice programmer to develop an understanding of programming concepts as well as provide automated assessment of programming tasks and correction of simple errors (Rongas *et al.*, 2004). Other advantages are that PATs can promote interactivity as well as support self-study, providing novice programmers with different pedagogical learning methods.

PATs can employ visualisation techniques to enhance algorithm comprehension, improve debugging ability and guide exploration to understand concepts (Pears *et al.*, 2007). The structure or execution of code can be visualised or a programming concept can be animated. There are PATs that have environments that allow novice programmers to construct algorithms and visualisations graphically, using flowcharts or drag-and-drop techniques (Kelleher and Pausch, 2005) and thus prevent novice users from making syntax errors. Structured editing, where elements are specified from a menu when typing the program, can also be used (Guzdial, 2004) where placeholders indicate to users where additional code should be specified, such as variable names or values. Structured editing assists users by providing valid constructs, thus allowing users to develop an executable solution, although logic errors may exist.

Microworlds can be used to make the understanding of programming more concrete as users can view the result of programming solutions as actions by characters or objects in a visual and/or animated environment (Kelleher and Pausch, 2005). General-purpose, professional programming languages can be abstract and difficult to understand. Microworld environments motivate users to want to learn as the environment is fun.

PATs use animation and/or visualisation in different ways. A solution can be created by typing code while the resulting program can be animated, such as a robot performing actions in a 3D world. Alternatively, graphical objects can be used to create the program solution even though the running program may not have any animation nor use a graphical user interface.

A PAT that uses visualisation techniques may not assist novice programmers. In some cases novice programmers that use a PAT do not perform better than those not using it (Levy *et al.*, 2001). The visualisation techniques used by a PAT may not be successful when the learners are not able to map programming concepts to elements in the animation. PATs may also use animations that still require explanations from the teacher to be successful. However, learners using a PAT with animation show increased satisfaction and motivation compared to learners who do not use a PAT.

5.2.2 Evaluation of Techniques used by PATs

It is important when evaluating the impact of the PATs on IT learner motivation towards programming and on IT learner understanding of programming concepts, to evaluate the different methods the learner uses in the PAT to create the program solution and the resulting "program". Kelleher and Pausch (2005) use several items to evaluate PATs with regards to the representation of code, construction of programs, support to understand programs and prevention of syntax errors.

Table 5.1: Criteria to evaluate the techniques used by PATs. † indicates criteria originating from Kelleher and Pausch (2005).

Criteria	**Description**
Representation of the program solution	
Text†	The program solution is programming code represented as text.
Flowchart†	The user creates a flowchart of the program solution.
Graphics	Graphical objects (such as building blocks) or pictures are combined to create the program solution.
Method of constructing programs	
Typing of code†	Program solution has a textual representation.
Assembling or positioning graphical objects†	Graphical objects are connected and/or positioned in relation to other graphical objects.
Selecting/form filling†	The user can select objects or text from a list provided. Variable names and values are filled into placeholders provided.
Format of resulting program	
Text (input & output)	A command-line interface type program is created.
Graphical User Interface (GUI)	A form window (similar to a print dialog box) created for user interaction.
Animation	The resulting program, game or video is animated.
Microworld	A microworld may include animation, but is restricted to a specific character (for example a robot or turtle).
Support to understand programs	
Debugging†	The program includes methods to check for errors in the solution
Animation of program execution	Animation used to demonstrate creation of variables, assigning of values, etc.
Testing user knowledge (questions)	Program assesses user knowledge of programming concepts and code execution while the program is running.
Preventing syntax errors	
Selection from valid options†	Only valid options are provided to users.
Dropping only in valid locations†	The solution can only be created in a way that will prevent errors.
Informative syntax error messages†	Syntax errors provided are explain the error and how to correct.

Relevant categories used by Kelleher and Pausch (2005), namely *representation of code*, *construction of programs*, *support to understand programs* and *preventing syntax errors* are included to define criteria (Table 5.1) to evaluate the techniques used by PATs identified by this research (Section 5.4).

The category, *format of resulting program*, has been included to evaluate the techniques used to represent the resulting program when the solution is executed. Within each category, the technique criteria have been adapted after researcher interaction with the different PATs identified (Section 5.3) resulting in certain of the criteria used by Kelleher and Pausch (2005) to be omitted and/or the addition of criteria items not used by Kelleher and Pausch (2005).

The representation of code category indicates techniques used to represent the code solution created by programmers (Kelleher and Pausch, 2005). Techniques include a text program solution in a specific programming language, flowcharts as well as the use of graphical objects connected together in some way to indicate flow of execution. Program solutions can be constructed by typing the code or pseudocode in the case of text representations. Solutions can also be created by dragging or selecting appropriate programming concepts from a list of options.

Once a program solution is created, the program is executed. An evaluation of the PATs has shown that PATs use different techniques to display the program output. Certain PATs execute the program solution as a console application that accepts text input and displays text output. Graphical user interfaces can be used to allow users to interact with the executing program. PATs can also execute the program solution using visualisation techniques such as animations and microworlds.

Techniques provided to support user understanding of programming concepts and knowledge include visualisation techniques that animate the execution of the program solution and the identification of errors in the program solution, to assist with debugging. Certain of the PATs evaluated assess user knowledge of programming and understanding of the execution of the programming solution. PATs can also employ techniques to prevent syntax errors in code by allowing users to select only from valid options and/or place selected options in valid locations. Informative messages allow users to identify syntax errors in the program solution.

5.2.3 Learning Preferences of IT learners

Grade 10 and Grade 11 learners in the control group of this study (n=105) completed the Visual, Aural, Read/Write and Kinesthetic (VARK) Questionnaire (Fleming and Baume,

2006) to determine their learning preferences (Chapter 2.3.1). The purpose of administering the VARK questionnaire to participating learners is to determine the learning preferences of IT learners that PATs should address (RQ2). The results of the VARK Questionnaire are relevant for the selection of PATs for use by IT learners thus the questionnaire has only been administered to control group learners.

The VARK questionnaire identifies four learning preferences (Section 2.5.1), namely, *visual* (use of graphics or drawings to explain a concept), *aural* (to hear an oral explanation of a concept by teacher or speak-out-loud repetition by learner), *read/write* (reading or writing out the explanation of a concept) and *kinaesthetic* (to see a demonstration of or to perform a task related to the concept being learnt).

The learning preference(s) of the IT learners were determined by totalling the number of responses for each of the four learning preferences and then determining which of the learning preferences is dominant (Fleming and Bonwell, 1997). Fifteen learning preference categories exist (Slater, Lujan and DiCarlo, 2007), indicating a combination of learning preferences (multimodal) or a single preferred learning preference (unimodal). Multimodal can further be classified as bimodal (two learning preferences are preferred), trimodal (three learning preferences are preferred) or quadmodal (all four learning preferences are preferred).

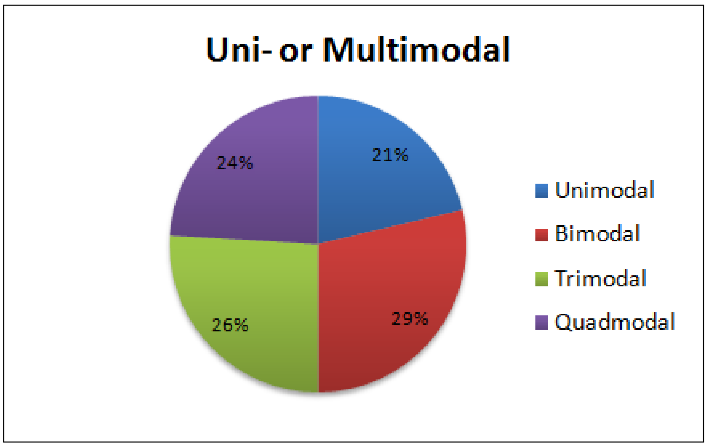


Figure 5.1: Distribution of IT learner learning preferences as multimodal (bi-, tri- or quadmodal) or unimodal

The majority of learners (79%, n=85) are multimodal (Figure 5.1). Overall, learners have a slight preference for the kinaesthetic learning preference (Figure 5.2). Figure 5.3 indicates that more unimodal learners prefer the kinaesthetic learning preference (48%, n=11), while only one (4%) of the unimodal learners prefers the visual learning preference.

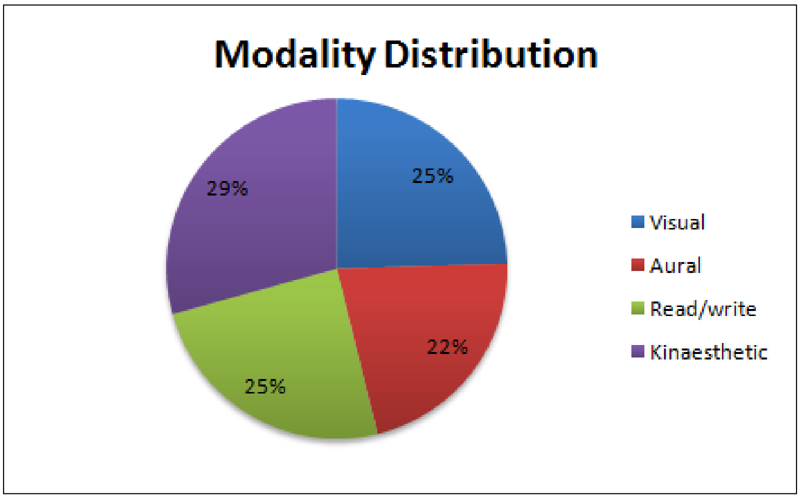


Figure 5.2: Distribution of learning preferences

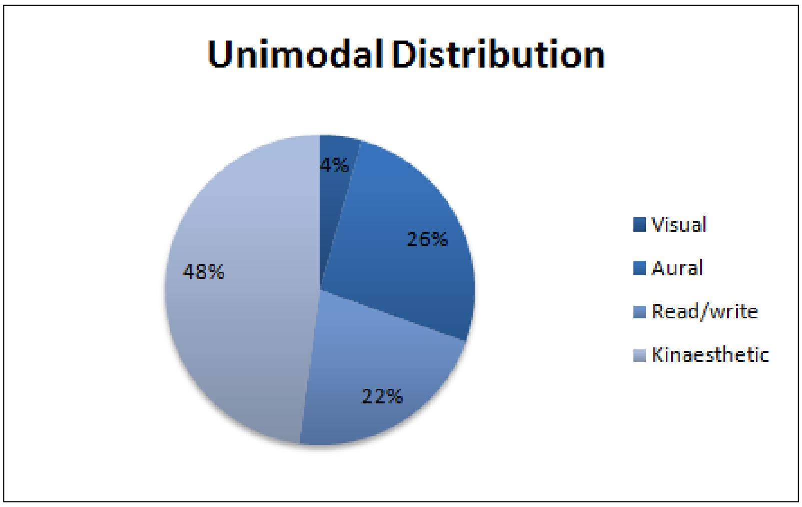


Figure 5.3: Distribution of unimodal learning preferences

Figure 5.4 shows the categorisation of multimodal learning preferences. Thirty-one percent (n=26) of learners are quadmodal (all four learning preferences are preferred). The majority of multimodal learners (65%, n=55) have visual and kinaesthetic as two of their preferred learning preferences.

The VARK questionnaire results indicate that PATs catering for visual and kinaesthetic learning preferences would fulfill the learning preferences of the majority of learners (54%). However, unimodal learners prefer either the kinaesthetic, read/write or aural learning preferences, with only one learner preferring the visual learning preference.

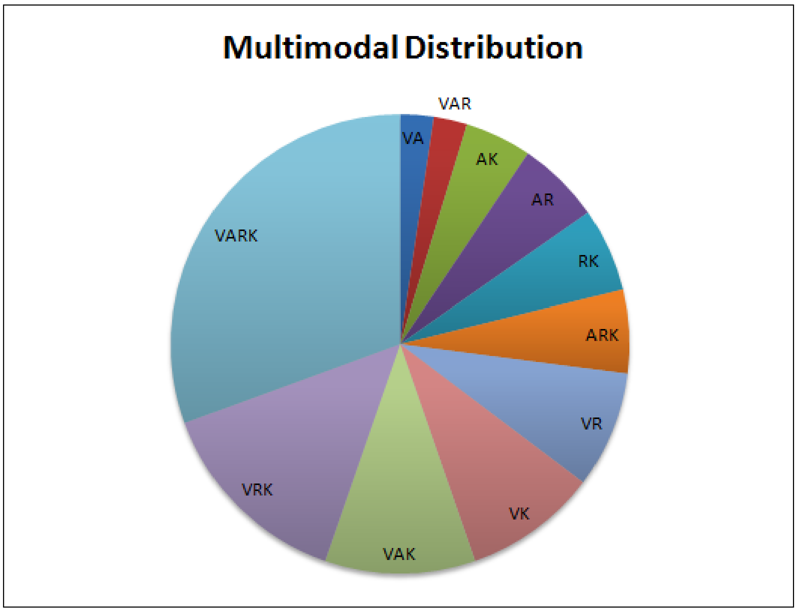


Figure 5.4: Distribution of multimodal learning preferences (V=Visual, A=Aural, R=Read/Write, K=Kinaesthetic)

5.3 Programming Assistance Tools for IT

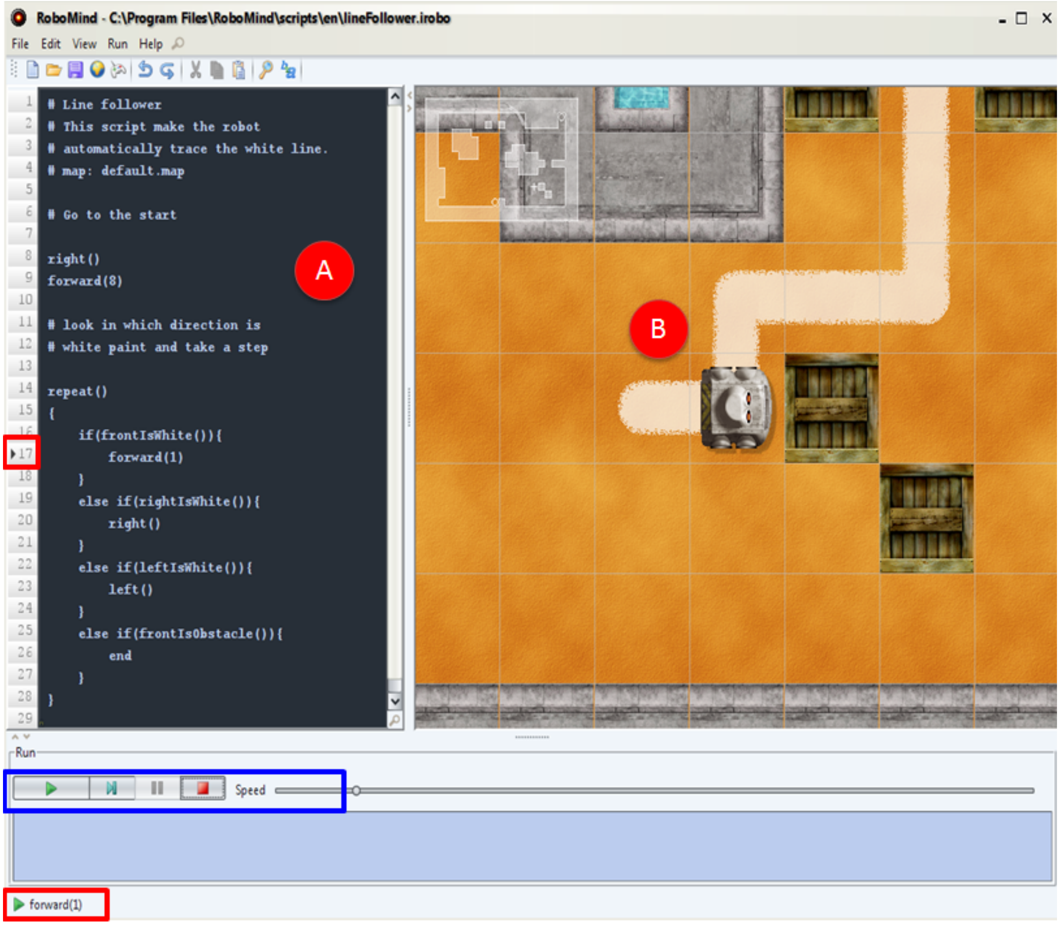
The PATs selected for review and evaluation have been identified based on availability. PATs identified had to be freely available for download in order to make the PAT available to IT learners (Section 1.5). The PATs reviewed are a representative sample of tools developed to support novice programmers.

PATs are evaluated using the selection criteria (Section 4.4). Only PATs that have met most of the criteria items are included in this section for further discussion, for the sake of brevity. PATs that were evaluated but which have been omitted as they did not address a majority of the selection criteria include *MatrixPro*[1], *KarelRobot*[2], *NoteTab*[3], *NotePad2*[4], *Phrogram*[5], *JHave*[6], *Squeak*[7], *EToys*[8] and *GameMaker*[9]. PATs that have been evaluated and included in this section for discussion include: *RoboMind* (Section 5.3.1), *BlueJ* (Section 5.3.2), *Greenfoot* (Section 5.3.3), *Scratch* (Section 5.3.4), *B#* (Section 5.3.5), *Jeliot* (Section 5.3.6), *Ville* (Section 5.3.7), *PlanAni* (Section 5.3.8), *Alice* (Section 5.3.9), and *jGRASP* (Section 5.3.10).

[1] http://www.cse.hut.fi/en/research/SVG/MatrixPro
[2] http://karel.sourceforge.net
[3] http://www.notetab.com
[4] http://www.flos-freeware.ch/notepad2.html
[5] http://phrogram.com
[6] http://jhave.org
[7] http://squeak.org
[8] http://www.squeakland.org
[9] http://www.yoyogames.com/make

5.3.1 RoboMind

RoboMind[10] has been designed as a tool that can be used as a first introduction to automation and programming without any prerequisites. A simple text-based educational programming language called ROBO is used to program a robot (on the screen) to interact with objects in a world specified by a map (Figure 5.5). *RoboMind* is suitable for use by primary education learners through to university students as the difficulty level can be adapted to the user's preference.



RoboMind - C:\Program Files\RoboMind\scripts\en\lineFollower.irobo
File Edit View Run Help
A
B
Run
Speed
forward(1)

```
# Line follower
# This script make the robot
# automatically trace the white line.
# map: default.map

# Go to the start

right()
forward(8)

# look in which direction is
# white paint and take a step

repeat()
{
    if(frontIsWhite()){
        forward(1)
    }
    else if(rightIsWhite()){
        right()
    }
    else if(leftIsWhite()){
        left()
    }
    else if(frontIsObstacle()){
        end
    }
}
```

Figure 5.5: Robot executing code to follow the white line in RoboMind

RoboMind provides users with a set of commands to move the robot through the world. Commands include movement commands such as `forward(n)` where `n` specifies the number of blocks to move, turning commands such as `right()`and "looking" commands which requires the robot to provide feedback about the space around it, for example, `frontIsObstacle()` returns true if there is a wall, box or other obstacle directly in front of the robot. The robot is also able to detect, pick up and drop beacons, paint blocks white or black and flip a coin to make a decision (randomisation).

[10] `http://www.robomind.net/en/index.html`

RoboMind implements basic programming concepts such as looping and nested *if*-statements. Procedures with parameters are also acceptable. No variables can be declared or used, except for the procedure parameters. The language syntax used is similar to Java although there is some variation such as the lack of semi-colons.

The user types out the code instructions to navigate the robot through the world (A in Figure 5.5) and then selects *Run* to execute the instructions. The tool will notify the user of any errors in the code. If the code is error-free, the user can watch the robot navigate through its environment based on the code instructions provided (B in Figure 5.5). The current line of code being executed is indicated by a pointer and a message at the bottom of the screen (red boxes in Figure 5.5). Users can stop or pause the execution of the program (blue box in Figure 5.5) as well as change the execution speed.

The *RoboMind* environment is freely available for individual, educational and commercial use (RoboMind, 2009). The *RoboMind 2.2* development environment is available as open source, allowing *RoboMind* to be adapted.

5.3.2 BlueJ

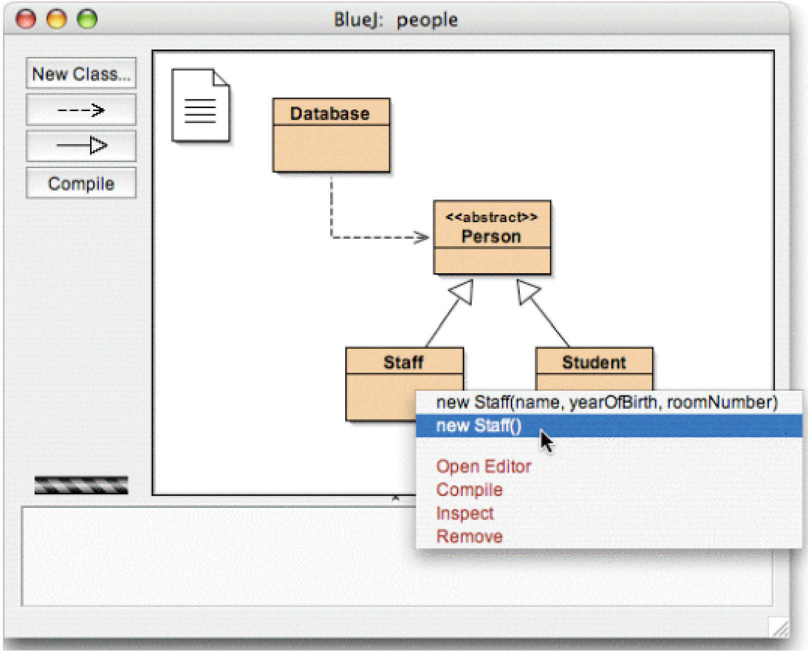


Figure 5.6: BlueJ Main Window showing interactive class creation (Kölling, 2004)

BlueJ[11] is a tool that can be used to introduce novice programmers to the concept of objects and classes using an objects first approach with the Java programming language.

[11] http://www.bluej.org

It is difficult to teach the concept of objects to novice programmers who have very little programming knowledge. If professional programming IDEs, such as Delphi or Netbeans/Eclipse for Java, are used to teach objects and classes, learners would need to understand many different concepts and complicated syntax before anything could be implemented (Kölling and Rosenberg, 2002). *BlueJ* can be used to assist with the understanding of objects and classes as it demonstrates the concepts without the learner having to write any code (Kölling and Rosenberg, 2001). Classes are created interactively using graphical objects and the associated Java code is generated automatically.

The advantages of *BlueJ* are that it is simple to use, interactive and uses visualisation to help novice programmers understand objects and classes. UML-like class diagrams provide a graphical overview of the project structure (Figure 5.6). Other features include interactive class creation and the ability to invoke interactively public methods (Kölling and Rosenberg, 2002). A disadvantage is that, although the interface is interactive and simple to use, teachers would have to design exercises based on the functionality provided by *BlueJ*, for example, exercises to create objects and classes using the graphical class diagrams and visualisations.

5.3.3 Greenfoot

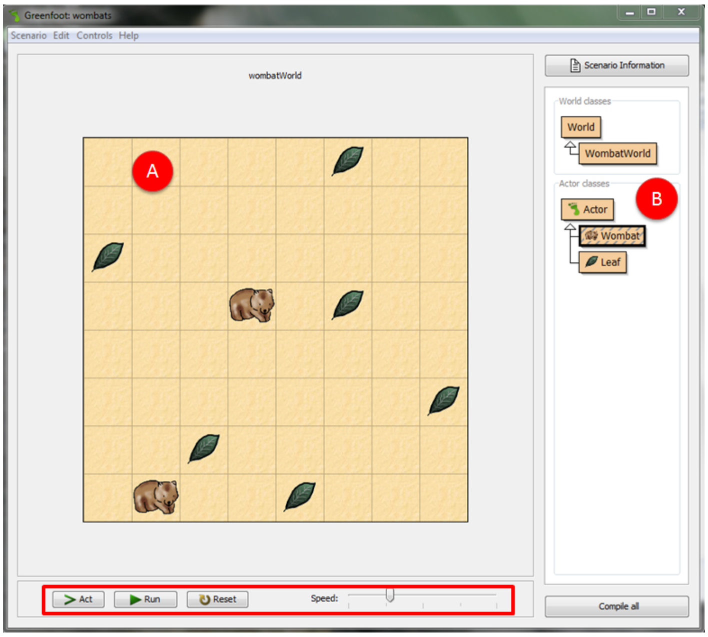
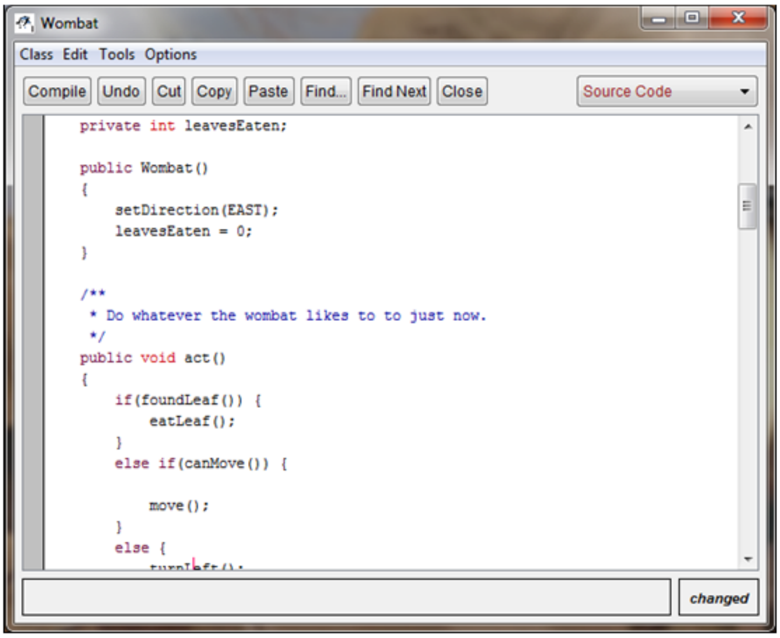


Figure 5.7: Greenfoot main window

Greenfoot[12] is a tool that can be used to teach object-oriented programming to novice programmers (Henrikson and Kölling, 2004). The framework provided by *Greenfoot* al-

[12] http://www.greenfoot.org

lows users to create easily different microworlds that are visually appealing and easy to interact with.



```
private int leavesEaten;

public Wombat()
{
    setDirection(EAST);
    leavesEaten = 0;
}

/**
 * Do whatever the wombat likes to to just now.
 */
public void act()
{
    if(foundLeaf()) {
        eatLeaf();
    }
    else if(canMove()) {

        move();
    }
    else {
        turnLeft();
```

Figure 5.8: Greenfoot code window for the wombat object

In *Greenfoot* all objects have a graphical representation and a position in the world, for example, the bear and leaf objects in the initial *Greenfoot* exercise (A in Figure 5.7). Users can interact with these objects directly and changes in the position and appearance of objects can be observed directly. The world itself (the background area behind *Greenfoot* objects - represented as a grid (A in Figure 5.7)) is also an interactive, programmable object. Classes associated with *Greenfoot* objects are displayed to the right of the world (B in Figure 5.7). Controls to run, stop, single-step or control the speed of simulations (red box in Figure 5.7) are provided for users to control program execution.

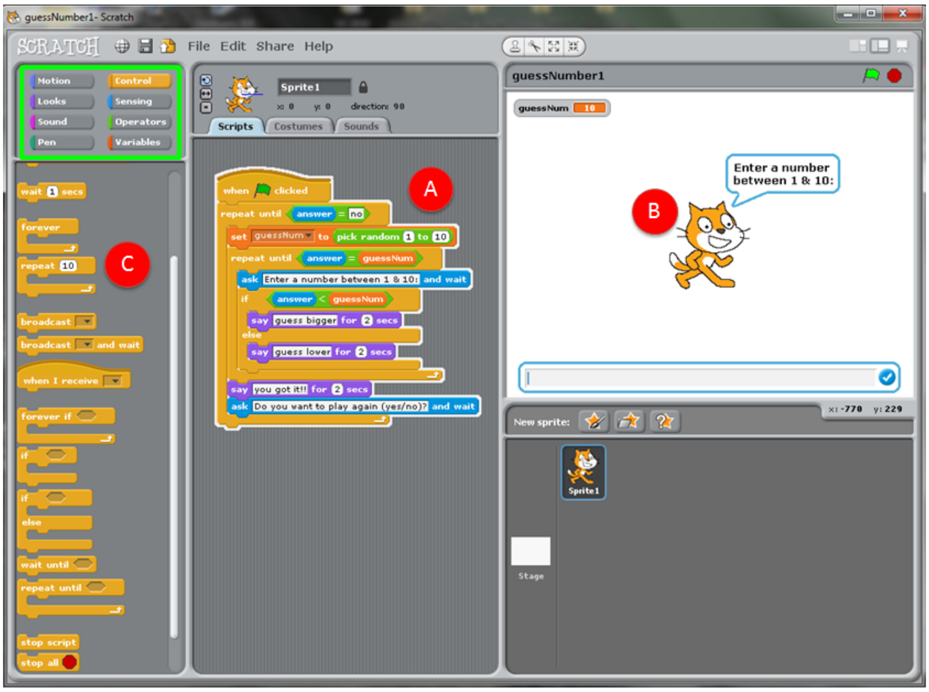
All object code can be modified to alter behaviours of objects in the world such as the code generated for a wombat object (Figure 5.8). The Java code associated with the objects is generated automatically. *Greenfoot* is designed, specifically, to improve novice programmer understanding of objects and classes using visual representations of objects.

5.3.4 Scratch

Scratch[13] was developed as an approach to programming that would allow children to start programming earlier (Utting, Cooper, Kölling, Maloney and Resnick, 2010). The

[13] http://scratch.mit.edu/

primary goal of *Scratch* is not to prepare users for careers as professional programmers but it allows people of different backgrounds and interests, who lack previous programming experience, to create easily their own interactive stories, games, animations and simulations (Resnick, Maloney, Monroy-Hernández, Rusk, Eastmond, Brennan, Millner, Rosenbaum, Silver, Silverman and Kafai, 2009; Utting *et al.*, 2010).



```
when ⚑ clicked
repeat until answer = no
  set guessNum to pick random 1 to 10
  repeat until answer = guessNum
    ask Enter a number between 1 & 10: and wait
    if answer < guessNum
      say guess bigger for 2 secs
    else
      say guess lover for 2 secs
  say you got it!! for 2 secs
  ask Do you want to play again (yes/no)? and wait
```

Figure 5.9: Creating a script in Scratch by dragging building blocks

Scratch uses a building block metaphor that allows users to build scripts by combining graphical blocks similar to building a jigsaw puzzle (A in Figure 5.9). This approach eliminates syntax errors as it allows the novice programmer to focus on finding the solution to problems (Maloney, Burd, Kafai, Rusk, Silverman and Resnick, 2004). The different blocks are designed in such a way that users are able to play around with the sequence and combinations of blocks. Blocks are also designed, specifically, to assist users by using shapes and colours to indicate different concepts. For example, control structures such as *repeat*-loop blocks are yellow and C-shaped to indicate to users that other blocks should be placed inside (A in Figure 5.9).

Similar programming construct blocks are grouped together in colour-coded categories (green box in Figure 5.9), for example, motion, sound and variables. Selecting a category lists all the associated blocks (C in Figure 5.9), thus allowing users to drag required blocks to the work area (A in Figure 5.9) to build a script. The program is executed in the window on the top right (B in Figure 5.9). The script blocks are highlighted as the program executes to indicate flow of code execution. In addition, users can set an option to execute blocks one step at a time (single-step), making it easier for users to follow the execution of code and associate code blocks with actions of the graphical objects and characters.

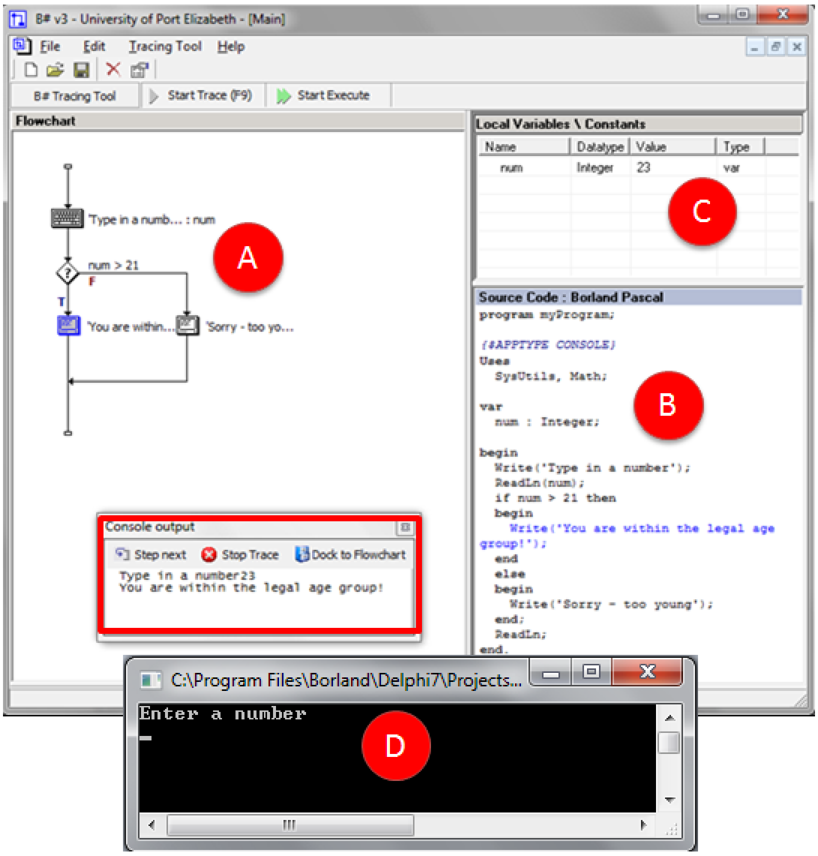
The advantages of using *Scratch* are that it is visually appealing and promotes active learning. Research by Malan and Leitner (2007) showed a decrease in the number of students dropping or failing an introduction to computer science course when *Scratch* was used to introduce programming concepts to the learners. A drawback of using *Scratch* is that users may struggle to move to a traditional programming environment without an intermediate software tool to provide a link between the programming concepts introduced in *Scratch* and the methods of implementing these concepts in a programming language (Resnick *et al.*, 2009) where syntax becomes relevant.

5.3.5 B#

B# (Greyling, Cilliers and Calitz, 2006) is an iconic programming environment designed to simplify programming tasks by assisting novice programmers with problem-solving strategies and the design of algorithms. *B#* uses a visual flowchart approach that supports programming concepts such as assigning values to variables, conditions, loops, inputs and outputs (Greyling *et al.*, 2006). Version 3 of *B#* is evaluated with regard to the selection criteria in Section 5.4.

Figure 5.10 is a snapshot of the *B#* user interface currently in the execution state. The iconic flowchart representation of the program running is depicted (A Figure 5.10). The associated code generated by the system is also displayed (B in Figure 5.10) allowing users to compare the flowchart created with the code generated. The code generated by *B#* is Object Pascal (Delphi) code; however, no graphical user interface components are used. Input and output is received using `readln` and displayed using `write/writeln`, respectively. The resulting program is a command line console program (D in Figure 5.10) that accepts text input and displays text output.

Functionality is provided to control execution of the program, such as stepping through the code one line at a time or stopping the execution at a particular point (red box in Figure 5.10). The current values assigned to variables are visible at all times during the execution of the code (C in Figure 5.10).



Name	Datatype	Value	Type
num	Integer	23	var

```
program myProgram;

{$APPTYPE CONSOLE}
Uses
  SysUtils, Math;

var
  num : Integer;

begin
  Write('Type in a number');
  ReadLn(num);
  if num > 21 then
  begin
    Write('You are within the legal age
group!');
  end
  else
  begin
    Write('Sorry - too young');
  end;
  ReadLn;
end.
```

```
Type in a number23
You are within the legal age group!
```

```
Enter a number
_
```

Figure 5.10: Stepping through code created using a flowchart in *B#*

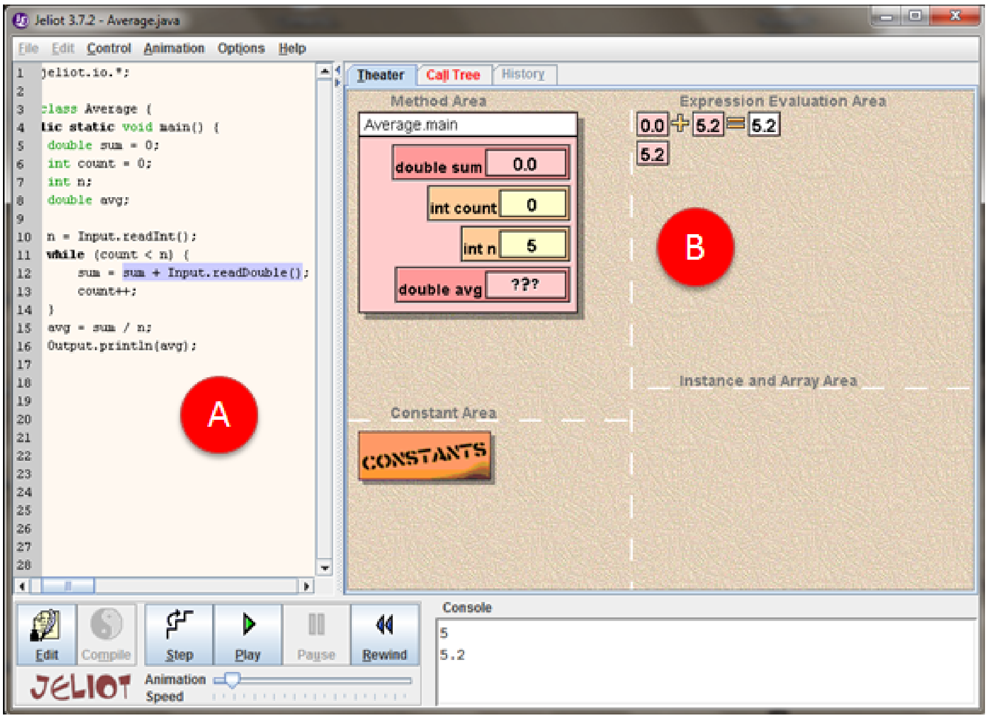
5.3.6 Jeliot

Jeliot[14] is a tool that is capable of animating programs to assist novice programmer understanding of introductory programming concepts. *Jeliot* supports the Java programming language. The current version, *Jeliot 3*, is capable of animating object allocation (Moreno, Myller and Bednarik, 2005).

Jeliot uses program visualisation to assist novice programmers to develop an accurate mental model during program execution (Moreno *et al.*, 2005). Four areas are used in the animation to indicate the current value of variables, the evaluation of expressions, the value of constants and the allocation of and reference to objects and arrays (B in Figure 5.11). The current line of execution in the code solution (A in Figure 5.11) is visible when the program is executing.

The standard *Jeliot* program provides animations to assist novice programmers to understand Java programs (Moreno *et al.*, 2005). However, *Jeliot 3* was redesigned to separate the interpretation and animation of the Java programs. *MCode*, a textual representation of a running program, is used to connect the interpretation and animation. *MCode* is language independent, thus, a Delphi interpreter, for example, could produce *MCode* for a

[14] http://cs.joensuu.fi/jeliot/



```
jeliot.io.*;

class Average {
lic static void main() {
 double sum = 0;
 int count = 0;
 int n;
 double avg;

 n = Input.readInt();
 while (count < n) {
       sum = sum + Input.readDouble();
       count++;
 }
 avg = sum / n;
 Output.println(avg);
```

```
5
5.2
```

Figure 5.11: Use of animation to explain program execution in Jeliot

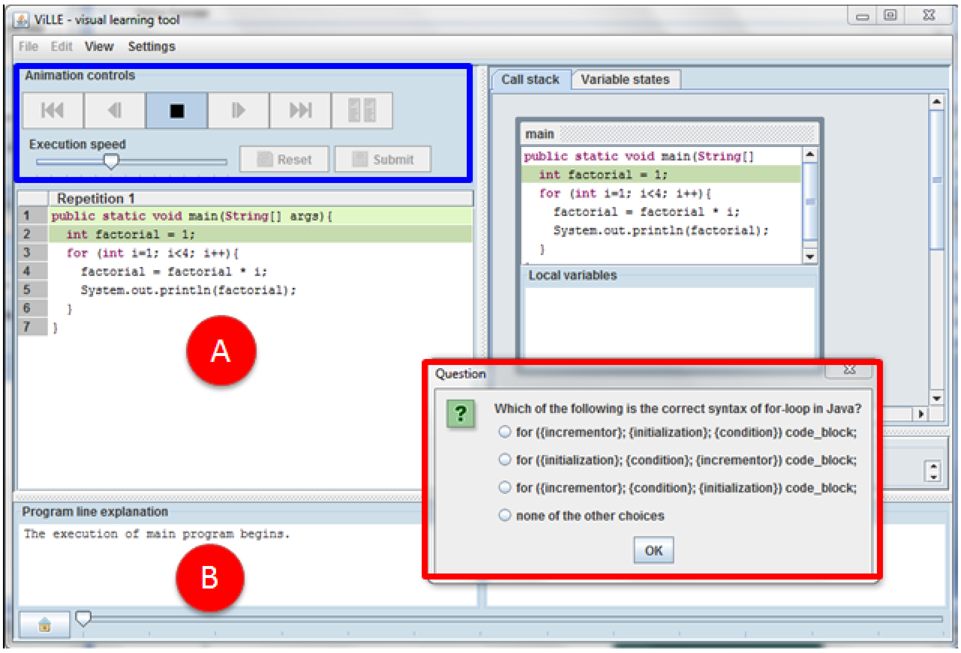
program written in Delphi and send it to *Jeliot 3* which would use the *MCode* to generate an animation for the program. In this way, *Jeliot* can be adapted for other programming languages, if an interpreter is available.

5.3.7 Ville

Ville[15] is a language-independent programming tool (Rajala, Laakso, Kailo and Salakoski, 2007). *Ville* has a built-in syntax editor that can be used to add new programming languages to the tool or to modify the syntax of the built-in languages. Visualisation is used to demonstrate code execution.

Ville provides a library of example programs to explain different programming concepts. The user (or teacher) is also able to add exercises. *Ville* allows the user to run a program (A in Figure 5.12) with control for the speed of execution, to stop execution or step forward and backward through the code (blue box in Figure 5.12). Visualisation is used to indicate to the user which line of code is currently being executed. An explanation for program lines is also provided (B in Figure 5.12). *Ville* can also be set up to ask the user questions about the current code being executed thus testing user knowledge of programming concepts and understanding of the code being executed (red box in Figure 5.12). The program line explanations and questions to test user knowledge would need to be set

[15] http://ville.cs.utu.fi/



```
public static void main(String[] args){
  int factorial = 1;
  for (int i=1; i<4; i++){
    factorial = factorial * i;
    System.out.println(factorial);
  }
}
```

```
public static void main(String[]
  int factorial = 1;
  for (int i=1; i<4; i++){
    factorial = factorial * i;
    System.out.println(factorial);
  }
```

The execution of main program begins.

Which of the following is the correct syntax of for-loop in Java?

- ○ for ({incrementor}; {initialization}; {condition}) code_block;
- ○ for ({initialization}; {condition}; {incrementor}) code_block;
- ○ for ({incrementor}; {condition}; {initialization}) code_block;
- ○ none of the other choices

Figure 5.12: Execution of a program in Ville using multiple choice questions to assess programming knowledge

up by a teacher for any new exercises to assist learner understanding of the programming concepts.

Ville provides users with the functionality to compare code in two different programming languages. Built-in programming languages include Java, Python, PHP, javascript, C++ and pseudo code. Additional languages can be added by mapping statements in the new language to Java statements. However, there are shortcomings with this approach. For example, in Delphi no semi-colon should appear after the last line before an *else* statement, but there is no way to indicate this in the syntax editor.

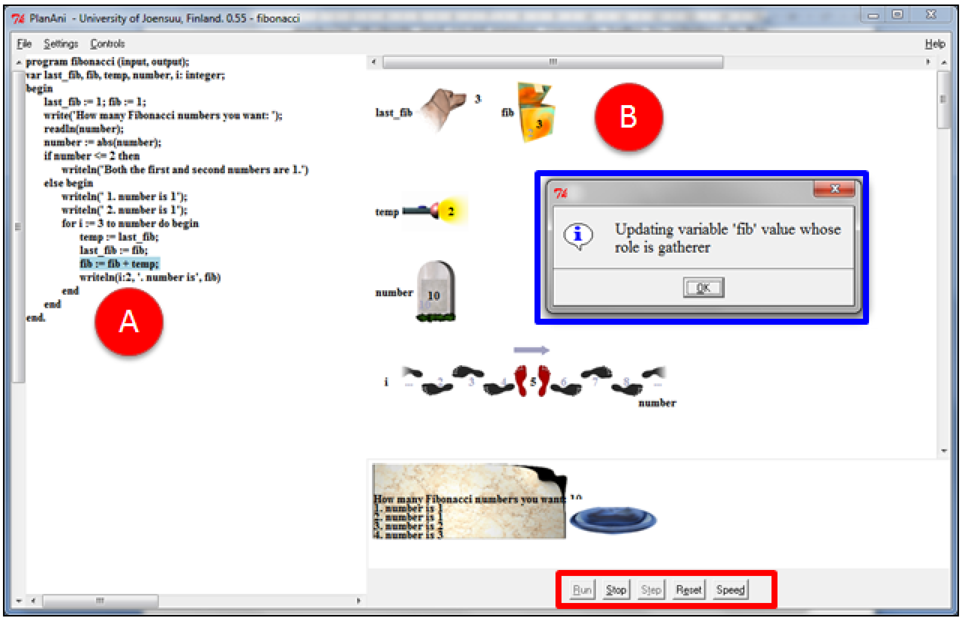
5.3.8 PlanAni

PlanAni[16] is a tool that uses visualisation to animate the roles of variables in a program (Byckling and Sajaniemi, 2006). In any program, each variable has a particular purpose or *role*. The role of a variable describes how the variable will behave within the program. *PlanAni* is a PAT specifically developed to enhance a novice programmer's understanding of the different roles of variables.

PlanAni allows novice programming users to select built-in code examples in one of four different programming languages: C, Java, Pascal or Python. Users can only view the execution of built-in examples (A in Figure 5.13). Novice programming users cannot

[16] http://cs.joensuu.fi/~saja/var_roles/planani/

create their own programs and the built-in example code cannot be edited in *PlanAni*. Java, as well as Pascal examples, are provided by the system. There are very few built-in examples, and *PlanAni* opens specific types of files - not normal code files. Teachers can create their own examples in this special file format that will not only provide the code for a particular solution but also include instructions on how each line should be animated and the messages that should be displayed to users to explain each line of code.



```
program fibonacci (input, output);
var last_fib, fib, temp, number, i: integer;
begin
    last_fib := 1; fib := 1;
    write('How many Fibonacci numbers you want: ');
    readln(number);
    number := abs(number);
    if number <= 2 then
        writeln('Both the first and second numbers are 1.')
    else begin
        writeln(' 1. number is 1');
        writeln(' 2. number is 1');
        for i := 3 to number do begin
            temp := last_fib;
            last_fib := fib;
            fib := fib + temp;
            writeln(i:2, '. number is', fib)
        end
    end
end.
```

Updating variable 'fib' value whose role is gatherer

OK

```
How many Fibonacci numbers you want: 10
1. number is 1
2. number is 1
3. number is 2
4. number is 3
```

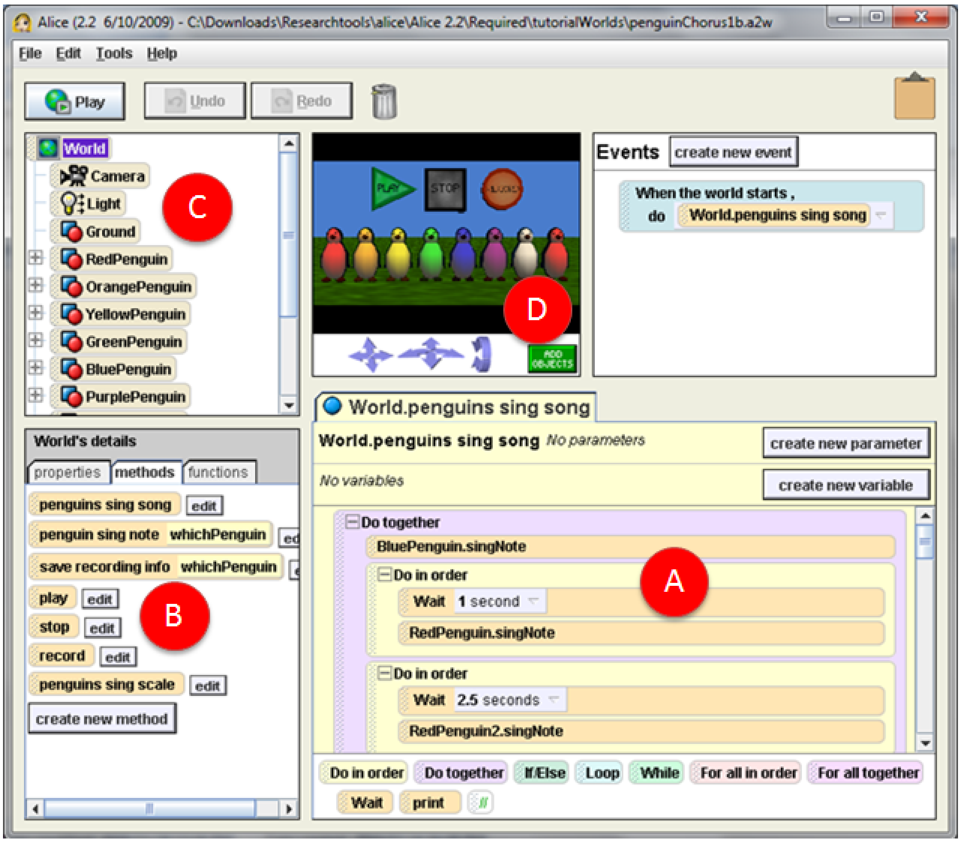
Figure 5.13: Animation of program code execution in PlanAni

Users can select to run the program which will start the animation. Each line of code is executed with messages (blue box in Figure 5.13) to explain to the user what task the line of code is performing or to describe the role of a particular variable and how its value is being assigned. Visualisation is used to indicate to the user what the current variable values are and how they are changed (D in Figure 5.13). Different graphical representations are used to depict the different variable roles, for example, footsteps represent a stepper variable, and a tombstone represents a fixed value variable. Functionality is provided to stop and step through the code one line at a time (red box in Figure 5.13)

5.3.9 Alice

Alice[17] is a 3D programming environment that can be used to teach introductory programming concepts using an "objects first" approach. *Alice* allows users to create 3D animations (D in Figure 5.14), games or videos (Henrikson and Kölling, 2004).

[17] http://www.alice.org



Alice (2.2 6/10/2009) - C:\Downloads\Researchtools\alice\Alice 2.2\Required\tutorialWorlds\penguinChorus1b.a2w

When the world starts , do World.penguins sing song

World.penguins sing song *No parameters*

No variables

```
Do together
    BluePenguin.singNote
    Do in order
        Wait 1 second
        RedPenguin.singNote
    Do in order
        Wait 2.5 seconds
        RedPenguin2.singNote
```

Figure 5.14: Program creation in Alice

Figure 5.14 is a screenshot of *Alice* that shows the environment used to create animations by assembling objects (C in Figure 5.14) and instructions (A in Figure 5.14) using drag and drop actions to generate the program solution (Cooper *et al.*, 2003). The position of each object is set in the virtual world. Each object added by the user, encapsulates its own data (private properties such as height, width and location). Users can select primitive methods for each object from a list (B in Figure 5.14). User defined methods can also be added.

The drag-and-drop method of code generation in *Alice* allows the user to focus on designing the solution instead of the complexity of the syntax and punctuation (Cooper *et al.*, 2003). The focus away from syntax and punctuation does mean that *Alice* does not assist novice programmers in their knowledge of programming language syntax. However, research has shown (Cooper *et al.*, 2003) that novice programmers quickly master syntax when making the transition from *Alice* to a programming language.

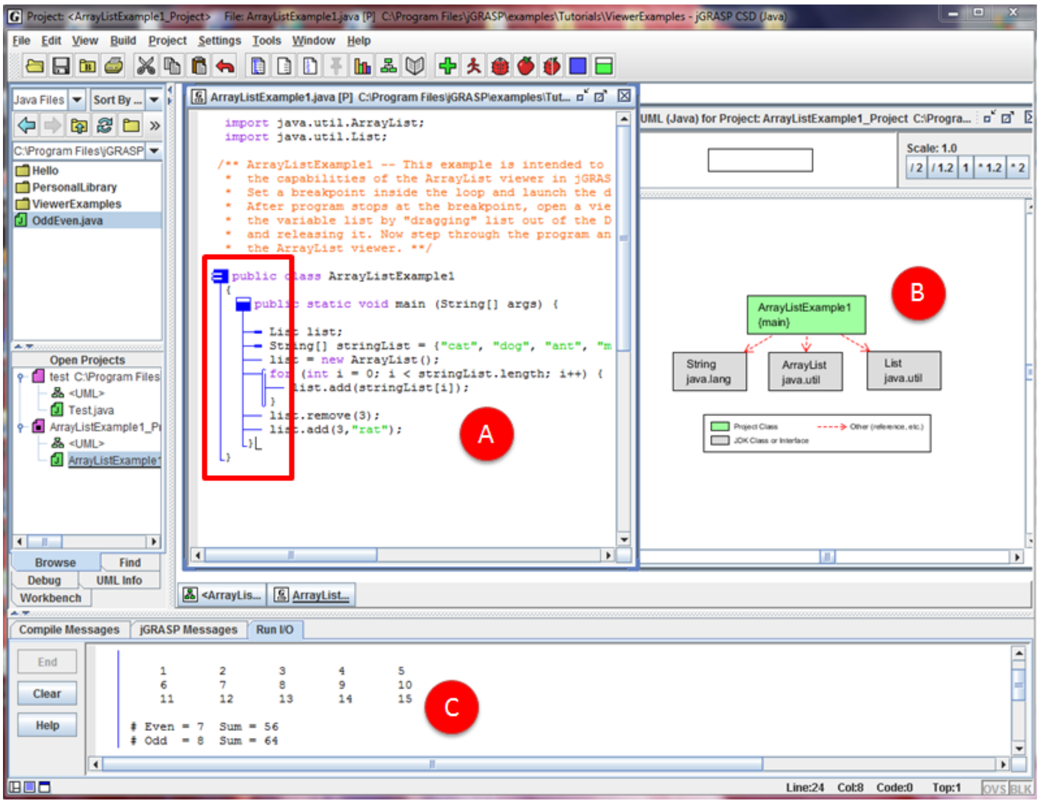
5.3.10 jGRASP

jGRASP[18] is a visualisation tool that provides automatic generations of control structure diagrams for source code visualization (red box in Figure 5.15) and UML Class diagrams (B in Figure 5.15). *jGRASP* was created to provide visualisations to help users improve their understanding of programming code. *jGRASP* provides a lightweight development environment that supports Java, C, C++, Objective-C, Ada and VHDL programming languages.

Figure 5.15 shows a screenshot of the *jGRASP* development environment. Java class files can be opened and edited in the main window (A in Figure 5.15). A UML class diagram is automatically generated by *jGRASP* from Java class files in a project (B in Figure 5.15). The resulting program of a code solution in *jGRASP* is a command-line type text interface that displays text output and receives text input from the user (C in Figure 5.15).

A *jGRASP* user is able to insert a breakpoint that will stop the execution of code at any point and allow the user to view the state of a data structure such as a linked list, array or binary tree (Cross, Hendrix, Jain and Barowski, 2007). Data structures are updated in

[18] http://jgrasp.org



```
 import java.util.ArrayList;
 import java.util.List;

/** ArrayListExample1 -- This example is intended to
 *  the capabilities of the ArrayList viewer in jGRAS
 *  Set a breakpoint inside the loop and launch the d
 *  After program stops at the breakpoint, open a vie
 *  the variable list by "dragging" list out of the D
 *  and releasing it. Now step through the program an
 *  the ArrayList viewer. **/

public class ArrayListExample1
{
   public static void main (String[] args) {

      List list;
      String[] stringList = {"cat", "dog", "ant", "m
      list = new ArrayList();
      for (int i = 0; i < stringList.length; i++) {
         list.add(stringList[i]);
      }
      list.remove(3);
      list.add(3,"rat");
   }
}
```

```
   1        2        3        4        5
   6        7        8        9        10
   11       12       13       14       15

# Even = 7  Sum = 56
# Odd  = 8  Sum = 64
```

Figure 5.15: jGRASP main code window

the viewer as individual statements are executed using functionality to step through the code one line at a time. This provides a deeper understanding of the use of data structures.

5.4 PAT Selection

The selection criteria (Section 4.4) are used to determine the suitability of the PATs identified in Section 5.3 for the teaching of IT programming in South African secondary schools. The PATs are evaluated on three categories of selection criteria: *programming knowledge* (Section 5.4.1), *programming skills* (Section 5.4.2) and *programming concepts* (Section 5.4.3). The techniques used by PATs to create program solutions, represent output programs and explain programming concepts, are also identified for each of the PATs (Section 5.4.4). Where applicable, the criteria are evaluated with respect to the learning preferences that are catered for by the PAT - indicated by a V (visual), A (Aural), R (Read/write) and/or K (Kinaesthetic). Alternatively, a check mark (✓) is used to indicate if a PAT meets the criteria.

5.4.1 Programming Knowledge

All of the reviewed PATs, with the exception of *Alice* and *Scratch*, can assist users to improve their knowledge of programming language syntax (Table 5.2) as program solutions are implemented or generated in a particular programming language (either Delphi/Pascal, Java or both). The statements used by *RoboMind* are similar to Java but the editor can be adapted to compile statements that users are more accustomed to using in a particular programming language. *Scratch* and *Alice* use drag and drop building blocks.

All of the PATs are constructivist to promote self-study (Table 5.2) by learners to improve their understanding of programming concepts. All of the PATs assist with the development of knowledge of programming principles and concepts, however, different learning preferences are addressed by each of the PATs in meeting this criterion. All of the PATs address the development of programming knowledge and principles criterion using the visual learning preference and all but one – *PlanAni* – use the kinaesthetic learning preference by allowing users to create their own programming solutions using the PAT. *Jeliot*, *Ville* and *PlanAni* also address the Read/Write learning preference, by providing textual explanations or questions to assess understanding of programming concepts. *PlanAni* is the only tool that does not allow users to program a solution, however, explanations of built-in examples help build user knowledge of programming principles, particularly the role of variables.

Table 5.2: Evaluation of PATs using selection criteria: Programming knowledge

Criteria	**RoboMind**	**BlueJ**	**Greenfoot**	**Scratch**	**B#**	**Jeliot**	**Ville**	**PlanAni**	**Alice**	**jGRASP**
Assists with the learning of the Delphi programming language syntax	•	J	J		D	J	J	DJ		J
Assists with developing knowledge of programming principles & concepts	VK	VK	VK	VK	VK	VK	VRK	VR	VK	VK
Constructivist to promote self-study	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Assists with the application of programming knowledge						✓	✓	✓		
Assists with the understanding of code execution	VR	V	V	V	VR	VR	VR	V	V	V

✓= PAT meets the criteria
• = PAT can be adapted to meet the criteria
D = Delphi
J = Java

V = Visual
A = Aural
R = Read/Write
K = Kinaesthetic

Jeliot, *Ville* and *PlanAni* assist with the application of programming knowledge. *Jeliot* and *PlanAni* use visualisation to demonstrate how different programming knowledge is applied to solving a programming solution. *Ville* includes a variety of simple and complex programming examples to demonstrate to users how programming knowledge can be applied to solve programming problems.

Table 5.3: Ranking of PATs based on evaluation of programming knowledge selection criteria

	PAT	**Score**
1.	PlanAni	15
2.	RoboMind (with adaptations)	13
	B#	13
4.	Ville	10
5.	Jeliot	10
6.	BlueJ	8
	Scratch	8
	Alice	8
	jGRASP	8

The top five PATs that address the programming knowledge selection criteria have been identified (Table 5.3). The PATs have been ranked based on the ranking score obtained by considering the number and importance of selection criteria that are addressed by the PAT (Appendix F). PATs that address the Delphi programming language syntax are preferred, as the selected PATs are provided to IT learners using Delphi. Only PATs that support the Delphi programming language are considered to meet the first criteria when the scores are calculated.

PlanAni meets all of the programming knowledge selection criteria. *B#* and *RoboMind* (after adaptations) cater for the Delphi programming language and are thus ranked above *Jeliot*. *Jeliot* addresses all of the programming knowledge criteria, however, *Jeliot* only provides support for the Java programming language syntax.

Ville is ranked above *Jeliot* as the former addresses the kinaesthetic learning preference in addition to the read/write and visual learning preferences in the development of knowledge and programming principles (Table 5.2).

5.4.2 Programming Skills

Scratch and *Alice* make use of the drag-and-drop interface which ensures that users can only use the correct statements and syntax. These two PATs thus indirectly support error handling but learners' error messages are not provided to learners to support debugging; hence, the blocks are greyed out (Table 5.4). Error handling and compiler messages are also not applicable for *PlanAni* as built-in examples are used which cannot be edited by the user.

Table 5.4: Evaluation of PATs using selection criteria: Programming skills

Criteria	**RoboMind**	**BlueJ**	**Greenfoot**	**Scratch**	**B#**	**Jeliot**	**Ville**	**PlanAni**	**Alice**	**jGRASP**
Promotes problem solving & planning				VK	VK				VK	
Provides simple error messages to assist with debugging	R			■	R			■	■	
Develops code comprehension				V	V	VR	VR	VR	V	
Feedback to guide solution creation				V	V				V	
Feedback regarding errors	R	R	R	■	R	R	R	■	■	R

✓= PAT meets the criteria
•= PAT can be adapted to meet the criteria
V = Visual
A = Aural
R = Read/Write
K = Kinaesthetic

Only two PATs - *RoboMind* and *B#* - use simple error messages to inform users of syntax errors in the code using language and terms that are simple for novice programmers to understand (Table 5.4). The remaining PATs - *BlueJ*, *Greenfoot*, *jGRASP* and *Jeliot* - use the standard Java compiler. The error messages are the same messages that expert programmers would receive in professional programming environments such as Netbeans or Eclipse.

The statements used in *Scratch* and *Alice* indicate to users where conditions or variables must be inserted or if other statements must be included within a loop or control structure in order to guide the creation of a solution (Table 5.4). *B#* allows users to build a solution using a flowchart diagram. Users are able to visualise the execution of the solution using the flowchart.

Table 5.5: Ranking of PATs based on evaluation of programming skills selection criteria

	PAT	Score
1.	B#	15
2.	Scratch	12.5
	Alice	12.5
4.	PlanAni	5.5
5.	RoboMind	5
6.	Jeliot	4
	Ville	4
8.	BlueJ	1
	Greenfoot	1
	jGRASP	1

The PATs can also be ranked based on the number and importance of programming skills selection criteria that are addressed (Table 5.5). Priority weightings are used to determine the score for each of the PATs in addressing the programming skill selection criteria (Appendix F). *B#* addresses all of the programming skills selection criteria. *Scratch* and *Alice* do not need to provide feedback regarding errors or simple error messaging as the drag and drop technique used by these two PATs do not allow syntactical errors to occur.

5.4.3 Programming Concepts

BlueJ, *Greenfoot* and *jGRASP* are able to open and compile any java source files, thus allowing them to implement all the programming concepts (Table 5.5). *BlueJ* and *jGRASP* only assist with the understanding of certain programming concepts, such as objects and classes, debugging and variables, for which visualisation is used or status information such as the current line of execution, is provided.

Programs in *Greenfoot* create microworld games or animations allowing users to observe the resulting behaviour of programming concepts used in the program. *Jeliot* allows users to visualise the code execution of programming concepts. *Jeliot* is not able to implement access to a database or handling text files (reading from or writing to).

Table 5.6: Evaluation of PATs using selection criteria: Programming concepts

Criteria	**RoboMind**	**BlueJ**	**Greenfoot**	**Scratch**	**B#**	**Jeliot**	**Ville**	**PlanAni**	**Alice**	**jGRASP**
Two-dimensional arrays		◇	✓			✓				✓
String handling		◇	✓	✓		✓	✓			◇
One-dimensional arrays		◇	✓	✓		✓	✓	✓		✓
Procedures	✓	◇	✓	✓		✓	✓		✓	◇
Functions		◇	✓			✓				◇
repeat-loops	•	◇	✓	✓	✓	✓	✓	✓	✓	◇
while-loops	✓	◇	✓	✓	✓	✓	✓	✓	✓	◇
Objects & classes		✓	✓	✓		✓			✓	✓
for-loops	✓	◇	✓	✓	✓	✓	✓	✓	✓	◇
if-statements	✓	◇	✓	✓	✓	✓	✓	✓	✓	◇
Correct use of parameters	✓	◇	✓			✓	✓			✓
SQL statements		◇	✓							◇
Accessing database		◇	✓							◇
case-statements		◇	✓		✓	✓				✓
File handling		◇	✓							◇
Variables	•	◇	✓	✓	✓	✓	✓	✓	✓	✓
Input		◇	✓	✓	✓	✓		✓	✓	◇
Output		◇	✓	✓	✓	✓	✓	✓	✓	◇

✓= PAT assists with understanding of the concept.

◇ = Concept can be implemented. No specific assistance provided to assist with understanding.

• = PAT can be adapted to include concept.

Table 5.7: Ranking of PATs based on evaluation of programming concept selection criteria

	PAT	**Score**
1.	Scratch	109
2.	Alice	76
3.	RoboMind (with adaptations)	72
4.	PlanAni	66
5.	B#	55

PATs have been ranked based on an evaluation of the programming concept selection criteria (Table 5.7, Appendix F). PATs that support or can be adapted to support the Delphi programming language, namely *RoboMind*, *B#* and *PlanAni* are considered for selection based on programming concept selection criteria. *Scratch* and *Alice* are also considered as the graphical objects used to create program solutions are independent of programming language and are designed to promote an understanding of the programming concepts, although not programming language syntax.

5.4.4 Techniques used by the PATs

The categories and items used to evaluate the techniques used by different PATs (Table 5.8), are discussed in Section 5.2.2. A distinction is made between how the code is represented as it is being created, the method used to create the program and the form of the final program (Table 5.8).

RoboMind is described as a microworld as the purpose of the program created is to control a robot in a map environment. *Scratch* and *Alice* allow the user to create games and animations and are not considered microworlds. *Scratch* supports both the text and pictures techniques to represent program solutions as the graphical blocks used to construct the program use programming concept text similar to a programming language syntax. Only *B#* uses a flowchart technique to represent the program solution.

Table 5.8: Evaluation of techniques used by PATs

	RoboMind	BlueJ	Greenfoot	Scratch	B#	Jeliot	Ville	PlanAni	Alice	jGRASP
Representation of the program solution										
Text	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Flowchart					✓					
Pictures		✓	✓	✓						
Construction of programs										
Typing of code	✓	✓	✓			✓		✓		✓
Assembling or positioning graphical objects		✓	✓	✓	✓				✓	
Selecting/form filling	✓			✓					✓	
Resulting program										
Text (input & output)		✓			✓	✓	✓	✓		✓
GUI				✓						
Animation			✓	✓					✓	
Microworld	✓									
Support to understand programs										
Debugging	✓	✓	✓		✓	✓	✓			✓
Animation of program execution						✓	✓	✓		
Testing user knowledge (questions)						✓	✓			
Preventing syntax errors										
Selection from valid options	✓			✓					✓	
Dropping only in valid locations				✓	✓				✓	
Informative syntax error messages	✓									

✓= PAT meets the criteria

None of the PATs evaluated execute the program by providing a GUI similar to Delphi programs for the user to interact with. *Scratch* allows users to create an interactive interface by implementing graphical picture objects as buttons that can use similar components to a Delphi GUI interface. *RoboMind* is the only evaluated PAT that provides informative syntax error messages, althougth *Alice* and *Scratch* do not require support for error messages. These two PATs only allow programming concept blocks to be selected from and dropped into valid locations. Support for syntax error messages is thus not required.

5.4.5 PAT Selection for this Research Study

Primary Objective 2 is to evaluate the impact of PATs on IT learner understanding of programming concepts and motivation towards programming. Grade 10 and Grade 11 learners at each school participating in the research study receive a PAT to use. The PATs selected for evaluation by the participating schools should assist learner understanding of programming concepts. The programming knowledge and programming concept criteria are of more relevance than the programming skills criteria which are not evaluated.

The selection of PATs also considers the techniques used by the PATs to create program solutions. The PATs selected should differ in the techniques used in order to evaluate learner feedback on different methods of program creation and the format of resulting programs on learner understanding and motivation. PATs that allow learners to create different program solutions are preferred to PATs that only demonstrate programming examples.

Table 5.9: Summary of PAT evaluation using selection criteria

	PAT	Knowledge	Skills	Concepts	Total	Selected
1.	B#	13	15	5.5	33.5	✓
2.	Scratch	8	12.5	10.9	31.4	✓
3.	Alice	8	12.5	7.6	28.1	
4.	RoboMind (Adapted)	14	5	7.2	26.2	✓
5.	PlanAni	15	3	6.6	24.6	

PATs that can assist with the understanding of the Delphi programming language are considered for selection (Table 5.9). The score calculated for *RoboMind* is based upon the implemetation of adaptations to support the Delphi programming language. *Alice* is not considered for selection as *Scratch* and *Alice* use similar techniques to create program solutions and produce a resulting program and *Scratch* obtained a higher score for addressing the programming concept criteria than *Alice*.

Four schools have consented to participate in the research study, however, treatment sample sizes at two of the schools are smaller than the other two schools. It was decided to select three PATs for evaluation in the schools, where two schools would receive the same PAT so that the number of learners evaluating the three PATs would be approximately equivalent (Section 7.2.1).

Scratch is selected as it has the highest score with regards to programming concepts. *RoboMind* is selected as it has the next highest programming concept score since *Alice* is not being considered and the second highest programming knowledge score. *PlanAni* does not allow users to create their own programs and only six example programs are provided. *B#* is the only one of the PATs under consideration that uses a flowchart technique for the creation of program solutions.

Scratch is provided to the two participating schools with the smallest number of participants. The reason for selecting *Scratch* as the PAT to provide to these schools, firstly because *Scratch* provides the best support for programming concepts (Table 5.9) and, secondly, because it is the PAT recommended for use by Grade 10 learners in the implementation of the new IT curriculum (Department of Basic Education, 2011).

5.5 Conclusion

This chapter identifies several PATs that have been developed for use by novice programmers to improve their understanding of programming concepts. All these tools are freely available. The formulated selection criteria (Section 4.4) are used to evaluate the tools. RQ3: *What PATs exist that would be suitable for use in SA secondary schools?* and Primary Objective 1: *To identify existing introductory programming assistance tools (PATs) that can be used to support learner understanding of programming concepts included in the IT subject curriculum implemented in a case study of South African secondary schools* are addressed in this chapter by identifying PATs and providing an indication of which criteria are met by each of the tools (Section 5.2).

The most important consideration when selecting a tool to teach IT programming is which programming language - Delphi or Java - is taught to the IT learners. Only a few of the PATs support Delphi (Pascal). IT learners that require support for the Java programming language have a wider variety of tools that cover all programming concepts - *BlueJ*, *Greenfoot*, *Jeliot* and *JGRASP*. These tools all use the standard Java compiler. *Jeliot* would be of most benefit to learners requiring assistance with programming in general as it also assists learners with code comprehension. *BlueJ* and *Greenfoot* focus more on promoting understanding of object-oriented programming concepts.

None of the PATS identified will assist IT learners with all programming concepts associated with the Delphi programming language. Most of the tools that support Delphi (Pascal) address the majority of the difficult concepts identified (Chapter 4).

The three most appropriate PATs to support the teaching of the Delphi programming language are identified for evaluation in South African schools teaching Delphi to its IT learners. These PATs are:

- *RoboMind*
- *Scratch*
- *B#*

These PATs all use different techniques to assist user understanding of programming concepts. Three PATs are identified for evaluation in the four participating tools. Each school receives one PAT. *Scratch* is provided to two of the participating schools with the lowest participation numbers.

The selected PATs are provided to participating schools in order to achieve Primary Objective 2: *To evaluate the impact of the selected programming assistance tools (PATs) on a novice programmer's understanding of programming concepts included in the IT subject curriculum implemented in a case study of South African secondary schools.* The differences between the three selected PATs and Delphi are identified before the impact of PATs on IT learner understanding of programming concepts can be evaluated. Chapter 6 discusses the adaptations made to *RoboMind* as well as the differences in the way certain programming concepts are presented in the three tools compared to Delphi.

Chapter 6

PAT Preparation for IT Learners

6.1 Introduction

Primary Objective 1 of this study is to identify programming assistance tools (PATs) suitable for use by IT learners in South African secondary schools. *RoboMind*, *Scratch* and *B#* have been identified (Section 5.5) as appropriate PATs. Despite their suitability, there are various shortcomings in terms of the educational support of the syntax of programming concepts which had to be identified. Where possible, these shortcomings were addressed before each of the PATs was provided to IT learners and thus before Primary Objective 1 could be met.

In order for the PATs to support the learning of programming concepts in the IT subject, the syntax of the programming concepts implemented in the PATs should be the same or similar to the syntax of the Delphi programming language as IT learners were to use the PAT to *support* the learning of programming using the Delphi programming language. No adaptations were implemented to change the manner in which the PATs visually represent programming concepts or the techniques used to construct program solutions and display program output. The techniques used by PATs were also evaluated and thus were not adapted before the evaluation and without research to support any adaptations to the visual representations of the programming concepts and techniques used to create program solutions.

The aim of this chapter is to compare the implementation of concepts in each of the PATs to the implementation of corresponding concepts in Delphi. *RoboMind* is the only PAT for which the programming concept syntax can be adapted to match the Delphi syntax. A discussion of the adaptations made to *RoboMind* addresses RQ4: *How can the selected PATs be adapted for use by IT learners in SA secondary schools to support the understanding of programming concepts?*. Addressing RQ4 achieves Secondary Objective 1.2:

Adapt selected PATs to make them suitable to support the achievement of programming learning outcomes in the IT subject curriculum implemented in South African secondary schools.

This chapter describes how Delphi implements programming structures such as looping and *if*-statements (Section 6.2) and then identifies how the implementation of certain programming concepts in each of the three PATs differs from the implementation in Delphi (Section 6.3). The adaptations to *RoboMind*, to allow it to implement concepts in a manner (syntax and semantics) similar to Delphi, are presented (Section 6.3.1). The reasons why *Scratch* and *B#* were not adapted are presented, together with a description of the differences between Delphi and *Scratch* (Section 6.3.2) and Delphi and *B#* (Section 6.3.3), respectively.

6.2 Programming Concepts in Delphi and the PATs

The three PATs selected (Section 5.5) have been identified as suitable for use by IT learners. The use of the PATs was self-administered and it was important that, without the guidance of the teacher, the PATs could assist learners to learn to program and not confuse learners. If a PAT is to support IT learner understanding of programming concepts, the implementation of programming language concepts in each of the PATs should be as close to that of the programming concepts implemented in Delphi. Table 6.1 lists the programming concepts, which IT learners are required to have an understanding of (Department of Education, 2008).

Scratch supports the most programming concepts of the three PATs (Table 6.1) but the syntax of programming concepts implemented in *Scratch* differ from the Delphi programming language syntax. Programming concept syntax in *Scratch* cannot be adapted as the source code is not available. Programming concepts supported by *B#* match the Delphi programming language syntax, except for input and output that differ. Programming concepts supported by *RoboMind* needed to be adapted in order for the syntax to match the Delphi programming language syntax.

The implementation of selected programming concepts in Delphi are presented in Section 6.2.1. An understanding of how programming concepts are implemented in Delphi is required in order to identify differences between the implementation of programming concepts in the PATs and to determine adaptations required for *RoboMind*.

Table 6.1: Evaluation of RoboMind, Scratch and B# using selection criteria: Programming concepts

Programming Concept	**Unadapted RoboMind**	**Scratch**	**B#**
Two-dimensional arrays			
String handling		⋆	
One-dimensional arrays		⋆	
Procedures	✓	⋆	
Functions			
repeat-loops	•	⋆	✓
while-loops	◇	⋆	✓
Objects & classes		⋆	
for-loops	◇	⋆	✓
if-statements	◇	⋆	✓
Correct use of parameters	⋆		
SQL statements			
Accessing database			
case-statements			✓
File handling			
Variables	•	⋆	✓
Input		⋆	⋆
Output		⋆	⋆

✓= Concept supported and syntax matches Delphi

◇ = Concept supported but syntax does not match Delphi. Can be adapted.

⋆ = Concept supported but syntax does not match Delphi. Cannot be adapted.

•= Concept not supported but PAT can be adapted to include concept

blank = PAT does not support and cannot be adapted to support the concept

The differences between Delphi and *RoboMind* are evaluated and adaptations made to *RoboMind* (Section 6.2.2) are presented. The differences between *Scratch* and Delphi are presented (Section 6.2.3). No adaptations were made to *Scratch* as the source-code is not available. The difference between *B#* and Delphi related to the method of interaction with the user in terms of input and output is presented (Section 6.2.4). Although the source code is available, no adaptations were made to *B#* to address the differences between the string handling associated with the input and output programming concepts.

6.2.1 Delphi

Delphi refers to Borland Software Corporation's Delphi development environment, which uses the Object Pascal programming language (Kerman, 2002). Object Pascal evolved from the Pascal programming language to support the development of Windows-based applications.

The basic programming concepts and structures, such as variables, looping and decision structures, are implemented as in the original Pascal programming language. Due to the support for Windows-based applications, Delphi can be used to develop GUI user interfaces that use components such as buttons, edit boxes, labels and pop-windows to receive input from and display output to users.

A description of the implementation of programming concepts in Delphi makes it easier to highlight the differences between Delphi and the three PATs in terms of programming concept syntax. Only selected programming concepts (as identified earlier) are presented. Before presenting the specific programming concepts, the following programming *rules* specific to Delphi are highlighted:

1. The keywords, `begin` and `end`, are used to delimit code blocks.
2. Code statements in Delphi (Object Pascal) are separated using a semicolon (;) (Kerman, 2002). The line before the `else` keyword as part of an *if-then-else* statement does not end with a semicolon (lines 8-9 of Listing 6.3).
3. Variable, procedure/function and reserved keyword names are not case sensitive.
4. `:=` is used to assign a value (right-hand side) to a variable or property (left-hand side), while `=` is a boolean comparison of two values.

Procedures A procedure is a subroutine that does not return a value. A procedure is defined using the `procedure` keyword (line 1 of Listing 6.1). This is followed by a procedure name (`moveSquare` in Listing 6.1) and a declaration of any parameter variables that must be passed to the procedure.

Local procedure variables are declared (line 2 of Listing 6.1) and the code to implement the task that the procedure should perform is enclosed in a `begin..end` block. The procedure is called using the procedure name followed by any data to be passed as parameters in brackets (line 15 of Listing 6.1).

Correct use of parameters Parameter variables are declared by specifying the parameter name and data type in brackets after the procedure name. In Listing 6.1 (line 1) an integer parameter named `size` is passed to the procedure. The parameter variable is used as a local variable in the procedure body (line 6 of Listing 6.1). In Listing 6.1 the number of steps to move is obtained as input from the user (line 14 of Listing 6.1)

Listing 6.1: Delphi: Procedures and use of parameters

```
procedure moveSquare(size :integer);
var i : integer;
begin
  for i := 1 to 4 do
  begin
    moveForward(size); //custom procedure
    rightTurn();       //custom procedure
  end;
end;

procedure whenButtonIsClicked();
var steps : integer;
begin
  steps := strToInt(edtSteps.text);
  moveSquare(steps);
end;
```

One-dimensional arrays Delphi supports static and dynamic arrays. The length of static arrays is defined when the array variable is declared (line 1 of Listing 6.2), while the length of dynamic arrays can be set in the code body. Array elements can only have one data type, indicated when the array is declared.

The starting and ending indices of static arrays are also indicated when the array is declared. The starting index is thus only zero (0) if specified. Square brackets ([]) are used to reference an element in the array at a specific position. Array elements are referenced when assigning values to the element (line 8 in Listing 6.2) or reading the value of an element (line 12 and 20 in Listing 6.2).

***repeat..until* loops** A *repeat*-loop is an indeterminate loop structure and can be used when the number of times the loop will execute is not known and cannot be determined before the loop starts executing. A *repeat*-loop is implemented using the `repeat` and `until` keywords to block the code that must be repeated.

The boolean condition that must be checked to determine whether the loop continues executing, appears after the `until` keyword (line 13 in Listing 6.2). The condition statement indicates when the loop must stop executing (that is, the loop will continue executing if the condition is false). The boolean condition is only checked after the code in the loop statements have been executed, thus a *repeat*-loop is a post-conditional loop and always executes at least once.

Listing 6.2: Delphi: Arrays and looping

```
 1 var arrNums : array[1..10] of integer;
 2     counter, sum : integer;
 3 begin
 4   sum := 0;
 5   counter = 1;
 6 
 7   for counter := 1 to 10 do
 8     arrNums[counter] := counter*2;
 9 
10   repeat
11     sum := sum + arrNums[counter];
12     inc(counter);   //increments counter by 1
13   until (counter > 10);
14 
15   sum := 0;
16   counter = 1;
17 
18   while (counter <= 10) do
19   begin
20     sum := sum + arrNums[counter];
21     inc(counter);   //increments counter by 1
22   end;
```

***while..do* loops** A *while*-loop is also an indeterminate loop structure implemented using the `while` and `do` keywords either side of a boolean condition that must be true for the loop to continue executing (line 18 of Listing 6.2). The condition is checked before the loop executes thus a *while*-loop is also a pre-conditional loop.

The *while*- and *repeat*-loops in Listing 6.2 perform the same task. The difference between the two is the boolean condition used to stop the loops (line 13 and line 18 in Listing 6.2) as well as the fact that the *while*-loop is pre-conditional and the *repeat*-loop is post-conditional.

***for*-loops** A *for*-loop in Delphi (line 4 in Listing 6.2) is a determinate loop structure (Kerman, 2002) and is used when the exact number of times the loop must execute can be specified using starting and ending values. The *for*-loop includes a loop counter variable (`counter` in Listing 6.2). When Delphi executes the *for*-loop code, the counter variable is initially assigned to the starting value and is compared to the ending value before executing the body of the loop. A *for*-loop is thus a pre-conditional loop. The counter variable is incremented or decremented by the value one (1) depending on if the *for*-loop uses the keyword `to` or `downto`, respectively.

String handling A string data type in Delphi is an array of characters. Any character in a string can be obtained by referencing the character position. Strings can be concatenated using a plus (+) sign (lines 8, 10 and 12 in Listing 6.3). Only string values are concatenated. Integer or real numbers that are to be included in a string text must be converted to a string using `intToStr` (lines 8 and 10 in Listing 6.3) or `floatToStr`, respectively.

***if*-statement** In Delphi, the *if*-statement boolean (true/false) condition is placed between the `if` and `then` keywords (line 7 in Listing 6.3). The `else` keyword is used to indicate statements that must be executed if the boolean condition (line 7 in Listing 6.3) evaluates to false.

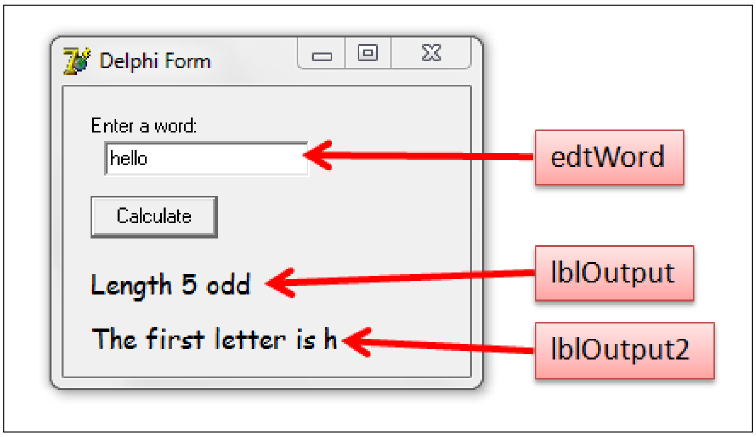
Listing 6.3: Delphi: String handling, *if*-statements, input and output

```
 1 var num : integer;
 2     inpStr : string;
 3 begin
 4   inpStr := edtWord.Text;
 5   num := length(inpStr);
 6
 7   if (num mod 2 = 0) then
 8     lblOutput.caption := 'Length␣' + intToStr(num) + '␣even'
 9   else
10     lblOutput.caption := 'Length␣' + intToStr(num) + '␣odd';
11
12   lblOutput2.caption := 'The␣first␣letter␣is␣' + inpStr[1];
13 end;
```

Variables Variables are declared after the procedure (or function) header and before the first `begin` keyword. Variables cannot be declared in the code body as is the case with other programming languages such as Java. The `var` keyword is used to indicate the declaration of variables. The variable names are listed followed by a colon (:) and the data type of the variables (lines 1-2 in Listing 6.3). Multiple variable names of the same data type can be listed together. Values are assigned to variables using `:=` .

Input IT learners are taught to receive input for program from users using graphical components such as an edit box (`edtWord` in Figure 6.1), radio button or checkbox. Variables can be assigned to the value of graphical components. For example, line 4 of the code in Listing 6.3 assigns a value entered by a user in the edit box (Figure 6.1) to the string variable `inpStr`.

Output IT learners are taught to display the output of a Delphi program in a graphical component such as a label (`lblOutput` and `lblOutput2` in Figure 6.1). A string value is typically assigned to a property of the graphical component. For example, a string value is assigned to the caption properties of the two label components in Listing 6.3 (lines 8, 10 and 12).



Length 5 odd

The first letter is h

Figure 6.1: Sample program (Listing 6.3) executed in Delphi

Figure 6.1 is a screenshot of the resulting GUI interface generated for input from and output to the user during execution of the sample program from Listing 6.3. The user types a word in the edit box (`edtWord`), clicks the button (named *Calculate*) and the program (Listing 6.3) will display the output in the two label components.

6.2.2 RoboMind

RoboMind supports a Java-type programming language called *Robo* that can implement looping, *if*-statements and self-defined procedures with or without parameters. *RoboMind* also supports the use of various pre-defined commands used to control the actions of the robot in the map world (Section 5.3.1).

The implementation of programming concepts in the unadapted *RoboMind* are presented (Section 6.2.2.1). The *RoboMind* source code has been adapted so that the supported programming concept syntax is the same or similar to the Delphi programming language syntax (Section 6.2.2.2). The source code has also been adapted so that *RoboMind* can support variables and *repeat*-loops.

6.2.2.1 Unadapted RoboMind

Programs in the unadapted *RoboMind* do not comply with any of the general Delphi programming *rules* (Section 6.2.1):

1. Curly brackets are used to delimit the start ({) and end (}) of code blocks – `begin` and `end` keywords are used in Delphi. *RoboMind* does not require a main code block to be delimited – there is no opening `{` or closing `}` .
   RoboMind requires code blocks to be delimited for programming statements related to *if*-statements and looping commands, regardless of the number of statements that are to be executed. Delphi does not require code blocks specified for *if*-statements and loops although only the first code statement is associated with the preceding *if*, *else*, *while*-loop, or *for*-loop command.

2. Code statements are not separated with a semicolon. No statement separator is used in *RoboMind*.

3. Procedure/function and reserved keyword names are case sensitive.

4. No assignments or boolean evaluations of equality (=) are supported in the unadapted *RoboMind*.

Table 6.2: Correspondence of programming concepts implemented in Delphi and unadapted RoboMind

Concept	RoboMind	Delphi
Procedures (and parameters)	`procedure turn (a,b)`	`procedure turn(a,b:integer)`
while-loop	`repeatWhile(frontIsClear())`	`while (frontIsClear()) do`
for-loop	`repeat(3)`	`for i := 1 to 3 do`
if-statement	`if (frontIsClear())`	`if (frontIsClear()) then`

The unadapted *RoboMind* supports five programming concepts, namely procedures, the use of parameters, *while*-loops, *for*-loops and *if*-statements. Only the implementation of procedures in the unadapted *RoboMind* is the same as in Delphi (Table 6.2).

Procedures The same as in Delphi, *RoboMind* uses the `procedure` keyword to define a procedure. A procedure is called using the procedure's name.

Correct use of parameters Parameters can be passed to procedures in *RoboMind*. However, only integer parameters are supported. The parameter data type is thus not specified when the procedure is defined.

***while*-loop** The *while*-loop is supported in the unadapted *RoboMind*, however, the syntax differs from the implementation in Delphi (Table 6.2). The `repeatWhile` keyword is used instead of the `while..do` keywords.

***for*-loop** *RoboMind* implements a determinate loop structure as `repeat(x)`, where x indicates the exact number of times the loop must execute. No counter variable is used as in Delphi.

***if*-statements** The only difference between the unadapted *RoboMind* and the Delphi programming language is that Delphi requires a `then` keyword after the condition is stated (Section 6.2.1) while *RoboMind* does not. For example, Table 6.2 compares statements in Delphi and the unadapted *RoboMind* that would be used to check if the front is clear. Nested *if*-statements are also supported by *RoboMind*.

6.2.2.2 Adapted RoboMind

RoboMind's source code is open source Java code. Adaptations can be made to any aspect of *RoboMind* including the graphics and the interpreter that converts a code solution in *RoboMind* to actions that the robot must perform. The scope of the adaptations for this research were restricted to making the syntax of programming concepts similar to or the same as the Delphi programming language syntax.

Adaptations were thus restricted to the implementation of the *RoboMind* compiler, specifically the *RoboCompiler.java* file (*Robo.Script.Command* package). Where reference is made to a file that has been adapted, the file (and the package in which the file is included) is part of the *RoboMind* source code.

Adaptations to the *RoboMind* source code could only be implemented after the method used to implement the *RoboMind* software tool, specifically the compiler, had been identified and understood. The compiler is implemented using a list of commands. Each command has a pointer to the next command. In the case of decision commands based on a boolean condition, one pointer indicates the command to execute if the condition evaluates as true and another if the condition evaluates as false.

A program in *RoboMind* is compiled in a two level process. The first compile level parses the program code to ensure that the syntax is correct. Different commands are identified and parsed accordingly, for example, if the *while* keyword is identified, a boolean expression should be parsed next. If there are no errors parsing the command structure, instances of specific command objects are created and added to a list of program commands. The program commands are maintained using an array list. Each program command instance maintains data such as the current line of execution in the editor and variable names or integer values passed as arguments for movement commands, for example.

The second compile level binds variable values to the list of program commands. For example, if the line `forward(n);`is used in the code, a *moveForwardCommand* object would have been added to the program commands list in the first compile level. In the second compile level the value assigned to the variable `n` in the code is associated with the *moveForwardCommand.* The second compile level also assigns indices to program command objects to assist with code execution. For example, the execution of an *if*-statement must point to the index of the first statement to be executed depending on the value of the boolean condition. The index is the position of the associated program command in the array list of program commands. After the second compile level has completed, the array list of program commands is interpreted and the commands are converted into visual actions performed by the robot.

Programming concepts and rules that are supported in *RoboMind* but the syntax does not match Delphi, were adapted by changing the keyword pattern strings and/or the order in which keywords or symbols that are parsed in the first compile level. *RoboMind* was also adapted to include *repeat*-loops and variables as these two concepts are already supported to a certain extent. *Repeat*-loops are similar to *while*-loops. The only differences are the order in which keywords are parsed and the boolean expression value (true/false) that is returned. *RoboMind* supports parameter variables as local variables in procedure code. The data structures used to maintain and use parameter variables in procedures were extended to support variables in the main *RoboMind* code.

Adaptations have been made to *RoboMind* source code so that the code implemented in *RoboMind* complies with the general Delphi programming rules (Section 6.2.1). The adaptations have been implemented as follows:

1. **Delimitation of code blocks:** The `begin` and `end` keywords have been defined in the English version of the Robo Language Definition file (*RLD_default_en.properties* in the *misc* package). The associated string patterns are initiated in the *RoboCompiler.java* file (*robo.script* package), namely `begin`, `end;` and `end` (used before an `else` keyword). The `begin` and `end` keywords are recognised instead of the curly brackets when the source code is parsed.
   The adapted *RoboMind* enforces the requirement of the unadapted *RoboMind* that code blocks be delimited to associate program statements with *if*, *else*, *while*-loop and *for*-loop commands.

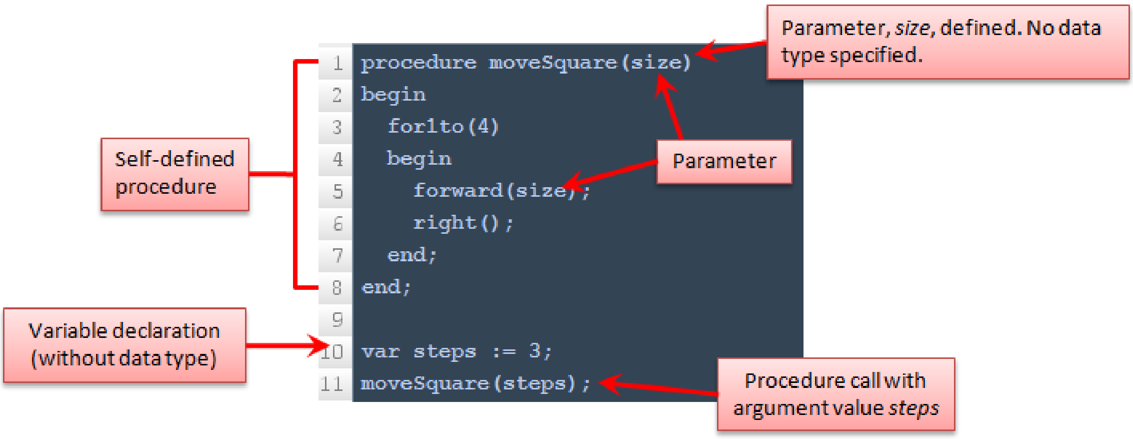
2. **Code statements are separated with a semicolon:** *RoboMind* has been adapted to accept the semicolon as the programming statement separator. This was implemented by parsing the semicolon character after statements such as procedure calls, variable declarations and assignments, the `end` keyword and after the `until` statement of a *repeat*-loop.

3. **Procedure, variable and reserved keyword names are not case sensitive:** The Java *Pattern* class (*Pattern.java*) is used by *RoboMind* to match string keywords in the program code to the syntax of programming concept keywords when the program is parsed. The adaptation to make the program code case insensitive has been implemented by including the *Pattern.CASE_INSENSITIVE* flag when the keyword string is compiled to an instance of the *Pattern* class.

The syntax of supported looping commands, namely *for*-loops and *while*-loops, have been adapted. The *RoboMind* compiler has also been adapted to support *repeat*-loops and the use of variables. The adapted RoboMind code in Figure 6.2 is equivalent to the Delphi sample program in Listing 6.1.

Procedures The implementation of procedures in *RoboMind* is the same as the implementation of procedures in Delphi. No adaptations were thus required. The procedure is defined using the `procedure` keyword followed by the name of the procedure (line 1 of sample code in Figure 6.2). The procedure is called using the procedure name and arguments in brackets (line 11 of sample code in Figure 6.2).

Correct use of parameters No changes have been made to the implementation of parameters in *RoboMind*. *RoboMind* does not support the specification of data types when the parameter variables are defined in the procedure declaration (line 1 of sample code in Figure 6.2). *RoboMind* only supports integer data types as procedure arguments. The compiler would have been adapted to include data types if procedures were required to pass more than one type of data type as parameters.



```
procedure moveSquare(size)
begin
  for1to(4)
  begin
    forward(size);
    right();
  end;
end;

var steps := 3;
moveSquare(steps);
```

Parameter, *size*, defined. No data type specified.

Parameter

Self-defined procedure

Variable declaration (without data type)

Procedure call with argument value *steps*

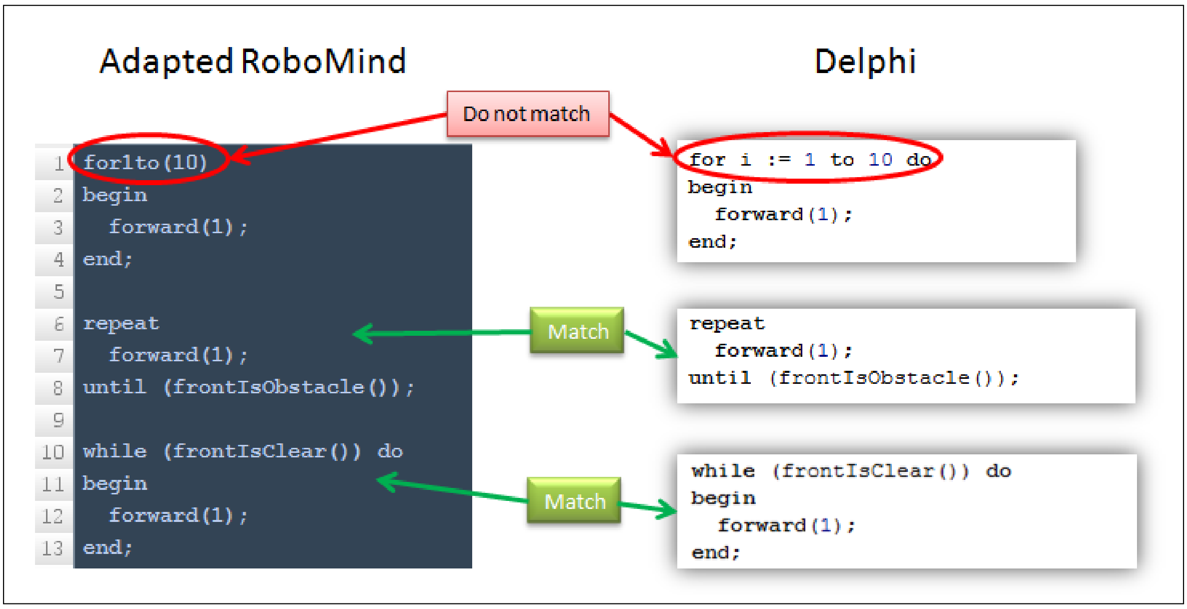
Figure 6.2: Adapted RoboMind: Procedures and use of parameters

***repeat*-loop** *RoboMind* supports the implementation of *while*-loops. The difference between the implementation of *repeat*-loops and *while*-loops in Delphi is that a *repeat*-loop is post-conditional and a *while*-loop is pre-conditional. The boolean condition of a *repeat*-loop must be true to stop execution of the loop while the boolean condition of a *while*-loop must be false to stop execution of the loop.

The code to compile a *while*-loop in *RoboMind* was used as a template for the implementation of the *repeat*-loop. The `repeat` and `until` keywords were defined in the English version of the Robo Language Definition file (*RLD_default_en.properties*) and the associated string patterns were initiated in the *RoboCompiler.java* file (*robo.script* package).

Two boolean methods – *tryParseBeginRepeatUntil()* and *tryParseEndRepeatUntil()* – were created in the *RoboCompiler.java* file (*robo.script* package) to parse the start and end of the *repeat*-loop, respectively, during the first compile level. An instance of the custom program command procedure, *BeginRepeatUntilCommand* is added to the list of program commands when the start of the *repeat*-loop is parsed. The end of the *repeat*-loop is specified in the list of program commands by adding an instance of the custom program command procedure, *EndRepeatUntilCommand* to the list. *EndRepeatUntilCommand* keeps a reference to the associated *BeginRepeatUntilCommand* object.

The second compile level sets the index of the first code statement that must be executed in the loop after the *repeat*-loop condition is evaluated. The code execution jumps back to this index in the list after evaluating the boolean loop condition to false. If the boolean loop condition is evaluated to true, the next program command in the command list, is executed. The *repeat*-loop in the adapted *RoboMind* is thus post-conditional and the loop continues if the condition is false, the same as in Delphi. The syntax of the *repeat*-loop also matches the Delphi syntax (Figure 6.3).



```
for1to(10)
begin
  forward(1);
end;

repeat
  forward(1);
until (frontIsObstacle());

while (frontIsClear()) do
begin
  forward(1);
end;
```

```
for i := 1 to 10 do
begin
  forward(1);
end;
```

```
repeat
  forward(1);
until (frontIsObstacle());
```

```
while (frontIsClear()) do
begin
  forward(1);
end;
```

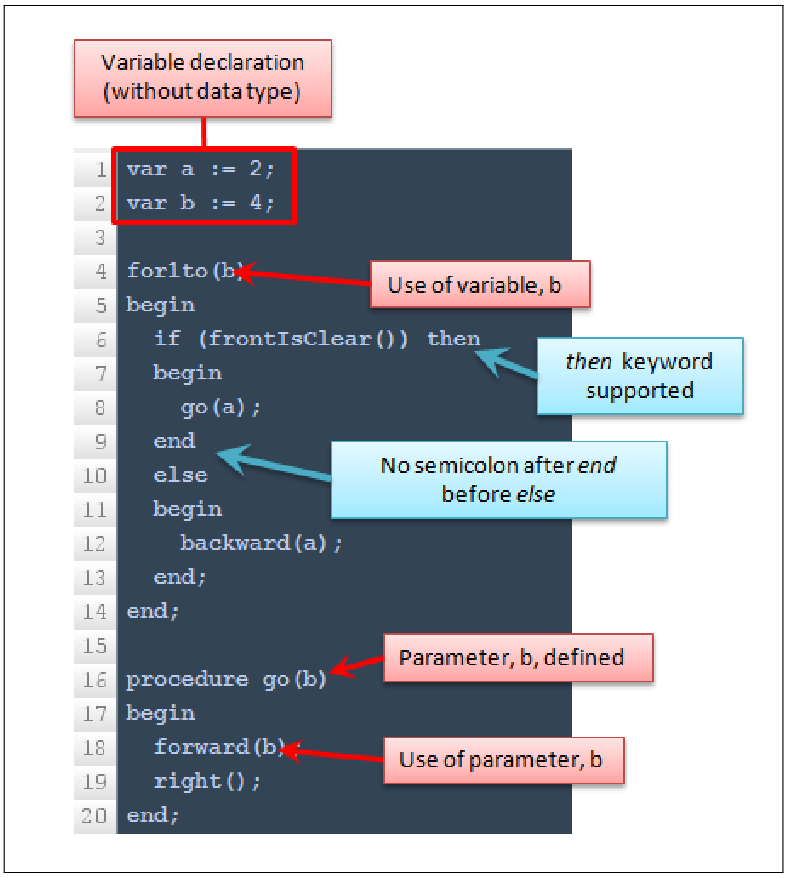
Figure 6.3: Adapted RoboMind: *for*-loop, *repeat*-loop and *while*-loop

***while*-loop** The `while` and `do` keywords have been defined in the English version of the Robo Language Definition file (*RLD_default_en.properties*) and initiated in the *RoboCompiler.java* file (*robo.script* package). The parsing code in the first compile level has been changed to parse the `do` keyword after the boolean loop condition. Figure 6.3 indicates the implementation of the *while*-loop in the adapted *RoboMind*.

***for*-loop** The `repeat(x)`command supported by the unadapted *RoboMind* is the equivalent of the *for*-loop where the loop counter starts from one and loops until the counter variable is greater than x. The loop thus iterates x times. The Delphi code equivalent is `for i := 1 to x do`, where `i` is the counter variable of integer type. The counter variable in the *RoboMind* implementation is not specified by the user, although a counter variable is used during compilation.

The adaptation implemented replaces the `repeat` keyword with the keyword `for1to` (Figure 6.3). The new keyword includes the *for* substring as well as indicates that the loop will iterate from the integer value one (1) to a specified ending value – an integer value or variable. The counter variable remains hidden and is not specified.

***if*-statements** The adapted *RoboMind* supports the `then` keyword to be present after the condition in the *if*-statement. The statement before the `else`keyword - the `end`



```
1  var a := 2;
2  var b := 4;
3
4  for1to(b)
5  begin
6    if (frontIsClear()) then
7    begin
8      go(a);
9    end
10   else
11   begin
12     backward(a);
13   end;
14 end;
15
16 procedure go(b)
17 begin
18   forward(b)
19   right();
20 end;
```

Variable declaration (without data type)

Use of variable, b

then keyword supported

No semicolon after *end* before *else*

Parameter, b, defined

Use of parameter, b

Figure 6.4: Adapted RoboMind: Variables and *if*-statements

keyword in the adapted *RoboMind* - should not be followed by a semicolon (line 9 of Figure 6.4). These two rules have been implemented in order to make the *RoboMind* programming language syntax consistent with the Delphi programming language syntax.

The code for parsing *if*-statements in the first compile level has been adapted to parse the `then` keyword after the boolean expression is evaluated. A semicolon is not parsed following the `end` before an `else` keyword.

Variables *RoboMind* has been adapted to support variables in the main procedure by allowing a variable to be created and assigned a value (lines 1 and 2 of Figure 6.4). No data types are indicated as only integer variables are accepted. Variables can be used as arguments for movement commands or self-defined procedures.

The implementation of parameters for self-defined procedures has been used as a template indicating the data structures and algorithms required to implement variables. An array list has been added to the *RoboMind* source code (*RoboCompiler.java*) to maintain variables and the values assigned to variables. Separate lists enable the compiler to differentiate between self-defined procedure parameters and variables within the scope of the main procedure (Figure 6.4).

In the Delphi programming language, all variables are declared before the begin statement for the program or procedure (Section 6.2). In the adapted *RoboMind*, variables can be declared anywhere in the code *before* the variable is used as an argument. The declaration of variables in the adapted *RoboMind* differs from Delphi because this was the best possible method of supporting variables at the time.

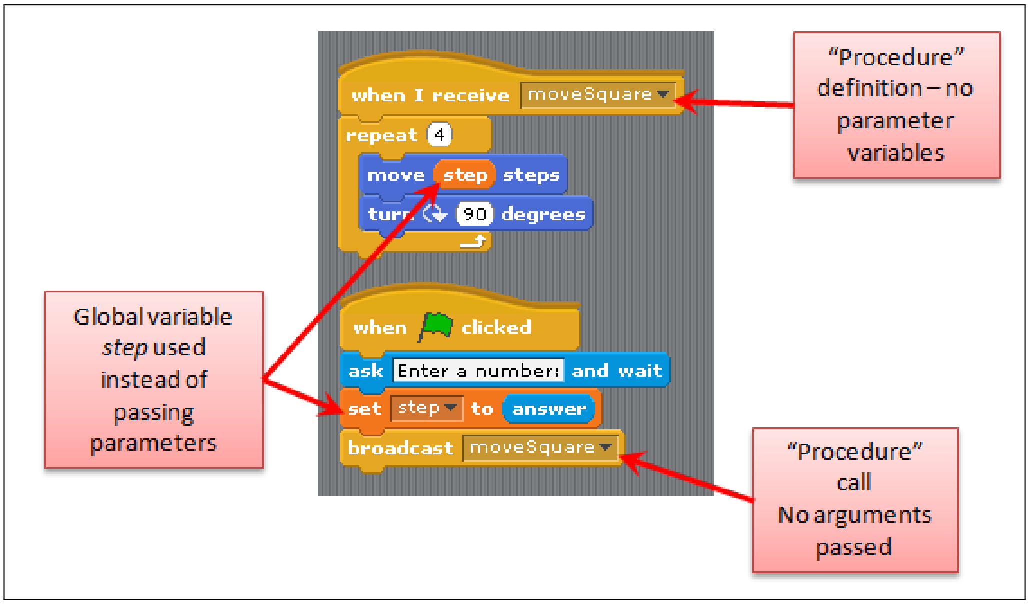
The possibility of implementing programming concepts not supported in the unadapted *RoboMind* is influenced by the robot microworld. The purpose of programs created in *RoboMind* are to control the robot in the microworld. Programming concepts that are not required for the robot's interactions with and movement in the microworld, are difficult to support in *RoboMind* as the programming output techniques and purpose of the robot in the microworld would need to be changed. The purpose of adaptation to the PATs is only focused on programming language syntax and supporting programming concepts that are already supported in some way, hence the support added for variables and *repeat*-loops in *RoboMind*.

6.2.3 Scratch

Scratch is a PAT that allows users to create programs using a drag-and-drop interface that does not enforce syntax rules (Section 5.3.4). Users, therefore, do not need to be concerned with the overhead of programming language syntax. More focus can be given to the planning of programming solutions. *Scratch* is not open-source software, thus no adaptations could be made to *Scratch*. The differences between the implementation of programming concepts in *Scratch* and Delphi are highlighted:

Procedures Multiple scripts can be created for different events that occur such as a key that is pressed or if one sprite touches another. Scripts can also be called when a name is *broadcast* (Figure 6.5 is the Scratch equivalent to the Delphi sample program in Listing 6.1). Broadcasting a specified name is the equivalent of a procedure call in Delphi. The *When I receive* block script is the equivalent of the procedure code.

Correct use of parameters Procedure parameters are not supported in *Scratch*. Global variables are used instead of passing parameter values to the procedure.



```
when I receive moveSquare
repeat 4
  move step steps
  turn ↻ 90 degrees

when ⚑ clicked
ask Enter a number: and wait
set step to answer
broadcast moveSquare
```

"Procedure" definition – no parameter variables

Global variable *step* used instead of passing parameters

"Procedure" call No arguments passed

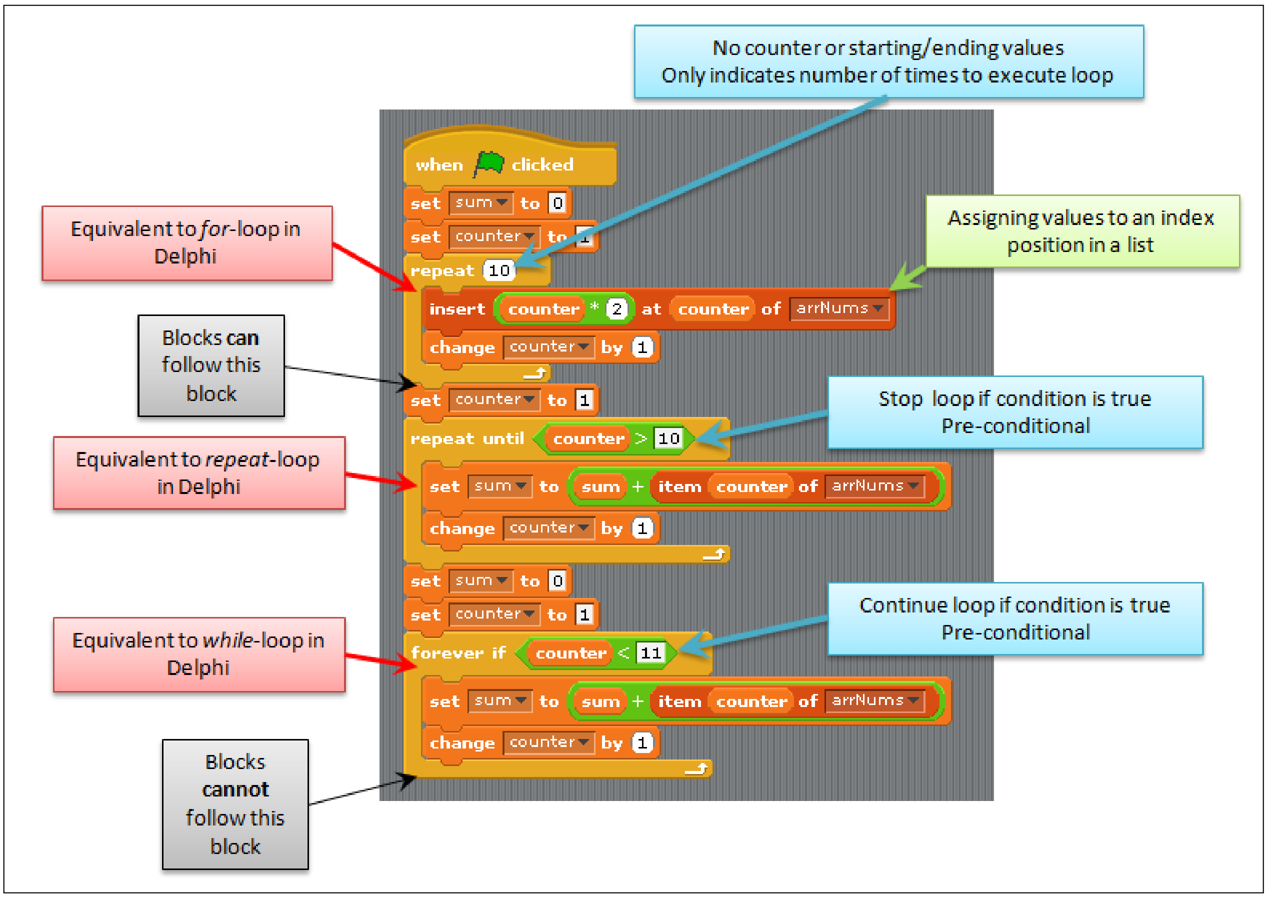
Figure 6.5: Scratch: Procedures

One-dimensional arrays Scratch supports the implementation of a list. A list is similar to a dynamic array in Delphi. The length of the list is not specified when the list is created. Values can be added to the list without specifying a position, in which case the value is appended at the end of the list. Values can also be inserted at specific positions in the list (Figure 6.6, which is the Scratch equivalent to the Delphi sample program in Listing 6.2), similar to assigning values to a static array at specific index positions.

***repeat*-loops** The Delphi *repeat*-loop equivalent in Scratch is the `repeat until` block (Figure 6.6). The loop terminates when the conditional statement is true, which is equivalent to the implementation of the *repeat*-loop in Delphi. However, in Delphi, the *repeat*-loop is post-conditional while in *Scratch* it is pre-conditional. The wording used on the block can also be associated to the syntax of the *repeat*-loop in Delphi.

***while*-loops** The `forever if` block in Scratch (Figure 6.6) is equivalent to the *while*-loop in Delphi. The loop continues while the conditional statement is true and the loop is pre-conditional, which is equivalent to the implementation of the *while*-loop in Delphi (Section 6.2.1). The wording on the block, however, does not correspond to the syntax of the *while*-loop in Delphi.

***for*-loops** The *Scratch* equivalent of the Delphi *for*-loop is the The `repeat` block (Figure 6.6). However, only the number of times the code in the block must be repeated is specified. The `repeat` block is thus equivalent to a Delphi *for*-loop in which the



```
when ⚑ clicked
set sum to 0
set counter to 1
repeat 10
    insert counter * 2 at counter of arrNums
    change counter by 1
set counter to 1
repeat until counter > 10
    set sum to sum + item counter of arrNums
    change counter by 1
set sum to 0
set counter to 1
forever if counter < 11
    set sum to sum + item counter of arrNums
    change counter by 1
```

No counter or starting/ending values
Only indicates number of times to execute loop

Equivalent to *for*-loop in Delphi

Assigning values to an index position in a list

Blocks can follow this block

Stop loop if condition is true
Pre-conditional

Equivalent to *repeat*-loop in Delphi

Continue loop if condition is true
Pre-conditional

Equivalent to *while*-loop in Delphi

Blocks **cannot** follow this block

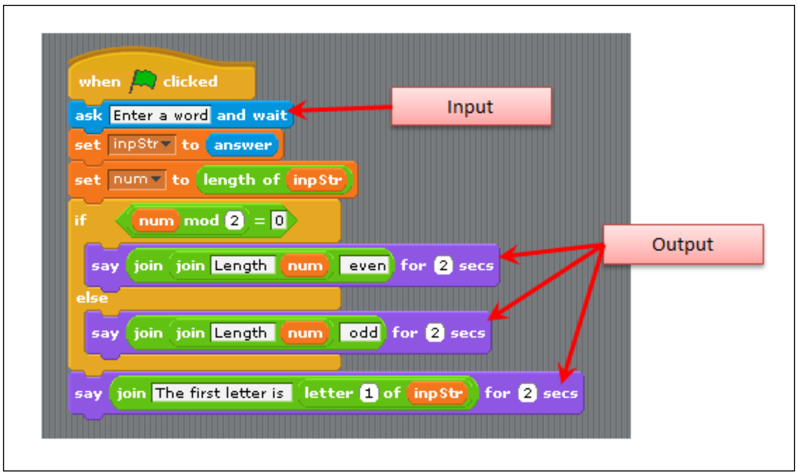
Figure 6.6: Scratch: Arrays and looping

counter variable iterates from the value one (1) to the integer value specified. No counter variable is defined in the *Scratch* `repeat` block. In addition, the wording used on the block is not consistent with the Delphi *for*-loop syntax.

String handling *Scratch* supports Delphi string operations (Figure 6.7, which is the Scratch equivalent to the Delphi sample program in Listing 6.3) such as joining words and values in a string, determining the length of a string and returning specific characters within a string. The wording used on the blocks differs from the syntax that is used in Delphi.

***if*-statements** *Scratch* supports *if*-statements (Figure 6.7). A simple `if` block as well as an `if else` block are available for use in program solutions. Nested *if*-statements are also supported in *Scratch*.

Variables Variables are supported although no data types are specified. Values are assigned to variables using the `set` block. The orange blocks in each of the three figures (Figure 6.5, 6.6 and 6.7) demonstrate the use of variables in *Scratch*.



```
when clicked
ask Enter a word and wait
set inpStr to answer
set num to length of inpStr
if num mod 2 = 0
    say join join Length num even for 2 secs
else
    say join join Length num odd for 2 secs
say join The first letter is letter 1 of inpStr for 2 secs
```

Figure 6.7: Scratch: String handling, *if*-statements, input and output

Input *Scratch* is able to execute scripts triggered by events such as mouse-clicks or key presses, as in Delphi. *Scratch* is also able to accept string input from the user using the `ask` block (Figure 6.7).

Output *Scratch* supports different types of program output. Sprites can move, change colours, draw pictures and play music. String output which is the most common form of output in Delphi, can also be displayed (Figure 6.7).

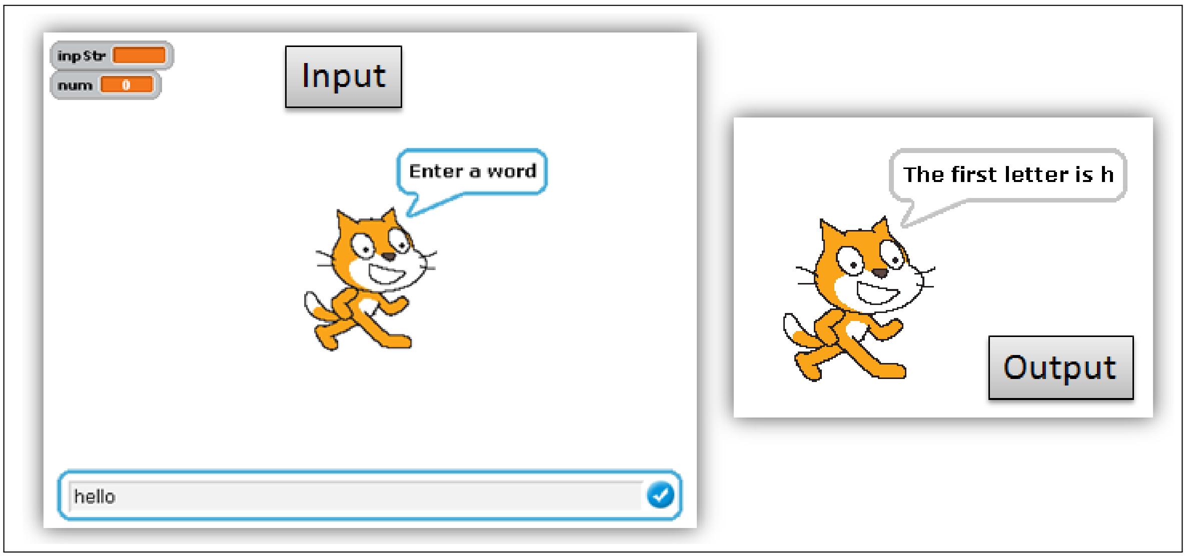


Figure 6.8: Sample program (Figure 6.7) executed in Scratch

Figure 6.8 is an example of the input and output screens that are displayed to users when the sample program in Figure 6.7 is executed. Any of the sprites can ask for user input that can be assigned to a variable and display speech bubbles to display messages to users as output.

6.2.4 B#

B# is an iconic PAT in which the user can create a flowchart to solve a particular programming problem by dragging icons from the toolbar. *B#* automatically generates program code for a Delphi *console application* from the flowchart created by the user.

The only programming concepts supported by *B#* that differ with regard to syntax from Delphi are input and output. *B#* interacts with users using a console application for the main execution, however, when tracing through the program GUI forms are used to accept user input. The Delphi programming environment can be used to create console applications as well as programs with a Graphical User Interface (GUI). The subject content outline for programming in the IT subject requires IT learners to create programs that interact with users (input and output) using a GUI. IT learners programming in Delphi are not required to and may not be explicitly taught to use a console line application interface.

Input In *B#* input is obtained from the user by displaying a message instructing the user what to enter (Figure 6.10) and then assigning the value entered by the user to a variable. The instruction to enter data is achieved by using the `writeln` command (Table 6.3). The program then waits for the user to enter a number and press *Enter*. The value entered by the user is assigned to a variable using the `readln` procedure (Table 6.3). No string to integer conversion is necessary if the user input is assigned to an integer variable.

Table 6.3: Comparison of statements to read in input from the user in Delphi and B#

Delphi	**B#**

`num := strToInt(edtNum.text);`	`writeln('Enter a number');`
	`readln(num);`

In Delphi the user types input into a graphical component such as an edit box. The Delphi code in Table 6.3 demonstrates how the text value in an edit box (`edtNum`) is converted to an integer value and is assigned to the integer variable `num`. The string to integer conversion is achieved using the `strToInt` function.

Output There are differences in the manner in which output would be displayed to a user in Delphi and *B#*. Delphi uses graphical components such as a label, memo box or rich edit to display string output. Table 6.4 shows the Delphi code used to display a message indicating the value entered by the user. In the Delphi example code, the output is displayed using a graphical label component (`lblOutput`) on a form. *B#* uses the `writeln` procedure to display a string in the command line console.

Table 6.4: Comparison of statements to display output to the user in Delphi and B#

Delphi	**B#**
`lblOutput.caption := 'The user entered' + intToStr(num);`	`writeln('The user entered', num);`

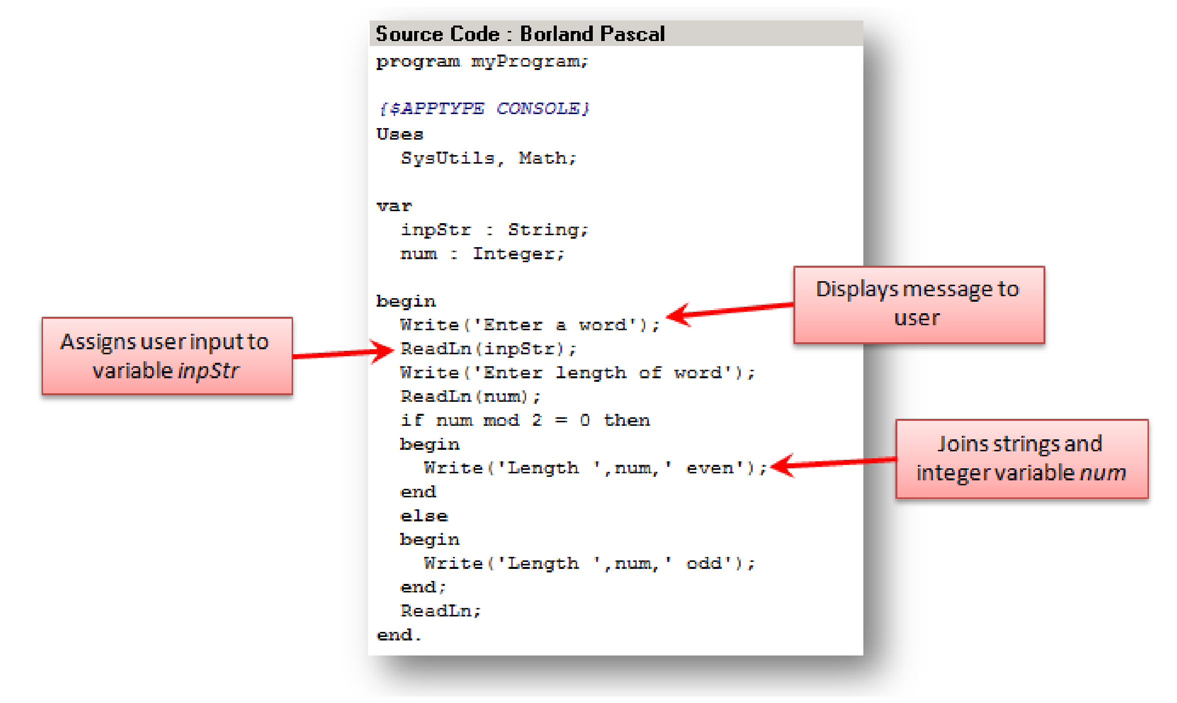
There is a difference in the manner in which the string output is constructed in Delphi and *B#*. In Delphi, the plus sign (+) is used to concatenate two string types. An integer to string conversion (`intToStr`) is necessary in Delphi if the value of an integer variable must be displayed (Table 6.4). However, in *B#* the `writeln` procedure accepts multiple arguments where different data types are separated by commas. The `writeln` procedure handles the concatenation of the arguments into a string to be displayed in the console application. For example, in Table 6.4 the string and integer variables are delimited using a comma and the integer variable, `num` is not converted to a string value.

An alternative would be first to assign the output string to a string variable and then only pass the string variable as the `writeln` argument (Listing 6.4). The string and number values can be concatenated using the plus sign (+) and an integer to string conversion is required for integer values using `intToStr` (`floatToStr` required to convert real numbers to a string). However, *B#* does not support string handling therefore strings cannot be concatenated using the plus sign. The `intToStr` function is also not recognised by *B#*.

Listing 6.4: String handling for console application

```
var s : string;
    num : integer;
begin
  s := 'The user entered' + intToStr(num);
  writeln(s);
end;
```

B# automatically generates program code based on the flowchart created by users. Figure 6.9 is an example of the program code generated in *B#* that corresponds to the Delphi sample program in Listing 6.3. The console application corresponding to the *B#* program



Source Code : Borland Pascal

```
program myProgram;

{$APPTYPE CONSOLE}
Uses
  SysUtils, Math;

var
  inpStr : String;
  num : Integer;

begin
  Write('Enter a word');
  ReadLn(inpStr);
  Write('Enter length of word');
  ReadLn(num);
  if num mod 2 = 0 then
  begin
    Write('Length ',num,' even');
  end
  else
  begin
    Write('Length ',num,' odd');
  end;
  ReadLn;
end.
```

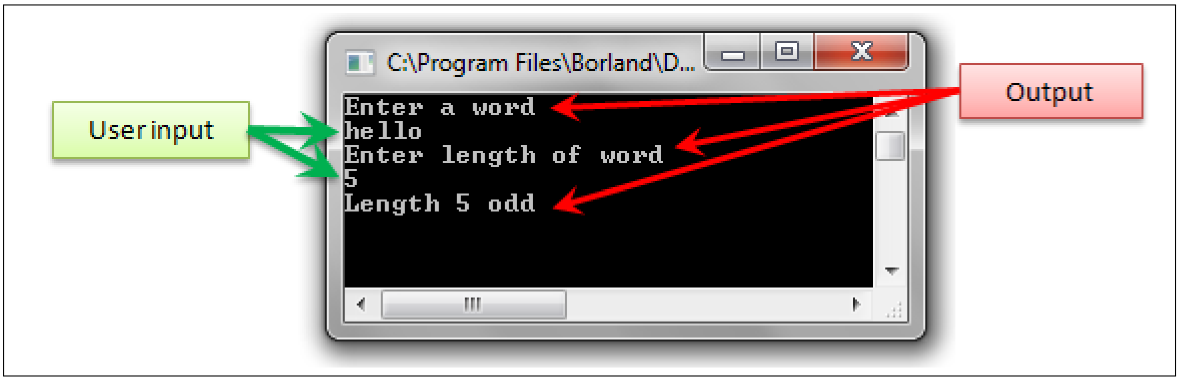
Assigns user input to variable *inpStr*

Displays message to user

Joins strings and integer variable *num*

Figure 6.9: B#: Input and output

code, is shown in Figure 6.10. The code and console application are the *B#* equivalents to the Delphi sample code listing in Listing 6.3 and the Delphi GUI form in Figure 6.1, respectively.



```
Enter a word
hello
Enter length of word
5
Length 5 odd
```

User input

Output

Figure 6.10: Console application created using B#

The main difference of the *B#* tool is that input and output of data is executed using a console application (Figure 6.10). IT learners using Delphi write programs that receive input and display output using graphical user interface components such as edit boxes and memo boxes (Figure 6.1).

The only adaptations that would be considered in *B#* is to add basic string handling and allow *B#* to recognise the `intToStr` function. This would allow the output to users to be handled similarly to Delphi. However, adaptations to *B#* have not been possible as the source code available is incomplete and does not allow the full project file to be opened or compiled.

6.3 Conclusion

The adaptations to the *RoboMind* tool have been implemented and supplementary support has been provided to support the understanding of programming concepts using the PATs. This is in order to address RQ4: *How can the selected PATs be adapted for use by IT learners in SA secondary schools to support the understanding of programming concepts?* and address Secondary Objective 1.2: *Adapt selected PATs to make them suitable to support the achievement of programming learning outcomes in the IT subject curriculum implemented in South African secondary schools.*

Table 6.5: Support of programming concepts: Adapted RoboMind, Scratch and B#

Programming Concept	**RoboMind**	**Scratch**	**B#**
Procedures	✓	⋆	
Functions			
One-dimensional arrays		⋆	
repeat..until loops	✓	⋆	✓
Objects & classes		⋆	
Two-dimensional arrays			
while..do loops	✓	⋆	✓
String handling		⋆	
for-loops	⋆	⋆	✓
Correct use of parameters	⋆		
if-statements	✓	⋆	✓
SQL statements			
File handling			
Accessing database			
case (Delphi)/*switch* (Java) statements			✓
Output		⋆	⋆
Variables	⋆	⋆	✓
Input		⋆	⋆

✓= Concept supported and syntax matches Delphi

⋆ = Concept supported but syntax does not match Delphi

blank = PAT does not support the concept

The three PATs - *RoboMind*, *Scratch* and *B#* - have been identified as the most suitable for IT learners using Delphi as the programming language based on an evaluation using the formulated selection criteria (Section 5.5). *RoboMind* has been adapted to support the Delphi syntax of programming concepts and support has been provided for variables and *repeat*-loops. Table 6.5 provides an indication of the programming concepts supported by the three PATs that are evaluated by the participating IT learners.

In addition to comparing the syntax of programming concepts implemented in Delphi and the three PATs, other properties and shortcomings of the PATs are identified. *RoboMind* is open-source which allows the programming language used to program the robot to be adapted. It can therefore be made the same or similar to the programming language in which the learner is learning to program. However, adaptations to include programming concepts not supported by *RoboMind* are influenced by the microworld environment with which the robot interacts. Any programming concepts implemented should have a purpose for the robot. For example, support for string handling would not be necessary until a reason for the robot to use strings is implemented such as interaction with a user or “reading” instructions.

Scratch cannot be adapted. The programming concept keywords used are general and not specific to a particular high-level programming language. Scratch can thus be used when learning any high-level language. The keywords used are also simple enough for learners to transfer to programming concepts being learnt in a high-level programming language, such as Java or Delphi. There are differences in the manner in which the looping structures are implemented. This may cause confusion for IT learners trying to find the equivalent looping structure in Delphi.

B# has also not been adapted due to difficulties opening the full project file in Delphi. IT learners using Delphi do not use a console application at any stage. However, many PATs use console applications to accept input from and display output to users, as do the IT learners using the Java programming language (Section 1.5).

The next phase of this research is the evaluation of the impact of the three PATs on IT learner understanding of programming concepts (Chapter 7) and motivation towards programming (Chapter 8). The evaluation is a comparison of the control and treatment group responses from participating learners.

Chapter 7

The Impact of PATs on the Understanding of Programming Concepts

7.1 Introduction

Three PATs - *RoboMind*, *Scratch* and *B#* - have been selected for participating treatment group learners to evaluate at the four participating schools. This chapter together with Chapter 8, addresses Primary Objective 2: *To evaluate the impact of the selected programming assistance tools (PATs) on a novice programmer's understanding of programming concepts included in the IT subject curriculum implemented in a case study of South African secondary schools.*

The aim of this chapter is specifically to address Secondary Objective 2.1: *Evaluate the impact of the proposed PATs on IT learner understanding of programming concepts.* This is achieved by comparing control and treatment group results in terms of perceived difficulty of IT (Grade 11). Multiple choice class test scores and end-of-year summative assessment marks for the IT subject are also analysed to assess IT learner understanding of programming concepts. Self-reported evaluations of the PATs by treatment group learners provide feedback from IT learners related to the usefulness of the PAT with respect to the understanding of specific programming concepts, as well as learner comments on what they preferred or did not prefer about the PAT.

The questionnaire and multiple choice class test results are used to address RQ5: *What impact do the different PATs have on IT learner understanding of programming concepts?* In addressing RQ5, the research hypothesis H_1 (Section 2.3) is tested to determine if the null hypothesis $H_{1,0}$, which states that there is no difference between the assessment means of the control and treatment groups (μ_1=μ_2, μ_1 is the control group assessment mean, μ_2 is the treatment group assessment mean), is rejected. If $H_{1,0}$ is rejected, the alternative hypothesis, $H_{1,1}$: $\mu_1 \neq \mu_2$, is accepted. The impact of the techniques used by PATs on learner understanding of programming concepts is also evaluated in order to address RQ7: *What techniques should PATs implement in order to assist IT learner understanding of programming concepts?*

The application of the experimental procedure is explained together with a presentation for research sample sizes, overall results and difficulties experienced (Section 7.2). The results for each of the PATs provided to the four participating schools, namely *RoboMind* (Section 7.3.1), *Scratch* (Section 7.3.2) and *B#* (Section 7.3.3), are presented. Finally, the impact of each of the PATs on IT learner understanding of programming concepts, to address RQ5 and RQ7, as well as whether the null hypothesis, $H_{1,0}$, is rejected based on the results, are evaluated in the conclusion (Section 7.4).

7.2 Application of Experimental Procedure

A breakdown of the number of learners participating in the research in the control and treatment groups (Section 7.2.1) and a description of the supporting documentation provided to treatment group learners when the PAT was administered (Section 7.2.2), are presented. The data collection methods are discussed (Section 7.2.3), followed by overall inferential analysis results comparing the entire control and treatment group samples (Section 7.2.4), regardless of PAT. The difficulties experienced during the experimental study, resulting in incomplete data, are highlighted (Section 7.2.5) before the results for the different PATs are presented (Section 7.3).

7.2.1 Research Sample

The participants for this research study consisted of Grade 10 and Grade 11 IT subject learners at four consenting schools in the Port Elizabeth area. The sample size of participants in each school-grade group was affected by the class size and willingness of individual learners to participate.

Learners in the Grade 10 and Grade 11 IT subject class at each school in the first year form part of the control group (Table 7.1). Learners in the Grade 10 and Grade 11 IT subject class at each school in the second year all received the PAT. However, use of the PAT was not enforced, thus consenting learners who chose not to use the PAT ultimately form part of the control group, while learners that indicated that they installed and used the PAT form the treatment group.

Table 7.1: Number of participants in control and treatment groups. Two schools received Scratch and are identified as Scratch$_1$ and Scratch$_2$.

	Control			Treatment		
PAT	**Grade 10**	**Grade 11**	**Total**	**Grade 10**	**Grade 11**	**Total**
RoboMind	25	27	52	24	4	28
Scratch$_1$	24	15	39	5	3	8
Scratch$_2$	7	5	12	14	6	19
B#	17	12	29	18	11	29
Total	73	59	132	60	24	84

Two schools received *Scratch* (Table 7.1). These were the schools with the smallest number of learners consenting to participate in the second year of the study. *Scratch* was the PAT selected to provide to two schools as it received the highest programming concept ranking score of the PATs selected (Section 5.4.5). *Scratch* has also been recommended in the new IT subject curriculum to teach programming to Grade 10 IT learners (Department of Basic Education, 2011).

7.2.2 Supplementary Support

A brief presentation demonstrating installation and use of the PATs was provided to treatment group learners when they received the PATs. Supporting documentation (Appendix G) accompanied each of the PATs, including a brief description of the PAT together with instructions on how to install the PAT. Details of how to use the PAT and to find example programs are included in the supporting documentation. The supporting documentation for each of the three PATs also includes a screenshot for each PAT accompanied by a brief description of the interface and guidelines for using the PAT.

Each of the three PATs included supplementary support in the form of help files, online help (website) and/or example programs to illustrate use of the PAT. The *Scratch* help files, included with the installation, provide a detailed description of how to use *Scratch.* The help topics available within the *RoboMind* environment explain the robot commands that can be used and provide four example programs. Tutorial help files explaining how to use *B#* are included with the *B#* installation but the help files are not accessible from within the *B#* programming tool interface.

In addition to the help files, *RoboMind* and *Scratch* include example programs when installed. *Scratch* provides users with access to a large collection of example programs which are categorised into subfolders such as animation, games, greetings, interactive art, simulations and stories. The example programs provided with the *Scratch* installation are complex, interactive scripts, in most cases using multiple sprite objects. Novice programmers may find these programs difficult to understand when first learning to use *Scratch*.

A folder containing example programs, compiled specifically for this research study, has been included with each of the three PATs provided to IT learners. The aim of the example programs is to demonstrate how programming concepts included in the IT curriculum can be implemented in the PAT. In particular, the example programs included with *Scratch* provide simple programs to assist users to start using *Scratch* as well as to demonstrate how different programming concepts, such as arrays, looping and conditional statements, can be implemented. In *RoboMind*, example programs, included with the original version, have been rewritten in the adapted *RoboMind* programming syntax. Additional exercises have also been included to demonstrate the different programming concepts that can be implemented in *RoboMind*. Similarly, the *B#* example programs folder provided for IT learners includes exercises that demonstrate how different programming concepts supported by *B#* can be implemented.

7.2.3 Data Collection, Analysis and Presentation

The results presented are categorised according to PAT received and experimental group (control/treatment). The following notation is used to identify a particular group:

$$<\!\mathit{ExperimentalGroup}\!>_{(PAT,Grade)}$$

where ExperimentalGroup ∈ {Control, Treatment}, PAT ∈ {Robo, Scratch$_1$, Scratch$_2$, BSharp} and Grade ∈ {10,11,10&11}. For example, $\mathit{Treatment}_{(BSharp,10)}$ refers to the Grade 10 treatment group evaluating *B#*, while $\mathit{Control}_{(Robo,10\&11)}$ refers to the entire control group corresponding to the *RoboMind* treatment group.

Grade 11 learners in the control and treatment groups completed the *Perceived Difficulty of Programming Questionnaire* (Section 2.5.1, Appendix C) at the beginning (pretest) and towards the end (posttest) of the year (Figure 7.1). The *Perceived Difficulty of Programming Questionnaire* evaluates IT learners' perceptions of the difficulty of specific programming concepts as well as learners' perceived ability to understand and apply programming knowledge and skills.

Multiple choice class tests are used to assess IT learner understanding of specific programming concepts (Section 2.5.2, Appendix D). The tests are administered to the entire

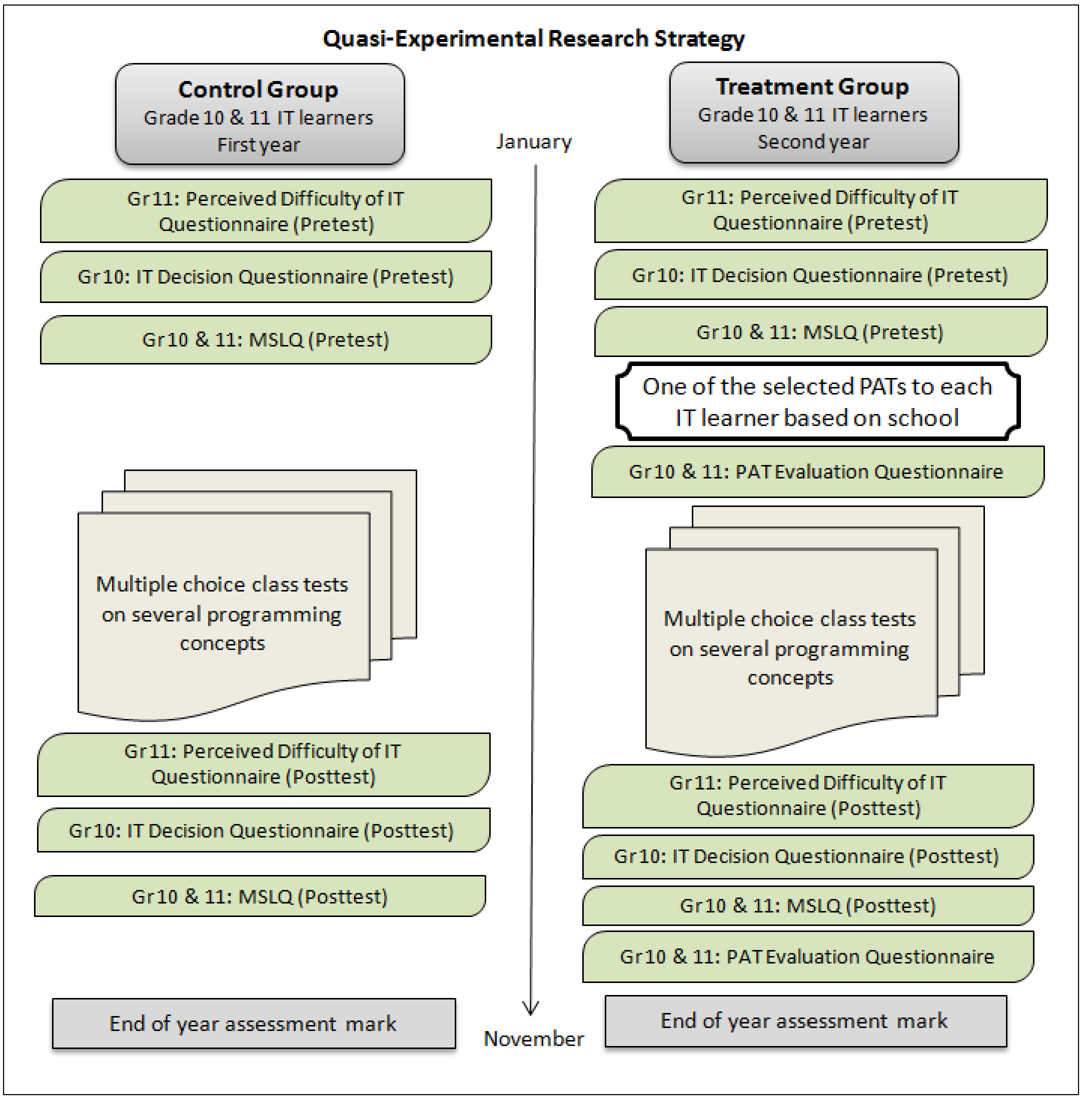

Quasi-Experimental Research Strategy
Control Group
Grade 10 & 11 IT learners
First year
Treatment Group
Grade 10 & 11 IT learners
Second year
January
Gr11: Perceived Difficulty of IT Questionnaire (Pretest)
Gr10: IT Decision Questionnaire (Pretest)
Gr10 & 11: MSLQ (Pretest)
One of the selected PATs to each IT learner based on school
Gr10 & 11: PAT Evaluation Questionnaire
Multiple choice class tests on several programming concepts
Gr11: Perceived Difficulty of IT Questionnaire (Posttest)
Gr10: IT Decision Questionnaire (Posttest)
Gr10 & 11: MSLQ (Posttest)
Gr10 & 11: PAT Evaluation Questionnaire
End of year assessment mark
November

Figure 7.1: Pre- and posttests administered during the quasi-experimental approach

IT class (including non-participating learners) by the IT teacher as a class test after the programming concept has been taught to the learners. IT classes of both academic years at all the schools wrote the same tests.

Table 7.2: Multiple choice class tests used to evaluate IT learner understanding of programming concepts

	Grade 10		Grade 11	
Programming Concept	**Questions**	**Total Mark**	**Questions**	**Total Mark**
if-statements	12	25		
for-loops	8	16		
repeat and *while*-loops	12	24		
One-dimensional arrays			10	20
Procedures and functions			10	20

Five multiple choice tests (Appendix D) have been administered to IT learners to assess their knowledge of specific programming concepts (Table 7.2). All the concepts assessed are identified as difficult concepts from the study of related literature (Section 3.4). All of the programming concepts evaluated, using multiple choice tests, except for *if*-statements, are in the top ten most difficult programming concepts with which IT learners need assistance (Section 4.4).

The Grade 10 multiple choice tests evaluate programming concepts that are supported by all three of the selected PATs (Table 6.1). The questions in the *repeat*- and *while*-loops test are separated to report results on *repeat*-loops and results on *while*-loops. No questions in the *repeat*- and *while*-loops test use a combination of the two loops thus the separation of results is possible.

In addition to the multiple choice class tests, the end-of-year summative assessment marks achieved by participating Grade 11 IT learners were also used to evaluate the impact of the PATs on IT learner understanding of programming concepts. No pretest score was available for Grade 10 learners as IT is not offered in Grade 9.

The *PAT Evaluation Questionnaire* (Section 2.5.2, Appendix E) was administered to treatment group IT learners to evaluate their use of the PAT they received. Only the items to determine the participant's perceived usefulness of the PAT for the understanding of specific programming concepts are reported in this chapter. The remaining items in the questionnaire relate to the participant's motivation towards using the PAT (Chapter 8). Grade 10 and Grade 11 feedback from the *PAT Evaluation Questionnaire* are combined as the sample sizes for individual grades were too small for the evaluation of certain of the PATs.

Two-way analysis of variance (ANOVA) and analysis of covariance (ANCOVA) inferential statistics are used to examine the effect of independent variables as well as the interaction between independent variables (Cohen *et al.*, 2007), respectively. The independent variables used are experimental group – control or treatment – and PAT. The dependent continuous variables evaluated are multiple choice class test, final IT subject assessment mark and *Perceived Difficulty of Programming Questionnaire* results. Cohen's d is calculated to measure practical significance only for statistically significant results.

7.2.4 Overall Results

Two-way ANOVA results of the multiple choice class tests indicate that there are no statistically significant results (Table 7.3) for the main effect experimental group or the interaction effect (PAT*experimental group), indicating that the use of the PATs did not result in a statistically significant difference in the class test marks of those learners who used the PAT (treatment group) and those who did not (control group). There was also no overall statistically significant difference between the control and treatment group learners, regardless of which PAT was used.

Two-way ANCOVA analysis was used to determine a statistically significant difference between the control and treatment group final Grade 11 IT assessment mark. The Grade 10 final marks were used as the covariate for the analysis. The analysis indicated that there is no statistically significant result (Table 7.3) for the main effect experimental group or the interaction effect (PAT*experimental group) on the final IT assessment mark for Grade 11 learners.

Table 7.3: ANOVA: Impact of PAT and experimental group on multiple choice test scores and final Grade 11 IT mark

		if	*for*	*repeat*	*while*	**Procedures**	**Final Mark**
		df=1;89	df=1;122	df=1;128	df=1;128	df=1;44	df=1;53
Experimental group	F	1.735	1.732	0.01	0.62	1.27	0.26
	p	0.4001	0.191	0.914	0.433	0.265	0.613
PAT*experimental group	F	1.920	1.053	2.24	2.35	1.97	1.22
	p	0.153	0.372	0.087	0.075	0.167	0.305

7.2.5 Difficulties Experienced

The completeness and reliability of the results reported in this chapter and the chapter hereafter have been impacted by the research methods implemented. Difficulties with regards to the data collection process have resulted.

Questionnaires and class tests were administered to learners by the IT teacher during the course of the year. The IT teacher's work schedule also had to be taken into consideration and the research placed an additional burden on IT teachers. There was no direct control over the administration of class tests and questionnaires to ensure that the planned research methods were followed. In certain cases questionnaires were not administered as planned and some multiple choice class test marks were not obtained due to administrative issues.

The IT teacher was changed, during the treatment year, at one of the schools using *Scratch*. The results of the two *Scratch* schools are thus reported separately as the change in the teaching environment could influence results evaluating the impact of the PAT.

Another difficulty experienced with regards to the analysis was as a result of the small treatment group sample sizes. Not all participants in the treatment group used the PAT received and learners did not complete all questionnaires. This is a useful result for the research, however, as small sample sizes (n<10) reduce the reliability of the data analysis. Missing data is brought to the reader's attention in the appropriate places of analysis.

7.3 Evaluation of PATs

The reported results are presented according to the PATs provided to participating learners, namely *RoboMind* (Section 7.3.1), *Scratch* (Section 7.3.2) and *B#* (Section 7.3.3). The *Scratch* feedback is separated into the feedback from the two schools evaluating *Scratch* and identified as *Scratch*$_1$ and *Scratch*$_2$. Grade 10 and Grade 11 questionnaire and multiple choice test results are provided for each PAT. The results aim to provide feedback on the impact of each PAT on IT learner understanding of programming concepts.

7.3.1 RoboMind

The majority of the learners in Grade 10 (87%, n=24) who received *RoboMind* utilised the PAT. In contrast, only 29% (n=4) of the Grade 11 learners who received *RoboMind* used the PAT. Grade 11 learners indicated that they did not need to use a PAT as they were confident in their understanding of Delphi, did not have enough time to use *RoboMind* or found *RoboMind* too complicated to use.

The results from the different questionnaire feedback and class test results are presented based on the research question the results address. The results that evaluate the impact of *RoboMind* on IT learners' understanding of programming concepts (Section 7.3.1.1) address RQ5. The results that provide feedback regarding the techniques used by the *RoboMind* (Section 7.3.1.2) to support IT learner understanding of programming concepts, address RQ7.

7.3.1.1 Understanding of Programming Concepts

The impact of *RoboMind* on the understanding of specific programming concepts is evaluated using three instruments (Table 7.4), namely, the *Perceived Difficulty of Program-*

Table 7.4: Adapted RoboMind support of programming concepts results

	PDPQ			Multiple Choice Tests			PAT Eval			
Programming Concept							Eval1		Eval2	
	μ_1	μ_2	**ANCOVA**	μ_1	μ_2	**ANOVA**	μ_2	**T-stat**	μ_2	**T-stat**
Procedures	2.56 (1.72) n=27	2.5 (0.58) n=4	F(1;16)=0.183 p=0.674	88.15 (13.31) n=27	90.00 (11.55) n=4	F(1;29)=0.069 p=0.794	2.50 (1.64) n=6	t(5)=-0.745 p=0.49	2.00 (1.85) n=8	t(7)=-1.528 p=0.170
repeat-loops	2.22 (1.34) n=27	1.00 (0.00) n=4	F(1;27)=6.49 **p=0.017***	70.00 (22.04) n=22	74.81 (20.45) n=27	F(1;47)=0.63 p=0.433	2.43 (1.63) n=7	t(6)=-0.93 p=0.386	2.71 (1.85) n=17	t(16)=-0.96 p=0.170
while-loops	2.04 (1.19) n=27	1.00 (0.00) n=4	F(1;27)=5.54 **p=0.026***	77.92 (23.63) n=22	84.13 (12.74) n=27	F(1;47)=1.38 p=0.247	2.00 (1.62) n=6	t(5)=-1.94 p=0.386	2.65 (1.27) n=17	t(16)=-1.14 p=0.269
for-loops	1.63 (0.84) n=27	1.00 (0.00) n=4	F(1;27)=3.49 p=0.073	79.09 (18.8) n=22	82.15 (15.21) n=27	F(1;47)=0.40 p=0.532	2.38 (1.69) n=8	t(7)=-1.05 p=0.329	2.71 (1.21) n=17	t(16)=-1.00 p=0.332
Correct use of parameters	2.52 (1.72) n=27	3.50 (2.38) n=4	F(1;22)=1.87 p=0.186							
if-statements	1.44 (0.85) n=27	1.00 (0.00) n=4	F(1;27)=2.76 p=0.108	73.82 (14.75) n=22	79.56 (16.62) n=27	F(1;47)=1.60 p=0.213	2.96 (1.01) n=25	t(24)=-0.17 p=0.870	3.00 (1.30) n=21	t(20)=-0.00 p=1.00
Variables	1.19 (0.40) n=27	1.00 (0.00) n=4	F(1;27)=1.65 p=0.210				2.88 (1.12) n=24	t(23)=-0.55 p=0.588	2.70 (1.17) n=20	t(19)=-1.14 p=0.267

* p<0.05

ming Questionnaire (PDPQ) (Appendix C), multiple choice class tests (Appendix D) and *PAT Evaluation Questionnaire* for *RoboMind* (Appendix E). The specific programming concepts evaluated are the concepts supported by *RoboMind* (Section 6.2.2). The questionnaire results for the *Perceived Difficulty of Programming Questionnaire* and the *PAT Evaluation Questionnaire* are self-reported metrics, while the multiple choice class tests are tests to assess learner knowledge of the programming concepts. The multiple choice tests are standard for all participants. One sample t-tests are used to test the evaluation mean against the neutral value of 3 for the PAT Evaluation Questionnaire results. Higher evaluation values indicate learners believe the PAT assisted. Standard deviation values are indicated in brackets below the mean values.

A comparison of the mean scores of the control and treatment group posttest results for the *Perceived Difficulty of Programming Questionnaire* indicate that learners in the treatment group perceived the programming concepts to be easier than the control group did, except for the use of parameters (Table 7.4). The differences between the control and treatment group means for the *repeat*- (F(1;27)=6.49, p=0.017) and *while*-loop concepts (F(1;27)=5.54, p=0.026) are statistically significant.

The multiple choice class test results (Table 7.4) indicate that the treatment group test means are higher than the control group means for all of the programming concepts evaluated. No class tests specifically evaluated learner understanding of the use of parameters or variables. One-way ANOVA analysis showed no statistically significant differences between the class test means. The array test results for Grade 11 learners are not included as *RoboMind* does not support arrays and the array test was administered to learners in the treatment group before *RoboMind* was received.

$Treatment_{(Robo,10\&11)}$ group respondents evaluated the usefulness of *RoboMind* twice in the academic year (Section 7.2.3). The usefulness of *RoboMind* in terms of assisting with the understanding of the programming concepts supported was evaluated using a semantic differential scale where a rating of 1 indicates that learners strongly disagreed and a rating of 5 indicates that learners strongly agreed that *RoboMind* assisted their understanding of the concept. The mean evaluation ratings (Table 7.4) for all of the concepts are less than, or equal to the neutral value, 3. One sample t-tests indicate that none of the programming concept mean values are statistically significantly different from the neutral value. Both evaluation results indicate the highest mean ratings for the support provided by *RoboMind* for the understanding of *if*-statements (μ_2=2.96 for the first and μ_2=3.00 for the second evaluation).

In summary, the results indicate statistically significant mean score differences between the control and treatment group for *repeat*- and *while*-loops in terms of learner perceptions of difficulty. The treatment group mean class test mark for these two concepts is higher than the control group mean, although not statistically significant. The results from all three instruments indicate support for the understanding of *if*-statements. For each of these concepts, *RoboMind* supports the correct syntax and implementation (Section 6.2.2).

Treatment$_{(\mathit{Robo},10\&11)}$ group respondents (n=10) provided qualitative feedback about what they liked and disliked about *RoboMind* (*PAT Evaluation Questionnaire*, Appendix E). Ninety percent of respondents (n=9) indicated that *RoboMind* helped to practise basic programming concepts. Forty percent of respondents (n=4) indicated that they disliked that the syntax of certain of the programming concepts supported in *RoboMind* (Section 6.2.2) are slightly different to coding in Delphi. Forty percent of the responses (n=4) indicated dislike for the fact that *RoboMind* supports a limited number of programming concepts.

7.3.1.2 Evaluation of Techniques

RoboMind represents the program solution output as a robot moving around a map world (Table 7.5). The program solution is text code displayed next to the microworld animation during program execution.

Table 7.5: RoboMind techniques

Representation of the program solution	
	Text
Construction of programs	
	Typing of code
	Selecting/form filling
Resulting program	
	Microworld
Support to understand programs	
	Debugging
Preventing syntax errors	
	Selection from valid options
	Informative syntax error messages

Treatment$_{(\mathit{Robo},10\&11)}$ learner responses (40%, n=6) to open-ended questions in the *PAT Evaluation Questionnaire* related to the techniques used by *RoboMind* to support programming indicated that having the microworld next to the code being executed made it easier to see how the code is being implemented. Learners did not make any comment (like or dislike) related to typing of the code or the textual representation of the code.

7.3.2 Scratch

Scratch was administered to participating Grade 10 and Grade 11 IT learners at two of the participating schools. The results for each of these schools on the impact of *Scratch* on understanding of programming concepts are presented separately as *Scratch*$_1$ (Section 7.4.2.1) and *Scratch*$_2$ (Section 7.4.2.2). The results are separated per school as different circumstances at each of the schools influence the impact of *Scratch* on IT learners' understanding of programming concepts. Specifically, *Scratch*$_1$ school had a change of teacher during the treatment year, whereas *Scratch*$_2$ had no change of teacher. Learner feedback on the evaluation of the techniques used by *Scratch* to support programming are combined as evaluation of the PAT is not influenced by the school environment (Section 7.4.2.3).

7.3.2.1 Understanding of Programming Concepts: Scratch$_1$

Twenty-five percent of Grade 10 learners who received *Scratch* indicated that they utilised the PAT. *Scratch* was used by 38% of the Grade 11 learners who received it. A high rate of attrition due to teaching circumstances at the school, out of the researchers' control, resulted in small sample sizes with regards to $Treatment_{(Scratch1,10\&11)}$ group results.

The *Perceived Difficulty of Programming Questionnaire*, *PAT Evaluation Questionnaire* and multiple choice class tests were used to evaluate the impact of *Scratch* on IT learners' understanding of the programming concepts supported by *Scratch* (Table 7.6). Control and treatment group perceived difficulty of the programming concepts and multiple choice class test marks are compared.

The mean ratings (*Perceived Difficulty of Programming Questionnaire*) for input, *if*-statements, string handling and procedures indicate that the $Treatment_{(Scratch1,11)}$ group respondents perceived these concepts to be more difficult than the $Control_{(Scratch1,11)}$ group respondents. A two-way ANCOVA analysis does not indicate statistically significant differences between the control and treatment group mean ratings for any of the programming concepts supported (Table 7.6). The concepts with the lowest treatment group mean ratings (n=3) and with higher corresponding control group ratings (n=15), are the *for*-loop (μ_1=1.73, μ_2=1.33), *while*-loop (μ_1=1.93, μ_2=1.33) and array concepts (μ_1=1.87, μ_2=1.33). Only three treatment group learners completed the *Perceived Difficulty of Programming Questionnaire*, affecting the reliability of the comparison between control and

Table 7.6: Scratch$_1$ support of programming concepts results

	PDPQ			**Multiple Choice Tests**			**PAT Eval**	
Programming Concept	μ_1	μ_2	**ANCOVA**	μ_1	μ_2	**ANOVA**	μ_2	**T-stat**
String handling	1.80(0.86) n=15	2.00(1.00) n=3	F(1;15)=0.75 p=0.400				2.00(1.41) n=4	t(3)=-1.41 p=0.252
One-dimensional arrays	1.87(0.99) n=15	1.33(0.58) n=3	F(1;15)=0.82 p=0.380	97.67(4.95) n=15	86.67(23.09) n=2	PATs received after test	1.5(0.71) n=2	t(1)=-3.00 p=0.205
Procedures	2.5(1.13) n=15	3.3(2.08) n=3	F(1;12)=1.60 p=0.230	92.00(14.74) n=15	75.00(35.36) n=2	F(1;24)=1.78 p=0.202	2.00(1.41) n=2	t(1)=-0.500 p=0.500
repeat-loops	2.27(1.1) n=15	1.67(1.15) n=3	F(1;14)=0.41 p=0.534	70.00(30.08) n=20	46.67(28.28) n=9	F(1;24)=1.35 p=0.257	3.00(1.15) n=4	t(3)=0.00 p=1.00
while-loops	1.93(0.88) n=15	1.33(0.58) n=3	F(1;15)=0.57 p=0.462	71.43(22.71) n=20	53.97(30.12) n=9	F(1;24)=0.61 p=0.422	2.50(1.29) n=4	t(3)=-0.77 p=0.495
for-loops	1.73(0.8) n=15	1.33(0.58) n=3	F(1;15)=0.81 p=0.382	62.84(20.08) n=19	59.63(21.67) n=8	F(1;24)=0.00 p=0.999	3.00(1.41) n=5	t(3)=0.00 p=1.000
if-statements	1.27(0.46) n=15	1.68(1.15) n=3	F(1;15)=1.07 p=0.317	72.00(17.79) n=20	68.00(14.02) n=8	F(1;24)=0.65 p=0.430	3.60(1.14) n=5	t(4)=1.18 p=0.305
Variables	1.33(0.35) n=15	1.33(0.58) n=3	F(1;15)=0.62 p=0.444				3.00(1.00) n=5	t(4)=0.00 p=1.000
Input	1.27(0.46) n=15	1.33(0.58) n=3	F(1;15)=0.02 p=0.899				3.2(1.48) n=5	t(4)=0.30 p=0.78
Output	1.47(0.64) n=15	1.33(0.58) n=3	F(1;15)=0.01 p=0.914				3.00(1.41) n=5	t(4)=0.00 p=1.000

treatment group results. The multiple choice class test mean scores for the treatment group learners are lower than the control group scores for all of the programming concepts supported. None of the differences are statistically significant (Table 7.6).

$Treatment_{(Scratch 1,10\&11)}$ learners provided feedback regarding the usefulness of the *PAT* in learning about the supported programming concepts. The results are obtained using the *PAT Evaluation Questionnaire*. The mean evaluation ratings (Table 7.6) for input, output, variables, *if*-statements, *for*-loops and *repeat*-loops are greater than or equal to the neutral value, 3. One-sample t-test analysis indicates that none of the ratings are statistically significant from the neutral value.

Only one of the two evaluations (Section 2.5.1) is reported as learners did not complete the questionnaire and because of administrative difficulties (Section 7.2.5). The number of learners, who evaluated the usefulness of *Scratch* to understand the programming concepts supported, ranged from two to five learners, depending on the concept (Table 7.6).

$Treatment_{(Scratch 1,10\&11)}$ group learner evaluations of the difficulty of programming concepts (*Perceived Difficulty of Programming Questionnaire*) as well as the neutral mean results from the evaluation of the PAT (*PAT Evaluation Questionnaire*) indicate, although not significantly, that *Scratch* provides support for the looping concepts, particularly *for*-loops. *Scratch* supports a repeat block (Section 6.2.3) as the Delphi *for*-loop equivalent where the number of times to repeat the code statements is indicated. The *for*-loop characteristic of finite repetitions is demonstrated using a repeat block even though the syntax does not match. The multiple choice class test results do not provide any indication of programming concepts supported by *Scratch* in terms of improving learner understanding.

An evaluation of the comparison of the treatment and control group results indicates that treatment group learners struggled to understand programming concepts (*Perceived Difficulty of Programming Questionnaire* and multiple choice test results). The IT teacher was replaced after the first term of the treatment year. The results suggest that the learners' transition to a different IT teacher and teaching environment influenced the poorer performance (tests) and greater perceived difficulty of programming concepts by treatment group learners. The results thus suggest that *Scratch* is not able to support learners who do not have a stable teaching environment.

7.3.2.2 Understanding of Programming Concepts: Scratch$_2$

The *PAT Evaluation Questionnaire* responses indicate that *Scratch* was installed and used by fourteen Grade 10 and six Grade 11 learners. These learners form the respec-

Table 7.7: Scratch$_2$ support of programming concepts results

	PDPQ			**Multiple Choice Tests**			**PAT Eval**	
Programming Concept	μ_1	μ_2	**ANCOVA**	μ_1	μ_2	**ANOVA**	μ_2	t
String handling	2.25(1.89) n=4	3.5(1.05) n=6	F(1;7)=0.32 p=0.591				1.17(0.91) n=14	t(13)=-5.26 ***p*=0.0002****
One-dimensional arrays	4.75(2.06) n=4	3.00(1.58) n=5	F(1;2)=0.12 p=0.763	92.00(10.95) n=5	79.17(13.57) n=6	PATs received after test	1.89(1.17) n=9	t(8)=-2.86 ***p*=0.021***
Procedures	3.54(1.00) n=4	4.33(1.21) n=6	F(1;5)=15.31 p=0.012	83.33(28.87) n=3	Marks not received	–	2.14(1.47) n=7	t(6)=-1.55 p=0.172
repeat-loops	3.5(2.52) n=4	3.83(1.60) n=6	F(1;7)=0.08 p=0.790	40.00(30.98) n=6	51.43(29.05) n=14	F(1;14)=0.09 p=0.767	2.38(1.31) n=16	t(15)=-1.91 p=0.076
while-loops	3.5(2.08) n=4	3.33(1.21) n=6	F(1;7)=0.17 p=0.690	50.00(29.62) n=6	30.20(29.21) n=14	F(1;14)=0.03 p=0.860	2.13(1.15) n=16	t(15)=-3.05 ***p*=0.008****
for-loops	2.5(1.73) n=4	2.83(0.98) n=6	F(1;6)=1.89 p=0.218	64.67(17.80) n=3	50.57(23.61) n=14	F(1;14)=0.72 p=0.409	2.29(1.31) n=17	t(16)=-2.22 ***p*=0.041***
if-statements	1.5(1.00) n=4	1.83(0.41) n=6	F(1;7)=1.62 p=0.243	86.4(13.74) n=5	73.23(20.62) n=13	F(1;14)=1.40 p=0.257	2.25(1.34) n=16	t(15)=-2.24 ***p*=0.041***
Variables	1.25(0.5) n=4	2.00(0.00) n=6	F(1;7)=19.97 p=0.003				2.76(1.25) n=17	t(14)=-0.78 p=0.450
Input	2.75(1.26) n=4	2.50(1.22) n=6	F(1;7)=19.97 p=0.564				2.88(1.45) n=17	t(16)=-0.33 p=0.743
Output	2.25(0.96) n=4	2.50(0.55) n=6	F(1;7)=0.12 p=0.734				3.00(1.27) n=17	t(16)=-0.00 p=1.000

* p<0.05
** p<0.01

tive treatment groups. $Treatment_{(Scratch2,11)}$ group learners perceived input, *while*-loops and arrays to be easier (Table 7.7) than the $Control_{(Scratch2,11)}$ group learners, as indicated by the lower mean ratings of these programming concepts in the posttest of the *Perceived Difficulty of Programming Questionnaire.* The $Control_{(Scratch2,11)}$ group mean ratings for the remaining programming concepts are less than the mean ratings for the $Treatment_{(Scratch2,11)}$ group. ANCOVA analysis indicates that the differences between the mean ratings are not statistically significant (Table 7.7).

The $Treatment_{(Scratch2,10)}$ group mean class test scores (Table 7.7) for *repeat*-loops and *while*-loops are greater than the $Control_{(Scratch2,10)}$ group mean test scores. The *for*-loop and *if*-statement test mean scores for the $Treatment_{(Scratch2,10)}$ group are less than the $Control_{(Scratch2,10)}$ group mean scores. The mean score differences are not statistically significant as indicated by one-way ANOVA analysis. The array class test for the $Treatment_{(Scratch2,11)}$ group was administered before the learners received *Scratch* and the procedure test scores were not recorded.

The usefulness of *Scratch* in terms of assisting learners with the understanding of supported programming concepts was evaluated using the PAT Evaluation Questionnaire (Table 7.7). The mean rating of all the programming concepts is less than or equal to the neutral value, 3. One sample t-test analysis indicates that the mean rating of arrays, *while*-loops, string handling, *for*-loops and *if*-statements is statistically significantly less than the neutral value, 3, indicating that learners did not agree that *Scratch* assisted with their understanding of these programming concepts.

The results indicate that treatment group learners perceived the difficulty of programming concepts (input, *while*-loops and arrays) to be easier when compared with the perception of the control group learners although the differences are not statistically significant. *PAT Evaluation Questionnaire* results contradict the learners perceived difficulty of programming concepts. This indicates that learners do not believe *Scratch* assisted with the understanding of these concepts.

The mean multiple choice test score for *repeat*-loops is higher when compared with the control group mean score, although the difference is not statistically significant. Treatment group learners perceived *repeat*-loops to be more difficult than control group learners did. Overall, the results suggest that *Scratch* did not impact learner understanding of programming knowledge.

7.3.2.3 Evaluation of Techniques

Scratch provides an interface where learners must drag and drop picture code blocks to build a program solution (Table 7.8). The resulting program can include animation and sound. User input can be accepted thus programs can be created that perform calculations based on user input or interactive games can be created that respond to keyboard input or mouse-events.

Table 7.8: Scratch techniques

Representation of the program solution
Text
Pictures
Construction of programs
Assembling or positioning graphical objects
Selecting/form filling
Resulting program
GUI
Animation
Preventing syntax errors
Selection from valid options
Dropping only in valid locations

Respondents' provided responses to open-ended questions in the *PAT Evaluation Questionnaire* about what they liked and disliked about *Scratch*. Only 6 of the 31 responses to what respondents liked about *Scratch* could be associated with the theme: the role of techniques to improve an understanding of programming concepts.

Respondents (n=4) liked that the graphical representation made the programming code solution easier to read. One learner indicated that *Scratch* helps with the understanding of programming problems while another learner indicated that *Scratch* helped with the understanding of loops. The remainder of the responses to what learners liked, as well as the responses to what learners disliked about *Scratch*, are associated with learner motivation towards programming (Section 8.3.2).

7.3.3 B#

Seventy-five percent (n=18) of Grade 10 learners and 61% (n=11) of Grade 11 learners who originally received *B#* indicated that they used it. The results are presented to evaluate the impact of *B#* on IT learner understanding of programming concepts (Section

7.3.3.1) and to evaluate the effect of the techniques used by *B#* to support the understanding of programming concepts (Section 7.3.3.2).

7.3.3.1 Understanding of Programming Concepts

The perceived difficulty of programming concepts was evaluated by Grade 11 control and treatment group learners (Table 7.9). The $Treatment_{(BSharp,11)}$ mean ratings for all of the programming concepts are less than the $Control_{(BSharp,11)}$ group mean ratings. Lower scores indicate that the $Treatment_{(BSharp,11)}$ group learners perceived the programming concepts to be easier than the $Control_{(BSharp,11)}$ group learners did. The mean scores are equal for *case*-statements. ANCOVA analysis indicates that the difference between the control and treatment group means are not statistically significant.

The multiple choice class test results (Table 7.9) indicate that $Treatment_{(BSharp,10)}$ group mean test scores are greater than the $Control_{(BSharp,10)}$ group mean test scores for the three looping concepts. One-way ANOVA analysis indicates that the mean score differences are not statistically significant. The $Treatment_{(BSharp,10)}$ group learner test scores for the *if*-statements test were not recorded (Section 7.2.5).

The usefulness of *B#* with regard to the understanding of the programming concepts supported, was evaluated twice (Table 7.9) by $Treatment_{(BSharp,10\&11)}$ learners in the academic year (Section 7.2.3). In both evaluations, the results indicate that learners did not agree that *B#* was useful to assist with the understanding of *repeat*-loops, *while*-loops, *if*-statements and *case*-statements. For all of these concepts, one sample *t*-test analysis indicated that the mean rating score is statistically significantly different from the neutral value, 3. The mean rating for output in the first evaluation are also statistically, significantly different from the neutral value. The results for *for*-loops were not captured due to an error in the questionnaire (Appendix E).

Treatment group learners have higher mean ratings than control group learners for the looping concepts evaluated using the perceived difficulty of programming concepts results and multiple choice test results. However, none of the mean differences are statistically significant and treatment group learners disagree that *B#* assisted with the understanding of the looping concepts. Overall, there is no indication from the results that *B#* assisted with learner understanding of programming knowledge.

Table 7.9: B# support of programming concepts results

	PDPQ			**Multiple Choice Tests**			**PAT Eval**			
Programming							**Eval1**		**Eval2**	
Concept	μ_1	μ_2	**ANCOVA**	μ_1	μ_2	**ANOVA**	μ_2	**T-stat**	μ_2	**T-stat**
repeat-loops	3.57 (0.98) n=7	2.5 (1.69) n=8	F(1;12)=0.58 p=0.462	44.00 (34.70) n=20	53.33 (25.67) n=18	F(1;36)=0.87 p=0.357	2.20 (1.03) n=10	t(9)=-2.45 ***p*=0.037***	2.00 (1.22) n=9	t(8)=-2.45 ***p*=0.040***
while-loops	2.88 (0.64) n=8	2.13 (1.55) n=8	F(1;13)=0.75 p=0.401	46.49 (34.10) n=20	62.70 (24.57) n=18	F(1;36)=2.77 p=0.105	2.00 (1.05) n=10	t(9)=-3.00 ***p*=0.015***	1.78 (1.09) n=9	t(8)=-3.35 ***p*=0.010***
for-loops	2.50 (0.76) n=8	2.00 (1.41) n=8	F(1;13)=0.05 p=0.821	60.89 (23.36) n=19	62.22 (29.05) n=18	F(1;35)=0.02 p=0.879	Error in questionnaire			
if-statements	1.88 (0.64) n=8	1.75 (0.89) n=8	F(1;13)=0.01 p=0.912	70.33 (20.14) n=12	Marks not received	–	2.00 (0.93) n=15	t(14)=-4.18 ***p*=0.001****	1.78 (1.09) n=10	t(8)=-3.35 ***p*=0.010***
case-statements	2.5 (1.20) n=8	2.5 (1.41) n=8	F(1;13)=0.36 p=0.557				2.00 (1.21) n=12	t(11)=-2.87 ***p*=0.015***	1.67 (0.87) n=9	t(8)=-4.62 ***p*=0.002****
Variables	2.00 (1.41) n=8	1.50 (0.76) n=8	F(1;12)=0.41 p=0.535				2.6 (1.06) n=15	t(14)=-1.47 p=0.164	2.00 (1.5) n=9	t(8)=-2.00 p=0.081
Input	1.75 (0.71) n=8	1.38 (0.74) n=8	F(1;13)=1.06 p=0.321				2.4 (1.12) n=15	t(14)=-2.07 p=0.057	2.11 (1.69) n=9	t(8)=-1.57 p=0.154
Output	2.13 (1.25) n=8	1.38 (0.74) n=8	F(1;13)=1.74 p=0.210				2.33 (1.11) n=15	t(14)=-2.32 ***p*=0.036***	2.11 (1.69) n=9	t(8)=-1.57 p=0.154

* $p<0.05$
** $p<0.01$

7.3.3.2 Evaluation of Techniques

B# creates a program solution using a flowchart (Table 7.10). Icons representing different programming concepts are dragged into the correct position in the flow of execution. Pascal code for the execution of a console application is automatically.

Table 7.10: B# techniques

Representation of the program solution
Text
Flowchart
Construction of programs
Assembling or positioning graphical objects
Resulting program
Text (input & output)
Support to understand programs
Debugging
Preventing syntax errors
Dropping only in valid locations

$Treatment_{(BSharp,10\&11)}$ responses to open questions in the *PAT Evaluation Questionnaire* were varied with regard to what they liked and disliked about *B#*. Learners indicated that they liked the automatic generation of the Pascal code representation (n=7), with specific mention of the automatic generation of variables by 3 learners. Two of the 7 responses indicated that the graphical representation of the flowchart explained programming concepts, particularly loops and *if*-statements.

Learners disliked that the generated code differs from Delphi (27% of responses, n=6). The only difference between the syntax of the supported programming concepts in *B#* and Delphi are the `writeln` and `readln` functions used for output and input, respectively (Section 6.2.4).

7.4 Conclusion

Three PATs -*RoboMind*, *Scratch* and *B#* - have been evaluated in terms of their impact on IT learner understanding of programming concepts. Data from the multiple choice class tests, *Perceived Difficulty of Programming Questionnaire* and *PAT Evaluation Questionnaire* have been used for the evaluations.

The first aim of this chapter has been to address RQ5: *What impact do the different PATs have on IT learner understanding of programming concepts?*. ANOVA/ANCOVA analysis results indicate that, overall, there are no statistically significant differences between the means of control and treatment groups for the three PATs for the multiple choice class test and end-of-year IT summative assessment mark. The null hypothesis, $H_{1,0}$, could thus not be rejected based on the assessment marks.

The overall *Perceived Difficulty of Programming Questionnaire* results could not reject the null hypothesis, $H_{1,0}$, for any of the PATs. There are no statistically significant results to indicate that *Scratch* and *B#* assisted learner understanding of programming concepts. The *Perceived Difficulty of Programming Questionnaire* results indicate that *RoboMind* assisted with the understanding of *repeat*- and *while*-loops. The null hypothesis was rejected for these two programming concepts. However, the null hypothesis could not be rejected for the remainder of the programming concepts. IT learner feedback identified the syntax of *RoboMind* to be an issue. The syntax of *for*-loops and variables in *RoboMind* differs from the Delphi programming language syntax (Section 6.2.2).

The second aim of this chapter has been to address RQ7: *What techniques should PATs implement in order to assist IT learner understanding of programming concepts?*.

Table 7.11: Impact of PAT techniques on understanding of IT programming concepts

Tool	**Technique**	**Impact**
RoboMind	Graphical environment	See what program is doing
	Code next to animation	Can follow code execution
	Code solution	Helps with programming, but syntax different to Delphi
Scratch	Graphical representation	Easy to read
B#	Flowchart representation	Explains programming concepts
	Automatically generated code solution	Like that code is generated from flowchart but differs from Delphi

The feedback received from IT learners regarding what they liked and disliked about the PAT (*PAT Evaluation Questionnaire*) they used has provided an indication of the impact that the different techniques used by the PAT have on IT learner understanding of programming concepts. Table 7.11 summarises what learners liked and disliked about the techniques with respect to understanding of programming concepts. All three PATs assist learners to read the programming solution more easily, indicating that learners can benefit from graphical representations of code solutions and the placing of output next to the code for more easy comparison (*RoboMind*).

This chapter has answered RQ5 and RQ7 and thus addressed Secondary Objective 2.2: *Evaluate the impact of the proposed PATs on IT learner understanding of programming concepts.* The evaluation of the combination of the assessment results, learner's perceived difficulty of programming and the evaluation of the usefulness of the PATs for each programming concept concludes that, overall, the null hypothesis, $H_{1,0}$, could not be rejected. The results have indicated that use of the PATs have had no statistically significant impact on IT learner understanding of programming concepts. Qualitative feedback from participants has suggested that the techniques used by the PATs have supported IT learner understanding of programming concepts. The impact of the PATs on IT learner motivation towards programming is evaluated in Chapter 8. The impact of the techniques used by the PATs on IT learner motivation is also evaluated.

Chapter 8

The Impact of PATs on Motivation Towards Programming

8.1 Introduction

The evaluation of the PATs aims to determine two factors, namely the impact of the PATs on IT learner understanding of programming concepts (Chapter 7) and the impact of the PATs on IT learner motivation towards programming. The aim of this chapter is to address the latter in order to achieve Secondary Objective 2.2: *Evaluate the influence of the proposed PATs on IT learner motivation towards programming.*

Secondary Objective 2.2 is achieved by comparing control and treatment group responses to the *Motivated Strategies for Learning Questionnaire (MSLQ)* (Section 2.5.1) and Grade 10 IT learner responses to the *IT Decision Questionnaire* (Section 2.5.1, Appendix B). Results obtained from these two questionnaires are used to address RQ6: *What impact do the PATs have on IT learner motivation towards programming?.*

The results are also used to test the research hypothesis H_2 (Section 2.3) in order to determine if the null hypothesis $H_{2,0}$, which states that there is no difference between the motivational strategy means of the control and treatment groups (ω_1=$\omega 2$, $\omega 1$ is the control group motivational strategy mean score, ω_2 is the treatment group motivational strategy mean score), is rejected. If $H_{2,0}$ is rejected, the alternative hypothesis, $H_{2,1}$: $\omega_1 \neq \omega_2$, is accepted.

Self-reported evaluations of the PATs by treatment group learners in the *PAT Evaluation Questionnaire* (Appendix E) provide feedback related to the ease of use and learnability of the PATs, as well as learner comments on what they liked and disliked about the PAT that can be related to their motivation to use the PAT. Learner feedback from the *PAT*

Evaluation Questionnaire, specifically referring to the influence of techniques used by the PAT on motivation, is used to address RQ8: *What techniques should PATs implement in order to motivate IT learners to learn programming concepts and to use a PAT?*

The research process (Section 8.2) describes the methods used to collect data for the results presented in this chapter. The results for each of the PATs provided to the participating schools - *RoboMind* (Section 8.3.1), *Scratch* (Section 8.3.2) and *B#* (Section 8.3.3)- are presented. The impact, if any, of each of the PATs on IT learner motivation towards programming and motivation to use the PATs, to address RQ6 and RQ8, are evaluated (Section 8.4) in the conclusion.

8.2 Application of Experimental Procedure

The results related to IT learner motivation towards programming and learner willingness to continue programming as a subject, are presented in this chapter. The *PAT Evaluation Questionnaire* (Section 2.5.2, Appendix E) was administered to IT learners to evaluate their use of the PAT they received. The items to determine the participant's evaluation of the PAT, in terms of factors that may motivate and techniques to motivate use of the PAT, are reported.

The *IT Decision Questionnaire* (Section 2.5.1) was administered to Grade 10 learners in the control and treatment groups as pre- and posttests (Figure 7.1). The results presented evaluate the control and treatment group responses to the perceived difficulty of IT as well as IT learners' decision to select IT as a subject.

Grade 10 and Grade 11 learners in the control and treatment groups completed the *Motivated Strategies for Learning Questionnaire (MSLQ)* (Section 2.5.1) at the beginning (pretest) and towards the end (posttest) of the respective academic year. Six different motivational subscales are evaluated (Table 8.1) using two-way ANCOVA analysis to determine if there is a main effect for the independent variable, experimental group (control/treatment), or an interaction effect (experimental group*use of the PAT).

Analysis of the effect of the experimental group on motivation (Table 8.2), regardless of which specific PAT is used, highlighted one significant result. The Grade 10 results indicate a small statistically significant main effect for the experimental group on test anxiety (F(1,117)=5.00, p=0.027 d=0.37). Treatment group learners had a greater variance between the pre- (ω_2=3.26, s=1.144) and posttest (ω_2=3.87, s=1.352) scores, showing an increase in test anxiety, compared to control group learners (pretest: ω_1=3.35,

Table 8.1: Motivated Strategies for Learning Questionnaire: Motivational subscales

	Subscale	Description
IG	Intrinsic goal orientation	Participation in task for reasons such as challenge, curiosity or mastery.
EG	Extrinsic goal orientation	Participation in task for reasons such as grades, performance and competition.
TV	Task value	How interesting, important or useful a task is.
CLB	Control of learning beliefs	Belief that outcomes are as a result of one's own effort.
SE	Self-efficacy for learning	Self-appraisal of one's ability to master a task.
TA	Test anxiety	Anxiety and negative thoughts that disrupt test performance.

s=1.234; posttest: ω_1=3.51, s=1.467). The analysis of the interaction effect (PAT received*experimental group) on test anxiety in Grade 10 was not statistically significant. No specific PAT thus affected the increase in test anxiety of learners in the treatment group. There were no other statistically significant differences between the control and treatment group *MSLQ* results (Table 8.2).

Table 8.2: ANCOVA analysis results: Effect of experimental group on MSLQ sub-categories

	Grade 10		**Grade 11**	
MSLQ category	**F(df)**	***p***	**F(df)**	***p***
IG	F(1;117)=0.06	p=0.809	F(1;63)=1.41	p=0.239
EG	F(1;117)=0.04	p=0.843	F(1;64)=0.09	p=0.769
TV	F(1;117)=1.09	p=0.299	F(1;63)=1.18	p=0.282
CLB	F(1;117)=0.04	p=0.841	F(1;64)=0.03	p=0.871
SE	F(1;117)=0.58	p=0.448	F(1;63)=0.08	p=0.780
TA	F(1;117)=5.00	***p*=0.027***	F(1;64)=0.77	p=0.383

* p<0.05

8.3 PAT Results

The results are grouped on the PATs provided to participating learners, namely *RoboMind* (Section 8.3.1), *Scratch* (Section 8.3.2) and *B#* (Section 8.3.3). The results of the Grade 10 *IT Decision Questionnaire* compare learner inclination to continue with IT as a subject at the end of Grade 10. IT learner evaluation of the PATs in terms of motivation to use the PAT, are also presented. The results aim to evaluate the impact of each PAT and the techniques used by the PAT on IT learner motivation towards programming.

8.3.1 RoboMind

ANCOVA analysis was used to determine any difference between the control and treatment group motivational subscale means. The results indicate that there are no statistically significant differences between the control and treatment group for each of the MSQL subscales (Table 8.3).

Table 8.3: ANCOVA analysis results: Effect of *RoboMind* on MSLQ subscales

	Grade 10		Grade 11	
MSLQ category	**F(df)**	***p***	**F(df)**	***p***
IG	F(1;44)=2.31	p=0.136	F(1;27)=0.00	p=0.953
EG	F(1;44)=0.02	p=0.883	F(1;27)=1.58	p=0.220
TV	F(1;44)=0.21	p=0.649	F(1;27)=0.00	p=0.962
CLB	F(1;44)=0.09	p=0.766	F(1;27)=0.05	p=0.819
SE	F(1;44)=0.27	p=0.606	F(1;27)=1.17	p=0.290
TA	F(1;44)=0.16	p=0.693	F(1;27)=0.00	p=0.948

A smaller percentage (11%) of $Treatment_{(Robo,10)}$ learners (n=3) would not consider taking IT again if given the choice compared to 18% of $Control_{(Robo,10)}$ learners (n=4). $Treatment_{(Robo,10)}$ learners who responded “no” considered the subject difficult and not necessary for their career choice after school. $Control_{(Robo,10)}$ learners who responded “no” believed IT to be difficult to understand and boring.

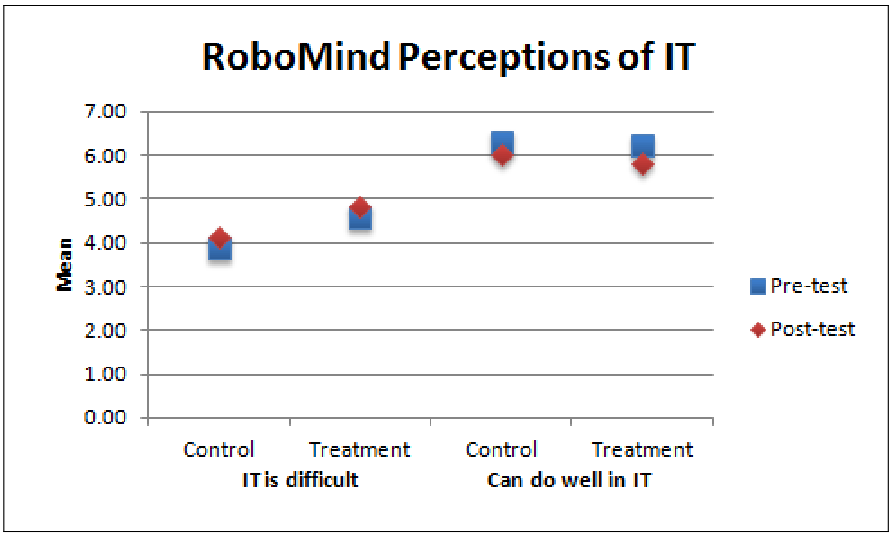


Figure 8.1: RoboMind: Learner perceptions of IT difficulty (1 = strongly disagree, 7 = strongly agree)

Grade 10 learners rated their belief that IT is a difficult subject and that they can do well in the IT subject, at the beginning and end of the year. ANCOVA analysis of the variance in pre- and posttest scores for *Control*$_{(Robo,10)}$ and *Treatment*$_{(Robo,10)}$ learners indicate no statistically significant differences for the *IT is difficult* (F(1;46)=0.58, p=0.451) or the *can do well in IT* (F(1;46)=0.17, p=0.683) items.

Table 8.4: Learner Evaluation of RoboMind (1 = strongly disagree, 5 = strongly agree)

Item	n	ω	s	t-test	p
Easy to use	29	3.28	0.92	1.61	p=0.118
Learn quickly	29	3.34	0.97	1.91	p=0.067
Need assistance	29	2.21	1.15	-3.73	***p*=0.001****
Confident using	29	3.14	1.06	0.70	p=0.490
** p<0.01					

Treatment$_{(Robo)}$ learner results (Table 8.4) indicated that *RoboMind* is easy to use, quick to learn to use and learners are confident using *RoboMind*. However, none of these results are statistically significantly different from the neutral value, 3. Results also indicate that learners do not need assistance using *RoboMind*. This result is statistically significantly, different from the neutral value (Table 8.4).

Learner comments on what they liked (n=8) and disliked (n=2) about *RoboMind* can be associated to motivation to use or not to use *RoboMind*. Learners liked that *RoboMind* is entertaining (n=2) and makes programming more like a game, not just coding as the robot can be programmed to move around (n=6). The microworld output technique is thus motivating for learners.

Learners (n=4) mentioned that they disliked the fact that using *RoboMind* became boring and that there was no goal or reason to keep playing (n=4). *RoboMind* is limited in its functionality, only allowing the robot to be programmed to move around the map world, pick-up/drop objects and paint.

8.3.2 Scratch

The results for two schools that evaluated *Scratch* are reported separately as *Scratch*$_1$ (Section 8.3.2.1) and *Scratch*$_2$ (Section 8.3.2.2). Circumstances at each of these schools differed due to a change of teacher at one of the schools, *Scratch*$_1$.

8.3.2.1 Scratch$_1$

ANCOVA analysis (Table 8.5) indicates a statistically significant difference between the Grade 10 control and treatment group means for the test anxiety subscale of the MSLQ (Table 8.1). $Control_{(Scratch1,10)}$ learners had a greater variance between the pre- (ω_1=2.94, s=0.90, n=20) and posttest (ω_1=3.29, s=1.4, n=20) scores, showing an increase in test anxiety, compared to $Treatment_{(Scratch1,10)}$ learners (pretest: ω_2=3.17, s=1.02; posttest: ω_2=3.29, s=1.38; n=9).

Table 8.5: ANCOVA analysis results: Effect of *Scratch*$_1$ on MSLQ sub-categories

	Grade 10		Grade 11	
MSLQ category	**F(df)**	***p***	**F(df)**	***p***
IG	F(1;19)=0.81	p=0.379	F(1;14)=0.36	p=0.556
EG	F(1;19)=0.60	p=0.447	F(1;15)=3.16	p=0.096
TV	F(1;19)=0.22	p=0.646	F(1;14)=3.57	p=0.069
CLB	F(1;19)=0.01	p=0.918	F(1;15)=0.10	p=0.755
SE	F(1;19)=1.53	p=0.232	F(1;14)=0.00	p=0.987
TA	F(1;19)=6.55	***p*=0.019***	F(1;15)=3.63	p=0.076

* $p<0.05$

Eighty-nine percent of $Treatment_{(Scratch1,10)}$ learners (n=9) discontinued with IT as a subject during the year or indicated that they would no longer take IT as a subject if given the choice. The majority of $Control_{(Scratch1,10)}$ group learners (75%, n=20) indicated that they would still take IT if given the choice.

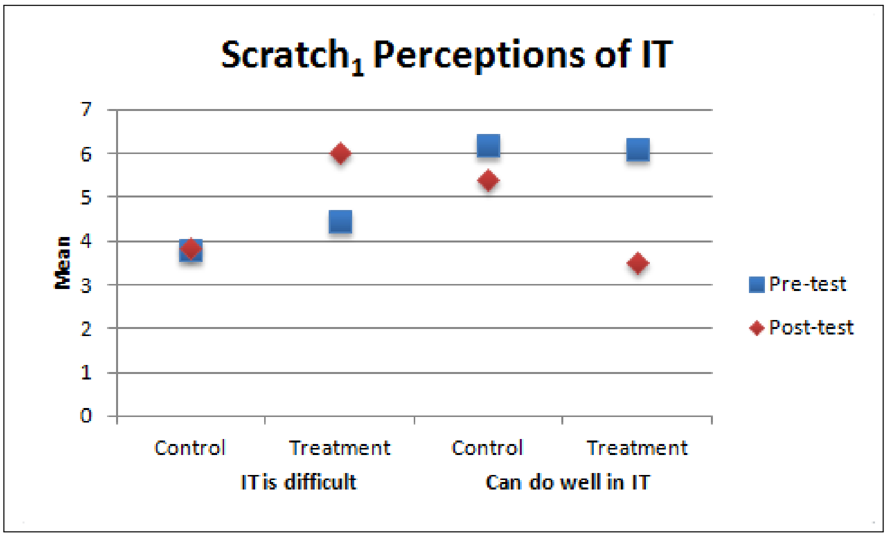


Figure 8.2: Scratch$_1$: Learner perceptions of IT difficulty (1 = strongly disagree, 7 = strongly agree)

The pre- and posttest results (*IT Decision Questionnaire*) to determine if learners think IT is difficult and if learners think they can do well in the subject (Figure 8.2) were compared for $Control_{(Scratch1,10)}$ and $Treatment_{(Scratch1,10)}$ group learners. ANCOVA analysis indicates a statistically significant main effect for the experimental group (F(1,18)=5.283, p=0.034) on the independent variable evaluating learners' perceived difficulty of IT. There is no statistically significant main effect for the experimental group on learners' belief that they can do well in IT (F(1,18)=4.32, p=0.523).

Table 8.6: Learner Evaluation of Scratch$_1$ (1 = strongly disagree, 5 = strongly agree)

Item	n	ω	s	t-test	p
Easy to use	17	3.71	1.26	2.30	***p*=0.035***
Learn quickly	17	4.00	0.79	5.22	***p*=0.0001****
Need assistance	17	2.00	1.06	-3.89	***p*=0.001****
Confident using	17	3.00	1.33	0.00	p=1.00

* $p<0.05$
** $p<0.01$

$Treatment_{(Scratch1,10\&11)}$ learners indicated (Table 8.6) that *Scratch* is easy to use and quick to learn to use. Results also indicate that learners do not need assistance using *Scratch.* One sample t-tests indicate that all three these results are statistically, significantly different from the neutral value, 3.

8.3.2.2 Scratch$_2$

ANCOVA analysis of the MSLQ results (Table 8.7) indicate a statistically significant difference between the means of the $Control_{(Scratch2,11)}$ (pretest: ω_1=5.10, s=1.43; posttest: ω_1=4.70, s=0.98; n=5) and $Treatment_{(Scratch2,11)}$ (pretest: ω_2=5.28, s=1.35; posttest: ω_2=4.83, s=0.94; n=6) group learners.

Table 8.7: ANCOVA analysis results: Effect of *Scratch*$_2$ on MSLQ sub-categories

	Grade 10		**Grade 11**	
MSLQ category	**F(df)**	***p***	**F(df)**	***p***
IG	F(1;17)=0.04	p=0.839	F(1;7)=0.07	p=0.806
EG	F(1;17)=1.81	p=0.196	F(1;7)=2.26	p=0.177
TV	F(1;17)=0.43	p=0.520	F(1;7)=5.72	***p*=0.048***
CLB	F(1;17)=0.10	p=0.753	F(1;7)=0.06	p=0.819
SE	F(1;17)=0.00	p=0.972	F(1;7)=2.73	p=0.143
TA	F(1;17)=0.08	p=0.786	F(1;7)=1.42	p=0.272

* $p<0.05$

The percentage of Grade 10 learners who would not repeat their decision to take IT as a subject is similar in the two groups, $Control_{(Scratch2,10)}$ (33%, n=2) and $Treatment_{(Scratch2,10)}$ (36%, n=5). Figure 8.3 indicates the $Control_{(Scratch2,10)}$ and $Treatment_{(Scratch2,10)}$ groups pre- and posttest mean scores for their rating of the difficulty of IT and their belief in their ability to do well in IT. ANCOVA analysis indicates no statistically significant main effect for the experimental group on the two variables (F(1;16)=2.04, p=0.172 and F(1;16)=1.71, p=0.209, respectively).

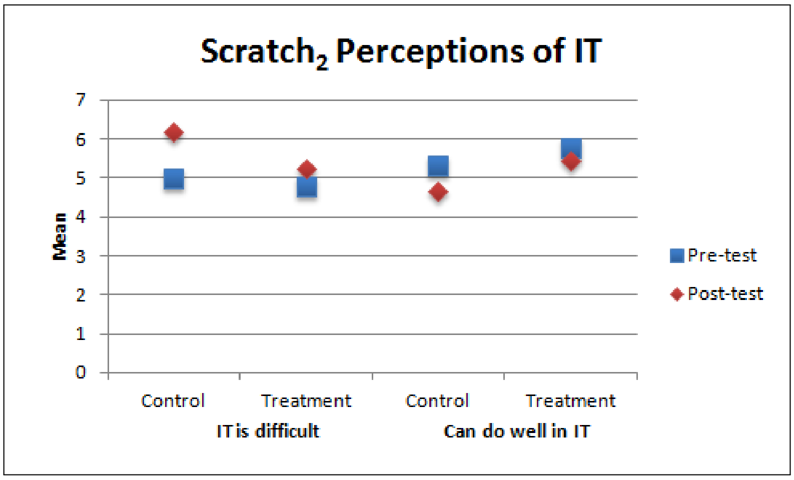


Figure 8.3: Scratch$_2$: Learner perceptions of IT difficulty (1 = strongly disagree, 7 = strongly agree)

Table 8.8: Learner Evaluation of Scratch$_2$ (1 = strongly disagree, 5 = strongly agree)

Item	n	ω	s	t-test	p
Easy to use	20	4.05	1.10	4.27	**p=0.0004****
Learn quickly	20	3.70	0.92	3.39	**p=0.003****
Need assistance	20	2.05	1.05	-4.05	**p=0.001****
Confident using	20	3.60	1.05	2.56	**p=0.019***

* p<0.05
** p<0.01

$Treatment_{(Scratch2,10\&11)}$ learner evaluation of the *PAT Evaluation Questionnaire* (Table 8.8) indicated that *Scratch* is easy to use and quick to learn to use. Results also indicate that learners do not need assistance using *Scratch* and are confident in their ability to use *Scratch*. One sample t-test indicates that all four mean values are statistically different from the neutral value, 3.

8.3.2.3 Combined Evaluation of Scratch Techniques

The evaluation of *Scratch* by learners at the two schools in terms of what learners liked and disliked about *Scratch* have been combined. The results are combined as learner evaluation of *Scratch* is independent of any influence of the IT teacher. The responses by learners could be classified in more than one theme.

Feedback to open-ended questions related to motivation in the *PAT Evaluation Questionnaire* indicate that 35% of learners (n=11) liked the animation, graphics and games that can be created in *Scratch*. Forty-two percent of learners (n=11) indicated that *Scratch* is easy to use and fun. *Scratch* uses a drag-and-drop technique to create program solutions and graphical program output.

Negative comments described *Scratch* as too simple (28%, n=8), limited (24%, n=7), impractical (10%, n=3) and with no goal or purpose (10%, n=7). The negative feedback suggests that learners may not be motivated to use *Scratch*. This would impact the effectiveness of *Scratch* to assist learner understanding of programming concepts and motivate them to program using *Scratch*.

8.3.3 B#

ANCOVA analysis was used to determine statistical significance between the control and treatment group means for the MSLQ sub-scales. The results indicated that there are no statistically significant differences (Table 8.9) between the control and treatment group responses.

Table 8.9: ANCOVA analysis results: Effect of *B#* on MSLQ sub-categories

	Grade 10		**Grade 11**	
MSLQ category	**F(df)**	***p***	**F(df)**	***p***
IG	F(1;34)=0.49	p=0.488	F(1;12)=1.60	p=0.230
EG	F(1;34)=0.12	p=0.730	F(1;12)=0.54	p=0.475
TV	F(1;34)=0.31	p=0.580	F(1;12)=0.03	p=0.865
CLB	F(1;34)=0.04	p=0.843	F(1;12)=0.33	p=0.578
SE	F(1;34)=0.10	p=0.749	F(1;12)=0.49	p=0.495
TA	F(1;34)=0.19	p=0.662	F(1;12)=0.00	p=0.946

In the posttest of the *IT Decision Questionnaire*, Grade 10 learners were asked if they would still take IT if they had the choice. Twenty-two percent of $Treatment_{(BSharp,10)}$ learners (n=4) indicated that they would not take IT as a subject again compared to 5%

of $Control_{(BSharp,10)}$ learners (n=1). Learners indicated that they would not take IT again because it is too time consuming (n=2) and they struggle to understand the work (n=3).

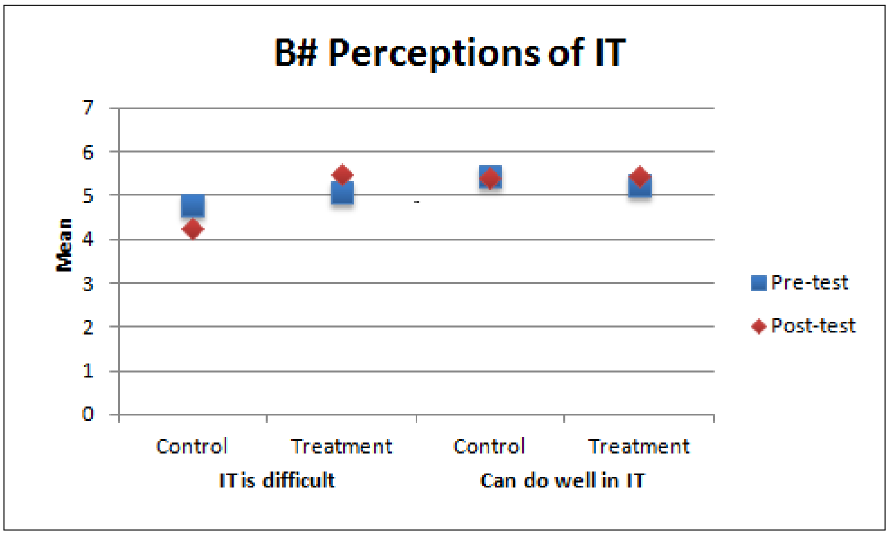


Figure 8.4: B#: Learner perceptions of IT difficulty (1 = strongly disagree, 7 = strongly agree)

The pre- and posttest mean results for $Control_{(BSharp,10)}$ and $Treatment_{(BSharp,10)}$ to determine if learners think IT is difficult and if learners think they can do well in the subject (Figure 8.4) were compared. ANCOVA analysis evaluating learner's perceived difficulty of IT (F(1;31)=3.39, p=0.075) and learners' belief that they can do well (F(1;31)=0.011, p=0.919), indicate that there is no statistically significant difference between the control and treatment group mean values.

$Treatment_{(BSharp,10\&11)}$ learners were required to evaluate *B#* (*PAT Evaluation Questionnaire*) using a 5-point semantic differential scale (1=strongly disagree and 5=strongly agree). The mean scores (Table 8.10) indicate that learners did not believe *B#* was quick to learn to use and learners were not confident using *B#*. One-sample t-test analysis indicates that the mean rating for these two items are statistically, significantly different from the neutral value, 3. The mean ratings for ease of use and whether learners needed assistance using *B#*, were neutral (Table 8.10).

Learner responses to open questions in the *PAT Evaluation Questionnaire*, where a learner response could be classified in more than one theme, indicated that learners do not want to use *B#* because it confuses their understanding of how concepts are implemented in Delphi (22%, n=7). Learners also indicated that they were happy with their understanding of Delphi and prefer using Delphi instead of *B#* (42%, n=14).

Table 8.10: Learner Evaluation of B# (1 = strongly disagree, 5 = strongly agree)

Item	n	ω	s	t-test	p
Easy to use	25	2.72	1.46	-0.96	p=0.347
Learn quickly	25	2.44	1.26	-2.22	**p=0.036***
Need assistance	25	2.72	1.40	-1.00	p=0.327
Confident using	24	1.29	0.26	-3.81	**p=0.0001****

* $p<0.05$
** $p<0.01$

The techniques used by *B#* were also evaluated using learner responses to what they liked and disliked about *B#* (*PAT Evaluation Questionnaire*). Learners indicated that they liked the visual representation of the program code using the flowchart (22%, n=4). Learners disliked that the console application output technique differs from the GUI form created when a Delphi program is executed (32%, n=7).

8.4 Conclusion

An evaluation of the three PATs -*RoboMind*, *Scratch* and *B#* - in terms of their impact on IT learner motivation towards programming has been concluded. Data from the *MSLQ*, *PAT Evaluation Questionnaire* and *IT Decision Questionnaire* were used for the evaluations.

One of the aims of this chapter has been to address RQ6: *What impact do the PATs have on IT learner motivation towards programming?* There is a statistically and practically significant main effect for experimental group (control/treatment) on test anxiety in the Grade 10 participant group. The results indicate a statistically significant increase in test anxiety of treatment group learners when compared with control group learners. This result is independent of the PAT used. The test anxiety motivational subscale score is the only result that rejects the null hypothesis, $H_{2,0}$. Overall, the motivational subscale results do not reject the null hypothesis, suggesting that the PATs did not have an impact on IT learner motivation towards programming.

There are no statistically significant results to show that *RoboMind* had an impact on IT learner motivation towards programming in terms of the *MSLQ* sub-categories and IT learner perceived difficulty of the IT subject (Section 8.3.1). IT learners' evaluation of *RoboMind* indicated that no assistance is needed to use *RoboMind*, making it suitable for self-study. Learner responses regarding ease of use, learnability and learner confidence using *RoboMind* were neutral.

The *MSLQ* results (Section 8.2.2) indicate that $\textit{Treatment}_{(Scratch2,11)}$ group learners had improved task value after using *Scratch* when compared with control group learners. $\textit{Control}_{(Scratch1,10)}$ learner responses indicated a greater increase in text anxiety by the end of Grade 10 when compared with treatment group learners. $\textit{Treatment}_{(Scratch1,10)}$ learners perceived the IT subject was more difficult by the end of Grade 10 when compared with their perceptions at the start of the year.

IT learner evaluations from participants at the two schools using *Scratch* indicate that it is easy to use, quick to learn to use and learners do not need assistance using it. $\textit{Scratch}_2$ results also indicate that learners are confident in their ability to use *Scratch*.

B# did not have an impact on IT learner motivation towards programming in terms of the MSLQ sub-categories and IT learner perceived difficulty of the subject (Section 8.3.3). IT learner evaluations indicate that *B#* is not quick to learn to use and learners are not confident in their ability to use *B#*. IT learner responses indicate that the use of *B#* confuses their understanding of programming concepts in Delphi.

The second aim of this chapter has been to address RQ8: *What techniques should PATs implement in order to motivate IT learners to learn programming concepts and to use a PAT?*. Treatment group learners at each of the schools evaluated the PATs using the *PAT Evaluation Questionnaire*. Responses to open-ended questions indicating what learners liked and disliked about the PAT were used to evaluate the techniques used by the PATs (Table 8.11). *RoboMind* was described as entertaining and learner comments indicated that *RoboMind* changes perceptions of programming, making it more like a game. The microworld output is motivating for learners, however, learners disliked that *RoboMind* is limited with no goal or purpose to programming the robot.

Table 8.11: Impact of PAT techniques on motivation to use the PAT

Tool	Technique	Impact
RoboMind	Microworld	Fun to program the robot to move
		Limited functionality with no goal - eventually becomes boring
Scratch	Drag-and drop	Too simple
	Graphical representation (output)	Easy to use and fun
B#	Flowchart	Simple to understand

Scratch, which uses a drag-and-drop interface to create program solutions, is described as easy to use and fun. Learners liked the program output, namely, the game and animations that can be created in *Scratch*. Despite these comments, learners may not be motivated to use *Scratch* to assist with programming. Learners described *Scratch* as too simple as solutions can be created easily by dragging objects. The program creation technique used by *Scratch* prevents syntax errors by allowing users to focus on planning a solution. However, learners felt that *Scratch* did not give them a sense of accomplishment.

Learners liked the flowchart representation of the program solution in *B#*. Learners liked that the Pascal code equivalent of the flowchart is generated automatically, however, differences in the syntax confused learners. Learners did not like the console application used by *B#* to output the program solution.

This chapter has answered RQ6 and RQ8 and thus addressed Secondary Objective 2.2: *Evaluate the influence of the proposed PATs on IT learner motivation towards programming.* Primary Objective 2 has thus also been addressed. This concludes the research on the evaluation of the impact of PATs on IT learner understanding of programming concepts and motivation towards programming. Chapter 9 presents a discussion of the general research conclusions and future research.

Chapter 9

Conclusion

9.1 Introduction

The demand for software developers is increasing, yet the number of learners choosing a career in Computing Science is decreasing, a trend occurring worldwide (Section 1.1). Governments are trying to promote interest in computing professions amongst school learners. However, several factors are affecting learner interest in computing professions as well as in Computer Science and Information Technology subjects at school. IT learners in SA struggle to understand introductory programming concepts (Chapter 1). Learners have limited time available during class with the IT teacher (Section 4.2). Learners who struggle with programming tend to lose motivation to continue with IT as a subject (Section 1.2).

Introductory programming research (Section 5.3) has resulted in the development of *Programming Assistance Tools (PATs)* designed to address the difficulties faced by novice programmers. PATs employ techniques such as the graphical representation of program solutions, visualisations to explain the use of programming concepts and the creation of animations and games. Unfortunately, IT teachers are not aware of and/or do not use PATs to support the learning of IT programming in SA schools (Section 4.2).

The purpose of this research was to evaluate the impact of PATs on IT learner understanding of programming concepts and motivation towards programming. The goal of the evaluation was to identify PATs that are able to complement the learning of programming concepts using the Delphi programming language and support IT learners during self-study.

This chapter reviews the research objectives (Section 9.2) and how these objectives have been successfully achieved (Section 9.3). The main contributions of the research are presented (Section 9.4) and limitations to the study and lessons learnt are highlighted (Section 9.5). The chapter concludes with suggestions for future work (Section 9.6) and a summary (Section 9.7).

9.2 Research Objectives

The purpose of the research objectives was to achieve the research goal of providing IT learners with existing PATs that are suitable or can be made suitable to support the achievement of the IT subject programming learning outcomes in South African secondary schools. The following thesis statement guided the research towards achieving the research objectives and answering the research questions:

> *Programming assistance tools can support IT learner development of programming skills and influence motivation to learn programming.*

The two primary objectives (POs) of this research were derived from the thesis statement and were both achieved:

PO1: To identify existing introductory programming assistance tools (PATs) that can be used to support learner understanding of programming concepts included in the IT subject curriculum implemented in a case study of South African secondary schools (Chapter 5).

PO2: To evaluate the impact of the selected programming assistance tools (PATs) on a novice programmer's understanding of programming concepts included in the IT subject curriculum implemented in a case study of South African secondary schools (Chapter 7 and 8).

The following secondary objectives (SOs) were researched in order to achieve the primary objectives (Table 1.2):

SO1.1: Formulate selection criteria for determining the suitability of PATs to support the achievement of programming learning outcomes in the IT subject curriculum (Chapter 4).

SO1.2: Adapt selected PATs to make them suitable to support the achievement of programming learning outcomes in the IT subject curriculum implemented in South African secondary schools (Chapter 6).

SO2.1: Evaluate the impact of the proposed PATs on IT learner understanding of programming concepts (Chapter 7).

SO2.2: Evaluate the influence of the proposed PATs on IT learner motivation towards programming (Chapter 8).

Secondary Objectives 1.1 and 1.2 address Primary Objective 1 and Secondary Objectives 2.1 and 2.2 address Primary Objective 2. A description of how the research objectives have been achieved, together with a review of the research questions answered in order to address the research objectives, are presented in the main findings (Section 9.3).

9.3 Main findings

The research objectives have been achieved in two phases. The first phase addresses Primary Objective 1 (Section 9.3.1) and the second phase addresses Primary Objective 2 (Section 9.3.2).

9.3.1 Primary Objective 1

The aim of Primary Objective 1 was to identify suitable PATs that could be used by IT learners participating in the study. The PATs identified were evaluated in the second phase of the study to address Primary Objective 2.

The suitability of PATs was determined by investigating the difficulties experienced by novice programmers (Chapter 3) and specifically, IT learners (Chapter 4). Research Question 1: *What programming difficulties and skills do PATs need to address and develop, respectively?* was answered by completing a review of introductory programming literature (Section 3.4) and administering questionnaires to IT teachers and learners (Section 4.4). The literature review and questionnaire results (*IT Teacher Questionnaires* (Appendix A) and *Perceived Difficulty of Programming Questionnaire* (Appendix C) identified programming knowledge that IT learners need to develop, such as knowledge of programming principles and knowledge of programming language syntax. Results showed that novice programmers struggle to identify how the programming problem should be solved and how to plan a solution. IT learners need assistance with debugging code and understanding error messages. The results from IT teachers and learners confirmed the programming difficulties identified in literature.

Selected PATs also had to be suitable to address the teaching environments experienced in SA schools. Feedback from IT teachers and learners related to the teaching and learning environment in SA schools was used to answer Research Question 2: *What factors* *may influence the use of PATs in SA secondary school learning environments?* (Section 4.4). The results showed that short class periods and overall limited contact time between teacher and learners meant that learners had to be able to use the PATs on their own. The PAT had to develop programming knowledge, skills and understanding of programming concepts during self-study. IT learners do not all have access after class time to a programming environment in which Delphi programming exercises can be implemented. A useful PAT would be able to implement programming exercises in the Delphi programming language syntax.

PATs selected by this research had to address the difficulties and factors identified in answering Research Question 1 and 2, respectively, in order for the PAT to have been of use to IT learners and to have assisted learners to overcome the difficulties of programming. A major contribution of this research study is the formulation of selection criteria (Section 4.4) from the difficulties and factors identified to successfully achieve Secondary Objective 1.1.

The selection criteria were categorised into three categories, namely programming concepts, programming knowledge and programming skills. The programming concept criteria included all programming concepts taught in the IT subject curriculum. Criteria in each of the categories were ranked in order of difficulty (programming concepts) or importance (programming knowledge and skills). The rankings were determined based on the common identification of criteria from literature and IT teacher and learner feedback.

VARK Questionnaire (Section 5.2.3) results and a review of introductory programming literature (Section 5.2.1) identified additional selection criteria in terms of the techniques used by PATs to create and display a program solution. VARK results indicated that PATs had to support all of the learning preferences, namely visual, aural, read/write and kinaesthetic, in order to cater for the learning preferences of IT learners. In the selection process, the techniques used by PATs to create program solutions, represent program solutions, display program output and assist with debugging were used to select PATs with different techniques. IT learner opinions regarding the different techniques on IT learner understanding of programming concepts and motivation towards programming could thus also be evaluated.

Existing PATs were identified to answer Research Question 3: *What PATs exist that would be suitable for use in SA secondary schools?*. Ten PATs were identified and reviewed (Section 5.3): *RoboMind*, *BlueJ*, *Greenfoot*, *Scratch*, *B#*, *Jeliot*, *Ville*, *PlanAni*, *Alice* and *jGRASP*. The 10 PATs were evaluated using the selection criteria in order to select the most suitable for use by IT learners (Section 5.4). The PATs were ranked in order of suitability for each of the three selection criteria categories. Only PATs that supported the Delphi programming language syntax were considered for the final selection. Three PATs selected for evaluation by IT learners in this study were *Scratch*, *RoboMind* and *B#*. Three PATs were selected for evaluation by the four participating schools. *Scratch* was selected for evaluation by the two participating schools with the smallest sample sizes (Section 5.4).

RoboMind was selected as, even though it did not support the Delphi programming language syntax, the source code is freely available for adaptation. The implementation of programming concepts in *RoboMind* was adapted to support the Delphi syntax (Section 6.2.2). The implementation of the adaptations required an understanding of the compiler used in the original *RoboMind*. The compiler was changed in this research study to accept Delphi keywords and implement *repeat*-loops and variables. A major contribution of this research is an adapted version of *RoboMind* suitable for use by IT learners. Adaptations to *RoboMind* addressed Research Question 4: *How can the selected PATs be adapted for use by IT learners in SA secondary schools to support the understanding of programming concepts?* and thus also successfully achieved Secondary Objective 1.2. The differences between the implementation of programming concepts in all three PATs compared to the implementation in Delphi were also identified (Section 6.2).

9.3.2 Primary Objective 2

The goal of Primary Objective 2 was to evaluate the impact of the PATs selected with the achievement of Primary Objective 1 on IT learner understanding of programming concepts (Secondary Objective 2.1) and motivation towards programming (Secondary Objective 2.2). Primary Objective 2 was achieved using a quasi-experimental approach over a two year period. Participants in the study were sampled from Grade 10 and Grade 11 IT learners at four participating schools. Participants in the first year of the study constituted the control group. Participants in the second year of the study received the PAT. However, use of the PAT was not enforced and not all participants who received the PAT, utilised it. Participants who received but did not utilise the PAT were included in the control group, while participants who utilised the PAT were included in the treatment group.

Control and treatment group results from multiple choice class tests (Appendix D), end-of-year summative assessment marks, learner perceived difficulty of IT concepts (Appendix C) and IT learner evaluation of the usefulness of the PAT received (Appendix E), were used to answer Research Question 5: *What impact do the different PATs have on IT* *learner understanding of programming concepts?*. There were no statistically significant results (Chapter 7) to indicate that any of the evaluated PATs impact IT learner understanding of programming concepts supported by the PAT. Results indicate that *RoboMind* may provide support for the *repeat*- and *while*-loop concepts (Section 7.3.1). Reasons for the lack of support, identified from IT learner evaluations of the PATs, indicate that *RoboMind* and *B#* (Section 7.3.2) should provide more support for the Delphi programming language syntax. Learner responses provided no reason for the lack of support offered by *Scratch* (Section 7.3.3).

The *Motivated Strategies for Learning Questionnaire* (MSLQ) and *IT Decision Questionnaire* (Appendix B) results were used to address Research Question 6: *What impact do the PATs have on IT learner motivation towards programming?*. The Grade 10 results (Section 8.2) indicated that treatment group learners, regardless of which PAT was used, had a greater increase in test anxiety between the start and end of the year than control group learners. No reason for the increase in test anxiety of Grade 10 treatment group learners was evident from the learner feedback. *Scratch*$_1$ results (Section 8.3.2) indicated that Grade 10 control group learners had a greater increase in test anxiety than treatment group learners. Grade 11 treatment group learners had a greater increase in task value compared to control group learners as indicated by the *Scratch*$_2$ results. These were the only *MSLQ* results indicating that the PATs had an impact on IT learner motivation towards programming.

Treatment group learners evaluated the PATs (Section 8.3) in terms of ease of use, learnability, assistance needed to use and confidence using the PAT (*PAT Evaluation Questionnaire*). Analysis of learner responses indicated that no assistance was needed to understand *RoboMind* and *Scratch* (*Scratch*$_1$ and *Scratch*$_2$ results). *Scratch* is easy to use and learners learnt to use it quickly. Learners indicated that they could not learn to use *B#* quickly and they were not confident in their abilities to use *B#*. The *MSLQ* results and responses to the *PAT Evaluation Questionnaire* describe the impact of the PATs on learner motivation towards programming. Secondary Objective 2.2 was therefore successfully achieved.

The techniques used by PATs were also evaluated to address Research Question 7: *What techniques should PATs implement in order to assist IT learner understanding of programming concepts?* (Chapter 7) and Research Question 8: *What techniques should PATs implement in order to motivate IT learners to learn programming concepts and to use a PAT?* (Chapter 8). Feedback from the three PATs indicated that a microworld or graphical program output such as animations or games are fun and entertaining, changing learners' perceptions of programming. However, to motivate learners to use the PAT, it should be challenging and with a goal or purpose for each program created. If a programming language is used to represent textually the program solution as in *RoboMind* and *B#*, the syntax must match the Delphi programming language syntax. Differences in the syntax confuse learners and make them hesitant to use the PAT.

9.3.3 Research Hypotheses

The main hypothesis (H_0) was tested using two sub-hypotheses (Section 2.3). The main hypothesis tested for this research was the following:

$H_{0,0}$: There is no difference in the performance/experience between control and treatment groups

$H_{0,1}$: There is a difference in the performance/experience between control and treatment groups

The first sub-hypothesis (H_1) evaluated the impact of the different PATs on IT learner understanding of programming concepts. The first sub-hypothesis stated the following:

$H_{1,0}$: There is no difference between the assessment means of the control and treatment groups (μ_1=μ_2, μ_1 is the control group assessment mean, μ_2 is the treatment group assessment mean)

$H_{1,1}$: There is a difference between the assessment means of the control and treatment groups ($\mu_1 \neq \mu_2$)

The results of this research did not reject the null hypothesis, $H_{1,0}$. The assessment means refer to the means of the control and treatment group *Perceived Difficulty of Programming Questionnaire* responses, multiple choice class test results and end-of-year IT summative assessment marks obtained by learners.

The second sub-hypothesis (H_2) evaluated the impact of the different PATs on IT learner motivation towards programming. The second sub-hypothesis was as follows:

$H_{2,0}$: There is no difference between the motivational strategy means of the control and treatment groups (ω_1=ω_2, ω_1 is the control group motivational strategy scores mean, ω_2 is the treatment group motivational strategy scores mean)

$H_{2,1}$: There is a difference between the motivational strategy means of the control and treatment groups ($\omega_1 \neq \omega_2$)

The results of this research did not reject the null hypothesis, $H_{2,0}$. The motivational strategy scores are obtained from the *Motivated Strategies for Learning Questionnaire* (MSLQ).

9.4 Research Contributions

This research has delivered contributions to the study of IT programming in SA secondary schools. The contributions are categorised as theoretical contributions (Section 9.4.1) and practical contributions (Section 9.4.2).

9.4.1 Theoretical Contributions

The theoretical contributions of this research study are the research design used to select (Section 9.4.1.1) and evaluate (Section 9.4.1.2) the PATs in SA secondary schools and the selection criteria used to select suitable PATs (Section 9.4.1.3). Findings from the evaluation of the PATs have identified criteria that a suitable PAT to support IT learners should address (Section 9.4.1.4).

9.4.1.1 Research Design: Selection of PATs

The PATs evaluated by IT learners were not randomly selected. The PATs were selected based on criteria related to the difficulties faced by IT learners in SA secondary schools. IT learners evaluate PATs that address the difficulties faced by IT learners (Figure 9.1). The selected PATs are identified using selection criteria formulated from the difficulties faced by IT learners.

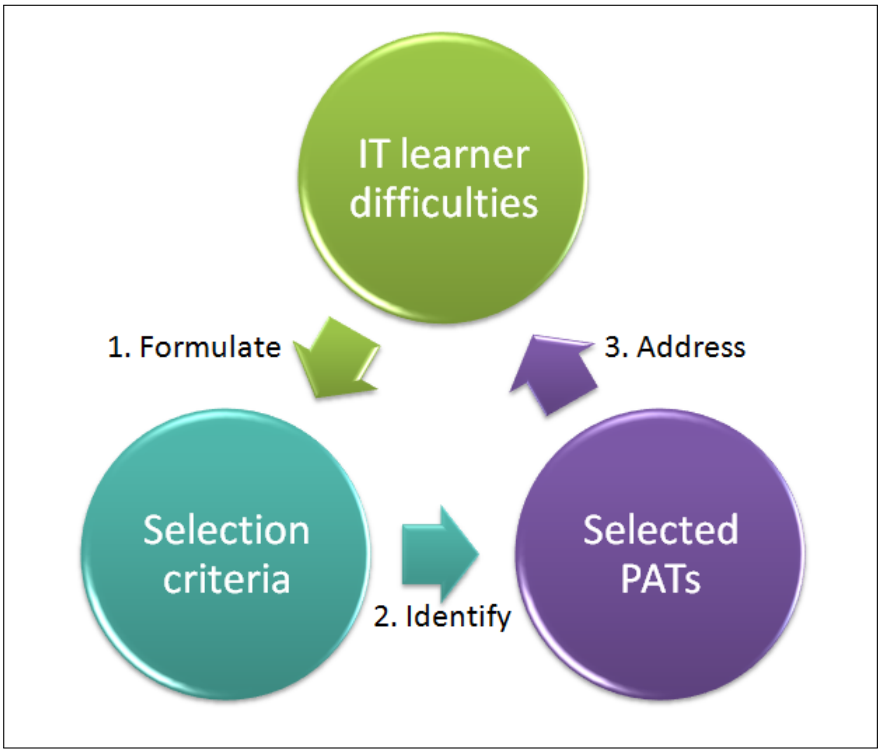


Figure 9.1: Process for the selection of PATs

This method can be generalised for use in any study to identify PATs for novice programmers. PATs suitable for the particular group of novice programmers can be identified using selection criteria derived from the difficulties faced by the particular group of novice programmers.

9.4.1.2 Research Design: Evaluation of PATs

A quasi-experimental approach for the evaluation of the PATs in SA secondary schools was utilised by this research. A non-equivalent groups design was used as the control and treatment group were not randomly assigned. Grade x of year y was the control group for the study, where $x \in \{10,11\}$. Learners from Grade x of year *y+1* received the PAT. Learners who utilised the PAT were included in the treatment group and learners who did not were included in the control group. Different questionnaires and multiple choice class tests were administered to Grade 10 and Grade 11 participants. Each school participating in the research was treated as a separate experiment as IT teachers and school learning environments were different. One PAT was allocated to the Grade 10 and Grade 11 learners at each school.

The consecutive years IT class for each grade, that is, *y* and *y+1*, were utilised instead of completing the study in one year as class sizes were too small to include both control and treatment groups. There was also no way to monitor cross-contamination between the control and treatment groups, specifically control group learners using the PATs.

The research design can be used to evaluate the impact of PATs or other treatments on IT learner understanding of programming concepts and motivation towards programming. Additional data collection instruments can be included as pre- and/or posttests. The success of the application of the quasi-experimental approach is dependent on the school at which it is applied. IT teacher work schedules, co-operation from IT learners to return questionnaires and utilisation of the PATs by treatment group learners, will impact the success of the study and the reliability of the results.

9.4.1.3 Selection Criteria

Selection criteria (Table 9.1) have been formulated (Section 4.4) that can be used to select suitable PATs to support IT learners. The criteria were formulated from a survey of introductory programming literature to identify programming difficulties faced by novice programmers. IT teacher and learner feedback were also used to identify programming difficulties faced by IT learners.

Table 9.1: Selection criteria for PATs. Criteria within each category (Programming concepts, knowledge and skills) listed in order of priority (highest first)

Category	**Criteria**	**Literature**	**Teacher**	**Learner**	**Weighting**
Concepts	Two-dimensional arrays	✓	✓	✓	18
	String handling	✓	✓	✓	17
	One-dimensional arrays	✓	✓	✓	16
	Procedures	✓	✓	✓	15
	Functions	✓	✓	✓	14
	repeat-loops	✓	✓	✓	13
	while-loops	✓	✓	✓	12
	Objects & classes	✓	✓		11
	for-loops	✓		✓	10
	if-statements	✓			9
	Correct use of parameters		✓	✓	8
	SQL statements		✓		7
	Accessing a database		✓		6
	case-statements		✓		5
	File handling			✓	4
	Variables				3
	Input (getting information from the user)				2
	Output (displaying information to the user)				1
Knowledge	Assists with the learning of the Delphi programming language syntax	✓	✓		5
	Assists with developing knowledge of programming principles & concepts	✓	✓		4
	Constructivist to promote self-study	✓	✓		3
	Assists with the application of programming knowledge	✓	✓		2
	Assists with the understanding of code execution	✓		✓	1
Skills	Promotes problem solving and planning	✓	✓	✓	5
	Provides simple error messages to assist with debugging	✓	✓	✓	4
	Develops code comprehension	✓		✓	3
	Feedback to guide solution creation		✓	✓	2
	Feedback regarding errors		✓		1

The selection criteria are categorised as programming knowledge, programming skills and programming concepts. The criteria in each of the the categories have been ranked in order of priority. The priority of the items was determined by common criteria identified from literature, IT teacher and IT learner feedback. The research has practically demonstrated a method of evaluating PATs using the selection criteria (Section 5.4).

The selection criteria can be used in future studies to evaluate PATs in terms of how they can assist IT learners to develop their understanding of programming concepts or programming knowledge, or develop the programming skills of IT learners. The selection criteria can be used to rank and compare several PATs (Section 5.4) that have been evaluated in terms of the programming concept, programming knowledge and programming skills selection criteria categories. The selection criteria can also be used as a guideline for designing and developing a new PAT to assist IT learners.

9.4.1.4 Recommendations for a Suitable PAT

The research has evaluated the impact of the selected PATs on IT learners understanding of programming concepts and motivation towards programming. Feedback from IT learners has identified positive and negative techniques used by the selected PATs. The feedback received provides the basis upon which more suitable PATs to support the learning of Delphi programming can be identified and/or developed.

The selection criteria should be part of the minimum requirements addressed by a suitable PAT. PATs should be fun but not too simple as IT learners want to be challenged. The use of graphics is entertaining and changes learner perceptions of programming. Typing textual code representations is not disliked by learners but graphical representations of program solutions, such as flow charts, assist learner understanding of concepts. If textual programs are used, it is important that the syntax matches Delphi so as not to confuse learners (Section 7.4).

9.4.2 Practical Contributions

This research study has three main practical contributions. Existing PATs have been evaluated to identify PATs suitable for use by IT learners (Section 9.4.2.1) and *RoboMind* was adapted to make it suitable for supporting the Delphi programming language (Section 9.4.2.2). The research has successfully evaluated the impact of the selected PATs on IT learner understanding of programming concepts and motivation towards programming (Section 9.4.2.3).

9.4.2.1 Evaluation of Selected PATs

Prior to this research, IT learners were not aware of and had not used PATs to assist their understanding of IT programming concepts. The aim of this research was to evaluate PATs that would be suitable for use by IT learners. No research had previously been done to evaluate PATs in SA schools.

Three PATs were selected and evaluated by IT learners. This research study has contributed by providing results on the impact of each of the PATs on IT learner understanding of programming concepts and motivation towards programming (Section 9.3.2). In addition, the techniques used by the PATs to create and represent programming solutions were also evaluated. The techniques were evaluated in terms of the impact on understanding of concepts and motivation to use the PAT and thus practice programming.

The new IT CAPS curriculum (Department of Basic Education, 2011) requires a PAT to be used by IT learners in Grade 10 before using Delphi in Grade 11. The evaluation of the PATs provides feedback for IT teachers to decide which PAT to use in Grade 10. In addition, the evaluation of the differences in the implementation of programming concepts in the PAT compared to Delphi (Section 6.2) allows IT teachers to be prepared for any knowledge gaps that may need to be addressed before the use of Delphi in Grade 11.

9.4.2.2 Adapted RoboMind for Delphi

RoboMind was successfully adapted (Section 6.2.2) so that the syntax of programs written to control the robot in the map world is equivalent to the Delphi programming language. The adapted *RoboMind* includes the same functionality as the unadapted version. The `begin` and `end` keywords are accepted in the adapted version to denote code blocks in place of the curly brackets. The adapted version also requires semicolons to delimit code statements.

The syntax of *if*-statements and *while*-loops were adapted to match the Delphi syntax. The unadapted version of *RoboMind* provides limited support for *for*-loops, using a `repeat` command. The command name was changed in the adapted version to `for1to`, although the functionality remains the same as in the unadapted version. In addition, the adapted version supports *repeat*-loops and provides limited support for variables (Section 6.2.2).

Evaluations of the adapted *RoboMind* indicate that learners are motivated towards programming in terms of ease of use and learnability. Adapted *RoboMind* also assists with the understanding of *repeat*- and *while*-loops. However, learners have indicated that it is confusing that the syntax does not match Delphi syntax exactly. Variables and *for*-loops do not match the Delphi syntax. Further adaptations and improvements are required before the adapted *RoboMind* can support the learning of Delphi by IT learners.

9.4.2.3 Research Results

The questionnaire and test results from IT learners were used to evaluate two research hypotheses (Section 2.3). Hypothesis H_1 was tested to determine the impact of the PATs on IT learner understanding of programming concepts. Based on the research results, the null hypothesis $H_{1,0}$ was not rejected. There was no conclusive evidence suggesting that learners using a PAT understood the programming concepts significantly better than those learners not using a PAT. This finding supports related research evaluating the impact of PATs used by novice programmers (Section 5.2.1).

The impact of PATs on IT learner motivation towards programming was evaluated by testing hypothesis H_2. The null hypothesis $H_{2,0}$ was also not rejected based on the *MSLQ* feedback from control and treatment group learners. The results did not provide evidence to suggest that learners using a PAT were significantly more motivated towards programming than those learners not using a PAT.

9.5 Limitations of the Research

The limitations of the research study are related to the use of IT learners in SA schools as participants. Only four schools participated in the research study. These were the schools, offering IT as a subject in the Port Elizabeth area, which consented to participate in the study.

Only schools in the Port Elizabeth area were considered due to proximity. Intensive interaction with the participating schools with regard to the administration of questionnaires and class tests was required, thus schools outside of the Port Elizabeth area were not considered. The research is limited by the fact that schools were not randomly selected provincially or nationally.

The participating schools all teach Delphi as the programming language. However, Delphi is not the only programming language taught in SA schools. Java is taught at schools in four of the nine provinces. The findings of this research are not generalisable to all IT learners in SA due to geographic and programming language restrictions.

The number of participating learners was influenced by the Grade 10 and Grade 11 IT class sizes at the participating schools. The use of the PAT by treatment group learners was also not enforced. This resulted in small (n<10) treatment group sample sizes for two of the PAT evaluations. The research results were further influenced by administrative issues at certain schools. Heavy teacher workloads and IT subject work schedules made it difficult to administer questionnaires and class tests. The onus was also on learners to return questionnaires to the IT teacher when completed.

Another problem encountered was that the research environment at the schools could not be controlled. At three of the schools, the same IT teacher taught both the control and treatment groups and it is thus assumed that the teaching environment was the same for both groups. The IT teacher did change at one of the schools evaluating *Scratch*. The results from this school can thus not be attributed to the use of *Scratch* alone.

9.6 Future Work

This research has produced several contributions in terms of the selection and evaluation of suitable PATs for IT learners. However, more work is needed to address the difficulties faced by IT learners. The contributions of this research form the basis for future work:

Extend the study: Limitations to the study (Section 9.5) have been highlighted with respect to the research population. Further research should be undertaken to repeat the study in more schools randomly selected throughout the country. The selection of PATs to support the Java programming language can also be evaluated.

Adaptations to RoboMind: This research adapted programming concepts supported in *RoboMind* to match the implementation in Delphi. The *for*-loops syntax was adapted and variables were implemented but the implementation of these two concepts did not match the exact implementation in Delphi. String handling and arrays were not implemented due to the restricted functionality of the robot in the map world. This research only implemented adaptations to the programming concepts. Adapting the actions performed by the robot in the microworld to support the need for input, output, string handling and/or mathematical calculations would address the limited functionality of *RoboMind*. The correct implementation of *for*-loops and variables, with the inclusion of data types, would address the syntax concerns of IT learners.

Develop a PAT for IT learners: This research has evaluated existing PATs in order to identify one suitable for use by IT learners. The IT learner feedback related to the techniques used by the PATs to support understanding of programming concepts and motivation towards programming, together with the selection criteria identified to assist with the understanding of programming concepts and knowledge and the development of programming skills, can be used to develop a PAT specifically designed for use by IT learners.

Usability Studies: This research did not evaluate the usability of the PATs. An evaluation of the gaze patterns of users using eye tracking as well as time to task and task completion would provide an evaluation of techniques and interfaces that support the learning of programming concepts and learner motivation towards programming. Eye tracking would identify specific techniques used by IT learners to improve programming concept knowledge.

9.7 Summary

The aim of this research study was to select and evaluate PATs that could address the difficulties faced by IT learners when learning to program. Three PATs, namely *RoboMind*, *Scratch* and *B#*, were selected using selection criteria derived from the novice programming difficulties and the difficulties faced by IT learners. *RoboMind* was adapted to support the Delphi programming language used by IT learners participating in the research study.

A quasi-experimental research approach was used to evaluate the selected PATs in SA schools. Based on the results from questionnaires, class tests and summative assessments administered to IT learners, there was no evidence to suggest that learners who used a PAT had a significantly better understanding of programming concepts or were significantly more motivated towards programming, than those learners who did not use a PAT. Feedback from IT learners provides reasons why the PATs were not used and what IT learners liked and disliked about the PATs.

The quantitative results together with the qualitative feedback from IT learners identified the shortcomings and strengths of the PATs evaluated. The evaluations of *RoboMind*, *Scratch* and *B#* provide suggestions and guidelines that can be used together with the selection criteria, to identify suitable PATs, other than the three selected and evaluated.

This research study has made several practical and theoretical research contributions. Theoretical contributions include the research design for the selection and evaluation of PATs, the selection criteria used to select suitable PATs for IT learners and suggested techniques that PATs should use to improve IT learners understanding of programming concepts and motivation towards programming. The practical contributions demonstrated by the research study include the evaluation of the suitability of existing PATs using the selection criteria, an adapted version of *RoboMind* suitable for use by IT learners, and the evaluation of the selected PATs.

The research was limited by the participating school environments such as IT teacher workload and small IT classes, resulting in incomplete questionnaire and class test data. In spite of these limitations, the research findings and contributions have formed the basis for future work. Extending the study by increasing the sample size of participating schools and the inclusion of usability studies would provide additional data making the results more generalisable. Further adaptations to *RoboMind* as well as the development of a new PAT specifically designed for IT learners based on the finding of the research, are also possible.

This research has made IT teachers and IT learners at the participating schools aware of the existence of PATs that can be used to support the understanding of programming concepts and improve learner motivation towards programming. This research has concluded that the PATs evaluated may not address all of the programming difficulties experienced by IT learners, however, IT teachers have been made aware of the strengths and shortcomings of these PATs. IT teachers have also been provided with a means of assessing the applicability of any PAT to the IT curriculum. This research has contributed and provided the basis of future work towards identifying and/or developing a PAT that IT learners are motivated to use and that can assist with their understanding of programming concepts, using the Delphi programming language.

List of References

Al-Imamy, S., Alizadeh, J. and Nour, M.A. (2006). On the development of a programming teaching tool: The effect of teaching by templates on the learning process. *Journal of Information Technology Education*, vol. 5, pp. 271–283.

Ala-Mutka, K. (2003). Problems in learning and teaching programming. In: *Codewitz Needs Analysis.* Institute of Software Systems, Tampere University of Technology.

Areias, C. and Mendes, A. (2007). A tool to help students to develop programming skills. In: *International Conference on Computer Systems and Technologies (CompSysTech'07).*

Artino, J. and Anthony, R. (2005). Review of the Motivated Strategies for Learning Questionnaire. Available at: `http://www.eric.ed.gov/ERICWebPortal/contentdelivery/servlet/ERICServlet?accno=ED499083` [Accessed 15 October 2009].

Baldwin, L.P. and Kuljis, J. (2000). Visualisation techniques for learning and teaching programming. *Journal of Computing and Information Technology*, vol. 4, pp. 285–291.

Bednarik, R. and Fränti, P. (2004). Survival of students with different learning preferences. In: *Proceedings of the Fourth Finnish/Baltic Sea conference on Computing Science Education*, pp. 121–125.

Bednarik, R. and Tukiainen, M. (2006). An eye-tracking methodology for characterizing program comprehension processes. In: *Proceedings of the 2006 symposium on Eye tracking research & applications (ETRA'06)*, pp. 125–132. New York, NY, USA.

Biggers, M., Brauer, A. and Yilmaz, T. (2008). Student perceptions of Computer Science: A retention study comparing graduating seniors vs. CS leavers. In: *Proceedings of the SIGCSE'08 Conference.*

Bonar, J. and Soloway, E. (1989). Preprogramming knowledge: A major source of misconceptions in novice programmers. In: Soloway, E. and Spohrer, J. (eds.), *Studying the Novice Programmer*, pp. 325–353. Hillsdale, NJ: Lawrence Erlbaum Associates.

Brewer, D. (2009). Control structures in programming: An explanation of the three basic structures found in programs. Available at: `http://dawn-brewer.suite101.com/control-structures-in-programming-a96234` [Accessed 20 September 2011].

Brooke, J. (1996). SUS: A “quick and dirty” usability scale. *Usability Evaluation in Industry*, pp. 189–194.

Bryant, R., Weiss, R., Orr, G. and Yerion, K. (2011). Using the context of algorithmic art to change attitudes in introductory programming. *Journal of Computing Sciences in Colleges*, vol. 27, pp. 112–119.

Byckling, P. and Sajaniemi, J. (2006). Roles of variables and programming skills improvement. *SIGCSE Bulletin*, vol. 38, no. 1, pp. 413–417.

Calitz, A. (2010). *A Model for the Alignment of ICT Education with Business ICT Skills Requirements.* DBA, Nelson Mandela Metropolitan University.

Chen, C. (2002). Self-regulated learning strategies and achievement in an introduction to Information Systems course. *Information Technology, Learning and Performance Journal*, vol. 20, no. 1, pp. 11–25.

Cohen, L., Manion, L. and Morrison, K. (2007). *Research Methods in Education.* 6th edn. Routledge.

Cooper, S., Dann, W. and Pausch, R. (2003). Teaching objects-first in introductory Computer Science. In: *Proceedings of the 34th SIGCSE technical symposium on Computer Science Education*, pp. 191–195. ACM, New York, NY, USA.

Creswell, J.W. (2009). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches.* Third edition edn. SAGE Publications.

Cross, J., Hendrix, T., Jain, J. and Barowski, L. (2007). Dynamic object viewers for data structures. *SIGCSE Bulletin*, vol. 39, no. 1, pp. 4–8.

Cumps, B., Viaene, S. and Dedene, G. (2010). Linking the strategic importance of ICT with investment in business-ICT alignment: An explorative framework. *International Journal on IT/Business Alignment and Governance*, vol. 1, no. 1, pp. 39–57.

de Raadt, M. (2008). *Teaching Programming Strategies Explicitly to Novice Programmers.* Doctoral dissertation, University of Southern Queensland.

Department of Basic Education (2011). Curriculum and Assessment Policy Statement Grades 10-12: Information Technology.

Department of Education (2003). National Curriculum Statement. Grades 10-12 (General). Information Technology.

Department of Education (2008). National Curriculum Statement. Grades 10-12 (General). Learning programme guidelines. Information Technology.

Egan, M. (2010). Recruitment of CS majors through a non-programmer’s programming contest. *Journal of Computing Sciences in Colleges*, vol. 25, no. 6, pp. 198–204.

Fidge, C. and Teague, D. (2009). Losing their marbles: Syntax-free programming for assessing problem-solving skills. In: Hamilton, M. and Clear, T. (eds.), *Eleventh Australasian Computing Education Conference (ACE 2009)*, pp. 75–82.

Fleming, N. (2010). VARK: A guide to learning styles. Available at: `http://www.vark-learn.com/english/index.asp` [Accessed 3 October 2009].

Fleming, N. and Baume, D. (2006). Learning styles again: VARKing up the right tree! *Educational Developments, UK, Staff and Educational Development Association (SEDA)*, vol. 7, no. 4, pp. 4–7.

Fleming, N.D. and Bonwell, C.C. (1997). VARK–Advice to users of the questionnaire. Available at: `http://www.ntlf.com/html/lib/suppmat/74vark2.htm` [Accessed 5 October 2009].

Gardner, M. and Feng, W. (2010). Broadening accessibility to Computer Science for K-12 education. In: *Proceedings of the 15th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'10)*, pp. 229–233. Bilent, Turkey.

Garner, S. (2007). A program design tool to help novices learn programming. In: *ICT: Providing choices for learners and learning*, pp. 321–324. Ascilite Singapore 2007.

Gayo-Avello, D. and Fernández-Cuervo, H. (2003). Online self-assessment as a learning method. In: *Proceedings of the 3rd IEEE International Conference on Advanced Learning Technologies*, pp. 254–255.

Gomes, A. and Mendes, A.J. (2007). An environment to improve programming education. In: *Proceedings of the 2007 international conference on Computer systems and technologies*, CompSysTech'07, pp. 88:1–88:6. ACM, New York, NY, USA. ISBN 978-954-9641-50-9.

Greyling, J., Cilliers, C. and Calitz, A. (2006). B#: The development and assessment of an iconic programming tool for novice programmers. In: *7th International Conference on Information Technology Based Higher Education and Training (ITHET'06)*, pp. 367–375.

Gribbons, B. and Herman, J. (1997). True and quasi-experimental designs. In: *Practical Assessment, Research and Evaluation*, vol. 5.

Guzdial, M. (2004). *CS Education Research*, chap. Programming Environments for Novices.

Haataja, A., Suhonen, J., Sutinen, E. and Torvinen, S. (2001). High school students learning Computer Science over the web. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning*, vol. 3, no. 2.

Havenga, M. and Mentz, E. (2009). The school subject Information Technology: A South African perspective. In: *Proceedings of SACLA'09*, pp. 77–83. Mpekweni Beach Resort, South Africa.

Heines, J. and Schedlbauer, M. (2007). Teaching object-oriented concepts through GUI programming. Eleventh Workshop on Pedagogies and Tools for the Teaching and Learning of Object Oriented Concepts, held at the 21st European Conf. on Object-Oriented Programming. Berlin, Germany.

Henrikson, P. and Kölling, M. (2004). Greenfoot: Combining object visualisation with interaction. In: *Companion to the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA'04)*, pp. 73–82.

Hooper, C., Carr, L., Davis, H., Millard, D., White, S. and Wills, G. (2007). AnnAnn and AnnAnn.Net: Tools for teaching programming. *Journal of Computers*, vol. 2, no. 2, p. 916.

Hu, C. (2008). Just say "A class defines a data type". In: *Communications of the ACM*, vol. 51(3), pp. 19–21.

Jones, R., Boyle, T. and Pickard, P. (2003). OBJECTWORLD: Helping novice programmers to succeed through a graphical objects-first approach. In: *Proceedings of 4th Annual LTSN-ICS Conference.*

Kelleher, C. and Pausch, R. (2005). Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, vol. 37, no. 2, pp. 88–137.

Kenny, D.A. (1975). A quasi-experimental approach to assessing treatment effects in the nonequivalent control group design. *Psychological Bulletin*, vol. 82, no. 3, pp. 345–362.

Kerman, M.C. (2002). *Programming and Problem Solving with Delphi.* Addison-Wesley Publishing Company, Inc.

Kölling, M. (1999). The problem of teaching object-oriented programming, Part 1: Languages. *Journal of Object-Oriented Programming*, vol. 11, no. 8, pp. 8–15.

Kölling, M. (2004). Bluej tutorial. Available at: `http://www.bluej.org/tutorial/tutorial-201.pdf` [Accessed 10 December 2010].

Kölling, M. and Rosenberg, J. (2001). Guidelines for teaching object orientation with Java. In: *Proceedings of the 6th conference on Information Technology in Computer Science Education (ITiCSE 2001).*

Kölling, M. and Rosenberg, J. (2002). *BlueJ - The Hitch-Hikers Guide to Object Orientation.* Mærsk McKinney Moller Institute for Production, University of Southern Denmark.

Kumar, A. (2006). Explanation of step-by-step execution as feedback for problems on program analysis, and its generation in model-based problem-solving tutors. *Technology, Instruction, Cognition and Learning (TICL) Journal.*

Kunkle, W.M. (2010). *The Impact of Different Teaching Approaches and Languages on Student Learning of Introductory Programming Concepts.* Thesis, Drexel University. Doctor of Philosophy.

Lahtinen, E., Ala-Mutka, K. and Järvinen, H. (2005). A study of the difficulties of novice programmers. In: *Proceedings of the 10th Annual SIGCSE Conference on innovation and Technology in Computer Science Education*, pp. 14–18. ACM, Caparica, Portugal.

Levy, R.B.-B., Ben-Ari, M. and Uronen, P.A. (2001). An extended experiment with Jeliot 2000. In: *Proc. First International Program Visualization Workshop*, pp. 131–140. University of Joensuu Press, Pavoo, Finland.

Lister, R., Adams, E., Fitzgerald, S., W.Fone, Hamer, J., Lindholm, M., McCartney, R., Moström, J., Sanders, K., Seppälä, O., Simon, B. and Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. In: *Working Group Reports From ITiCSE on innovation and Technology in Computer Science Education*, pp. 119–150. Leeds, United Kingdom.

Lister, R., Simon, B., Thompson, E., Whalley, J. and Prasad, C. (2006). Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. In: *Proceedings of the 11th Annual SIGCSE Conference on innovation and Technology in Computer Science Education*, pp. 118–122.

Lopez, M., Whalley, J., Robbins, P. and Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. In: *Proceeding of the Fourth international Workshop on Computing Education Research*, pp. 101–112.

Ma, L., Ferguson, J., Roper, M., Ross, I. and Wood, M. (2008). Using cognitive conflict and visualisation to improve mental models held by novice programmers. In: *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, pp. 342–346.

Malan, D. and Leitner, H. (2007). Scratch for budding Computer Scientists. In: *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'07)*, pp. 223–227.

Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B. and Resnick, M. (2004). Scratch: A sneak preview. In: *Second International Conference on Creating, Connecting, and Collaborating through Computing*, pp. 104–109.

Mannila, L. and de Raadt, M. (2006). An objective comparison of languages for teaching introductory programming. In: *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006*, vol. 276, pp. 32–37.

Medium Term Strategic Framework (2009). *Together Doing More and Better: A Framework to Guide Government's Programme in the Electoral Mandate Period (2009-2014)*. SA Government.

Mertens, D. (2004). *Research and Evaluation in Education and Psychology: Integrating Diversity with Quantitative and Mixed Methods*. Second edition edn. SAGE Publications.

Moreno, A., Myller, N. and Bednarik, R. (2005). Jeliot 3, an extensible tool for program visualization. In: *Proceedings of the Koli Calling 2005: 5th Annual Finnish / Baltic Sea Conference on Computer Science Education*.

NACE (2011). Spring 2011 salary survey report. Available at: `http://www.naceweb.org/Press/Releases/Top-Paid_Majors_for_the_Class_of_2011.aspx` [Accessed 11 October 2011].

NMMU (2011). Nelson Mandela Metropolitan University: Admission requirements. Available at: `http://www.nmmu.ac.za/default.asp?id=5888\&bhcp=1` [Accessed 21 October 2011].

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M. and Paterson, J. (2007). A survey of literature on the teaching of introductory programming. In: *Working group reports on ITiCSE on Innovation and technology in computer science education*, ITiCSE-WGR '07, pp. 204–223. ACM, New York, NY, USA.

Pendergast, M. (2006). Teaching introductory programming to IS students: Java problems and pitfalls. *Journal of Information Technology Education*, vol. 5, pp. 491–515.

Phillips, D.C. and Burbules, N.C. (2000). *Postpositivism and Educational Research*. Rowman & Littlefield Publishers, Inc.

Pintrich, R., Smith, A., Garcia, T. and McKeachie, W. (1991). *A Manual for the Use of the Motivated Strategies for Learning Questionnaire (MSLQ)*. National Center for Research to Improve Postsecondary Teaching and Learning, Ann Arbor, MI: University of Michigan Press.

Porta, M., Maillet, K. and Gil, M. (2010). Dec-CS: The Computer Science declining phenomenon. In: *Proceedings of the World Congress on Engineering and Computer Science 2010 (WCECS'2010)*, vol. 2. San Francisco, USA.

Putnam, R., Sleeman, D., Baxter, J. and Kuspa, L. (1989). A summary of misconceptions of high-school BASIC programmers. In: Soloway, E. and Spohrer, J. (eds.), *Studying the novice programmer*, pp. 301–314. Hillsdale, NJ: Lawrence Erlbaum Associates.

Rajala, T., Laakso, M., Kailo, E. and Salakoski, T. (2007). VILLE: A language-independent program visualization tool. In: *Conferences in Research and Practice in Information Technology*, vol. 88.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. and Kafai, Y. (2009). Scratch: Programming for all. In: *Communications of the ACM*, vol. 52, pp. 60–67.

Robins, A., Rountree, J. and Rountree, N. (2003). Learning and teaching programming: A review and discussion. In: *Computer Science Education*, vol. 13, pp. 137–172.

RoboMind (2009). RoboMind website. Available at: `http://www.robomind.net/en/index.html` [Accessed 13 March 2010].

Rogerson, C. and Scott, E. (2010). The fear factor: How it affects students learning to program in a tertiary environment. *Journal of Information Technology Education*, vol. 9, pp. 141–171.

Rongas, T., Kaarna, A. and Kalvianen, H. (2004). Classification of computerized learning tools for introductory programming courses: Learning approach. In: *Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT'04)*, pp. 678–680.

Saunders, M., Lewis, P. and Thornhill, A. (2006). *Research Methods for Business Students.* Fourth edition edn. Pearson Education Limited.

Seaton, C.G. (2007). *A Programming Language Where the Syntax and Semantics Are Mutable at Runtime.* Doctoral Dissertation, Department of Computer Science, University of Bristol.

Shuhidan, S., Hamilton, M. and D'Souza, D. (2009). A taxonomic study of novice programming summative assessment. In: *Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95*, ACE '09, pp. 147–156. Australian Computer Society, Inc., Darlinghurst, Australia, Australia. ISBN 978-1-920682-76-7.

Slater, J., Lujan, H. and DiCarlo, S. (2007). Does gender influence learning style preferences of first-year medical students? In: *Advances in Physiology Education*, vol. 31, pp. 336–342.

Slonneger, R. and Kurtz, B. (1995). *Formal Syntax and Semantics of Programming Languages.* Addison-Wesley Publishing Company, Inc.

Strieber, A. (2011). The 10 best jobs of 2011. Available at: `http://www.careercast.com/jobs-rated/10-best-jobs-2011` [Accessed 10 August 2011].

SUN (2011). Stellenbosch University: Admission requirements. Available at: `http://sun025.sun.ac.za/portal/page/portal/Maties/English/Admission\_requirements` [Accessed 26 October 2011].

Teague, D. and Roe, P. (2008). Collaborative learning - towards a solution for novice programmers. In: Simaon and Hamilton, M. (eds.), *Conferences in Research and Practice in Information Technology*, vol. 78.

Tew, A.E., McCracken, W.M. and Guzdial, M. (2005). Impact of alternative introductory courses on programming concept understanding. In: *Proceedings of the first international workshop on Computing education research*, ICER '05, pp. 25–35. ACM, New York, NY, USA. ISBN 1-59593-043-4.

Trochim, W. (2006). Research methods knowledge base. Available at: `http://socialresearchmethods.net/kb/index.php` [Accessed 6 May 2010].

UCT (2011). University of Cape Town: Admission requirements. Available at: `http://www.uct.ac.za/apply/criteria/nsc/` [Accessed 24 October 2011].

Utting, I., Cooper, S., Kölling, M., Maloney, J. and Resnick, M. (2010). Alice, Greenfoot, and Scratch - A discussion. *ACM Transactions on Computing Education*, vol. 10, no. 4.

Vainio, V. and Sajaniemi, J. (2007). Factors in novice programmer's poor tracing skills. In: *ITiCSE'07*, pp. 236–240. Dundee, Scotland.

Vickers, P. (2009). *How to Think Like a Programmer*. Cengage Learning EMEA.

Wehrwein, E., Lujan, H. and DiCarlo, S. (2007). Gender differences in learning style preferences among undergraduate physiology students. In: *Advanced Physiology Education*, vol. 31, pp. 153–157.

Williams, L., Wiebe, E., Yang, K., Ferzli, M. and Miller, C. (2002). In support of pair programming in the introductory computer science course. *Computer Science Education*, vol. 12, pp. 197–212.

Wilson, C., Sudol, L., Stephenson, C. and Stehlik, M. (2010). Running on empty: The failure to teach K-12 Computer Science in the digital age. Tech. Rep., ACM.

Wood, D., Bruner, J. and Ross, G. (1976). The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry and Allied Disciplines*, vol. 17, pp. 89–100.

Appendices

Appendix A

Questionnaires to IT Teachers

A.1 Open Questionnaire to IT Teachers

Questionnaire to IT Teachers
Teaching Environment and Challenges

The aim of this questionnaire is to gain a better understanding of current IT teaching environments and the difficulties currently faced by IT teachers. Please complete the following questions as honestly as possible to provide us with a better understanding of IT teaching environments and how future research may be of assistance. Where possible please explain your answers.

Teacher:	

1. There is currently a debate amongst IT teachers. The following are some of the points raised:
   - IT is aimed at producing programmers – most of the time is spent trying to cover the programming content while skimming over most of the other content
   - IT learners are not exposed to many other of the latest technologies and fields related to computers. Web applications, mobile applications and graphics are just some of the topics that learners should be exposed to. Learners should also get a better idea of the different professions and careers related to computing – that it is not all about programming.
   - IT is about development, therefore programming is important.
   - If the amount of content detail is reduced but learners are exposed to more concepts/topics, learners will only end up with a superficial knowledge and not have adequate skills in a particular concept area.
   - The main question that needs to be answered is: What knowledge should an IT learner at the end of Grade 12 have?

   Considering the statements above, what are your views on the current IT curriculum? Are a few minor changes sufficient?

2. In your opinion, what are the challenges you face as an IT teacher that other subject teachers may not have to deal with?

3. What is easier for you as the IT teacher compared to other subjects?

4. How would you compare IT to other subjects in terms of difficulty and amount of content that needs to be covered?

5. What do you consider to be the most challenging topic/concept/learning outcome with regards to IT?

6. What do you consider to be the easiest topic/concept/learning outcome with regards to IT?

7. What is your opinion with regards to IT textbooks or support material? (Do you use one or several textbooks when presenting IT? Do the textbooks/study guides available support the teaching of IT?)

8. Do you believe that the allocated school periods (time available) and timetable affects the teaching of IT (yes/no)? If yes, please explain? Do you think IT is more affected than other subjects?

9. What strategy do you use during practical programming lessons to assist learners and determine which learners are struggling?

10. Do you believe that learners have enough practical programming time in class to gain a basic but thorough understanding of the concepts being covered? Please feel free to explain your answer.

11. Do you believe that IT adequately prepares learners for a career in the Computing field?

12. Any other related comments/opinions/suggestions?

Thank you for time and assistance.

A.2 Questionnaire to IT Teachers

Questionnaire to IT Teachers
Teaching methods and use of Programming Assistance Tools

The aim of this questionnaire is to gain a better understanding of current IT teaching environments and the difficulties currently faced by IT teachers. Please complete the following questions as honestly as possible to provide us with a better understanding of IT teaching environments and how future research may be of assistance.

Teacher:		
School:		
Province:		
Programming language used: (mark with X)	Delphi	Java

Number of years teaching IT (or Computer Science HG):		
0-5 years	6-10 years	More than 10 years

1. Provide a rating to describe how difficult it is to **teach** each of the following programming concepts. Choose a value from 1 (extremely easy) to 7 (extremely difficult). Indicate with 0 if you have never taught the concept before.

	Concept	Rating							
1	Variables	⓪	①	②	③	④	⑤	⑥	⑦
2	Input (Getting information from the user)	⓪	①	②	③	④	⑤	⑥	⑦
3	Output (Displaying information to the user)	⓪	①	②	③	④	⑤	⑥	⑦
4	If statements	⓪	①	②	③	④	⑤	⑥	⑦
5	Case (Delphi) or switch(Java) statements	⓪	①	②	③	④	⑤	⑥	⑦
6	For Loops	⓪	①	②	③	④	⑤	⑥	⑦
7	While loops	⓪	①	②	③	④	⑤	⑥	⑦
8	Repeat (Delphi) or do While (Java) loops	⓪	①	②	③	④	⑤	⑥	⑦
9	String handling	⓪	①	②	③	④	⑤	⑥	⑦
10	One dimensional arrays	⓪	①	②	③	④	⑤	⑥	⑦
11	Two dimensional arrays	⓪	①	②	③	④	⑤	⑥	⑦
12	File handling	⓪	①	②	③	④	⑤	⑥	⑦
13	Accessing a database	⓪	①	②	③	④	⑤	⑥	⑦
14	SQL statements	⓪	①	②	③	④	⑤	⑥	⑦
15	Procedures	⓪	①	②	③	④	⑤	⑥	⑦
16	Functions	⓪	①	②	③	④	⑤	⑥	⑦

17	Correct use of parameters	⓪	①	②	③	④	⑤	⑥	⑦
18	Objects & classes	⓪	①	②	③	④	⑤	⑥	⑦
19	Problem solving	⓪	①	②	③	④	⑤	⑥	⑦
20	Algorithms	⓪	①	②	③	④	⑤	⑥	⑦
21	Planning (use of pseudocode to plan solution before coding)	⓪	①	②	③	④	⑤	⑥	⑦
22	Debugging (finding errors in the code)	⓪	①	②	③	④	⑤	⑥	⑦

2. Provide a rating to describe how difficult it is for learners to **<u>understand</u>** each of the following programming concepts. Choose a value from 1 (extremely easy) to 7 (extremely difficult). Indicate with 0 if you have never taught the concept before.

	Concept	Rating							
1	Variables	⓪	①	②	③	④	⑤	⑥	⑦
2	Input (Getting information from the user)	⓪	①	②	③	④	⑤	⑥	⑦
3	Output (Displaying information to the user)	⓪	①	②	③	④	⑤	⑥	⑦
4	If statements	⓪	①	②	③	④	⑤	⑥	⑦
5	Case (Delphi) or switch(Java) statements	⓪	①	②	③	④	⑤	⑥	⑦
6	For Loops	⓪	①	②	③	④	⑤	⑥	⑦
7	While loops	⓪	①	②	③	④	⑤	⑥	⑦
8	Repeat (Delphi) or do While (Java) loops	⓪	①	②	③	④	⑤	⑥	⑦
9	String handling	⓪	①	②	③	④	⑤	⑥	⑦
10	One dimensional arrays	⓪	①	②	③	④	⑤	⑥	⑦
11	Two dimensional arrays	⓪	①	②	③	④	⑤	⑥	⑦
12	File handling	⓪	①	②	③	④	⑤	⑥	⑦
13	Accessing a database	⓪	①	②	③	④	⑤	⑥	⑦
14	SQL statements	⓪	①	②	③	④	⑤	⑥	⑦
15	Procedures	⓪	①	②	③	④	⑤	⑥	⑦
16	Functions	⓪	①	②	③	④	⑤	⑥	⑦
17	Correct use of parameters	⓪	①	②	③	④	⑤	⑥	⑦
18	Objects & classes	⓪	①	②	③	④	⑤	⑥	⑦
19	Problem solving	⓪	①	②	③	④	⑤	⑥	⑦
20	Algorithms	⓪	①	②	③	④	⑤	⑥	⑦
21	Planning (use of pseudocode to plan solution before coding)	⓪	①	②	③	④	⑤	⑥	⑦
22	Debugging (finding errors in the code)	⓪	①	②	③	④	⑤	⑥	⑦

Teaching Style:

3. My teaching style is very constructivist[1] in nature.

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

4. I introduce new concepts to learners by providing learners with a reading assignment to first read about the new concepts themselves.

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

5. I provide learners with and explain **several** (more than 4) code examples demonstrating the new concepts before learners code themselves

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

6. I introduce all variations/aspects of a concept to the learners before any exercises. For example, if teaching If statements, I will explain *if, if-else*, *nested if* and the use of logical operators, all before the first exercise.

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

Programming Concept Exercises:

7. Most of the programming exercises that learners must complete are for homework.

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

8. I provide learners with solutions to exercises

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

9. I check learners' exercise solutions only if asked to by the learner

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

10. Most of the programming section class time is allocated for programming exercises

[1] A "constructivist" teaching style allows learners to be responsible for building their own knowledge. The teaching environment and lessons plans promote learner interaction and participation as opposed to the traditional "lecturing" teaching style.

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

11. I make sure learners understand the theoretical background of a concept before they begin with exercises.

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

12. Most learners that struggle do understand the concepts

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

13. Most learners that struggle have difficulty applying concepts to different problems

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

14. There is enough time in class to ensure that each learner understands the concepts satisfactorily

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

15. Many learners that understand the concepts struggle in examinations/tests

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

Debugging:

16. Most learners understand the compiler error messages

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

17. Most learners know how to use compiler error messages to debug their code.

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

18. I teach learners how to use the debugging features of the IDE such as breakpoints and stepping over and into code, to find syntax and logical errors in their code.

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

19. I know how to use the debugging features of the IDE such as breakpoints and stepping over and into code.

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

20. Many learners that struggle have difficulty with the programming language syntax

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

Programming Assistance Tools / Software

Definition: A programming assistance tool is an additional software program (not the programming IDE eg. Netbeans or Delphi 6/7) that demonstrates to learners how code executes, how variable values change, identifies syntax errors in a simple manner or allows learners to code using graphical elements. Some of programming tools use animations to explain concepts. Examples are programming assistance tools are Alice 3D, Jeliot 2000, GreenFoot, Ville, JGrasp, LOGO, etc.

21a. Have you used or are you currently using programming assistance tools to help students understand the programming concepts?

Yes	No

21b. If yes, which programming assistance tool(s) have you used or do you currently use?

If you answered **Yes** to 21a, please continue with question 24, else also answer questions 22 and 23.

22. I would like to use a programming assistance tool to help students understand the concepts better

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

23 I don’t personally have experience with any programming assistance tools that can be used

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

24 There is enough time in class for students to learn how to use a programming assistance tool to improve their understanding of programming concepts.

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree	Not Applicable

Thank you for time and assistance.

Appendix B

IT Decision Questionnaire

B.1 Pretest Questionnaire

Nelson Mandela Metropolitan University

for tomorrow

General Questionnaire to IT Learners (Grade 10)

Please complete the following questions regarding your decision to take IT. We would like to gather more information about learner interest in IT. We would appreciate your co-operation by completing the following questions. Remember that these are your honest views; there are no right or wrong answers.

Learner:			
Grade:		School:	

Programming Language (Mark with an X):	Delphi	Java

1. Which of the following statements reflect factors present in your decision to take IT as a subject (you may select more than one):

1	IT was one of my first choice subjects	
2	I had to decide between IT and another subject	
3	Taking IT was my own decision	
4	IT was recommended/suggested by someone else	
5	If IT was taught by another teacher I might not have selected it	
6	I enjoy working on a computer (e.g. games, typing, creating documents)	
7	I have some knowledge of programming or what it is all about	
8	I am interested in programming	

2. Do you enjoy Mathematics?

Yes		No	

3. Do you enjoy solving problems or puzzles?

Yes		No	

4. Did anyone influence your decision to take IT?

Yes		No	

If **yes**, please select one or more people who influenced your decision from the list below:

1	IT teacher	
2	Guidance teacher	
3	Friends deciding to take IT	
4	Friends/siblings already doing or have done IT	
5	Parents/guardians	
6	IT professional (someone you know or have seen in the IT industry)	

7	Other: (please specify)	

5. My **main** reason for selecting IT is (please select only one):

1	To understand more about the different aspects and topics related to computing in general	
2	To be able to understand how a computer works (hardware)	
3	To improve my understanding and use of different computer programs or application	
4	To be able to develop my own software programs	
5	To play LAN games with friends in class	
6	To have more access to a computer at school	
7	Other: (please specify)	

6. I think IT is a difficult subject if compared to other school subjects

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

7. I think I can do well in IT

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

8. I am considering a career in computers after school?

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

9. At the moment, what is your first choice career when you leave school?

Tick here if you have do not currently have one definite career in mind:	

Thank you.

B.2 Posttest Questionnaire

Nelson Mandela
Metropolitan
University

for tomorrow

General Questionnaire to IT Learners (Grade 10)

Please complete the following questions regarding your decision to take IT. We would like to gather more information about learner interest in IT. We would appreciate your co-operation by completing the following questions. Remember that these are your honest views; there are no right or wrong answers.

Learner:			
Gender (M/F):		School:	
Programming Language (Mark with an X):		Delphi	Java

1. If you could choose your Grade 10 subjects again, would you still choose IT?

Yes		No	

Please motivate your answer:

2. I think IT is a difficult subject if compared to other school subjects

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

3. I think I can do well in IT

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

4. I am considering a career in computers after school?

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

5. At the moment, what is your first choice career when you leave school?

Tick here if you have do not currently have one definite career in mind:	

Thank you.

Appendix C

Perceived Difficulty of IT Questionnaire (Grade 11)

Nelson Mandela
Metropolitan
University
for tomorrow

General Questionnaire to IT Learners

Please complete the following questions regarding programming in IT. We would like to gather more information about the difficulties faced by IT learners. We would appreciate your co-operation by completing the following questions. Remember that these are your honest views on IT programming; there are no right or wrong answers.

Learner:			
Grade:		School:	
Programming Language (Mark with an X):		Delphi	Java

1. Provide a rating for each of the following programming concepts. Choose a value from 1 (extremely easy) to 7 (extremely difficult). Indicate with 0 if the concept has not been covered or taught yet.

	Concept	Rating							
1	Variables	⓪	①	②	③	④	⑤	⑥	⑦
2	Input (Getting information from the user)	⓪	①	②	③	④	⑤	⑥	⑦
3	Output (Displaying information to the user)	⓪	①	②	③	④	⑤	⑥	⑦
4	If statements	⓪	①	②	③	④	⑤	⑥	⑦
5	Case (Delphi) or switch(Java) statements	⓪	①	②	③	④	⑤	⑥	⑦
6	For Loops	⓪	①	②	③	④	⑤	⑥	⑦
7	While loops	⓪	①	②	③	④	⑤	⑥	⑦
8	Repeat (Delphi) or do While (Java) loops	⓪	①	②	③	④	⑤	⑥	⑦
9	String handling	⓪	①	②	③	④	⑤	⑥	⑦
10	One dimensional arrays	⓪	①	②	③	④	⑤	⑥	⑦
11	Two dimensional arrays	⓪	①	②	③	④	⑤	⑥	⑦
12	File handling	⓪	①	②	③	④	⑤	⑥	⑦
13	Accessing a database	⓪	①	②	③	④	⑤	⑥	⑦
14	SQL statements	⓪	①	②	③	④	⑤	⑥	⑦
15	Procedures	⓪	①	②	③	④	⑤	⑥	⑦
16	Functions	⓪	①	②	③	④	⑤	⑥	⑦
17	Correct use of parameters	⓪	①	②	③	④	⑤	⑥	⑦
18	Objects & classes	⓪	①	②	③	④	⑤	⑥	⑦
19	Problem solving	⓪	①	②	③	④	⑤	⑥	⑦
20	Algorithms	⓪	①	②	③	④	⑤	⑥	⑦
21	Planning (use of pseudocode to plan solution before coding)	⓪	①	②	③	④	⑤	⑥	⑦
22	Debugging (finding errors in the code)	⓪	①	②	③	④	⑤	⑥	⑦

2. Choose **one** concept from the list above that you find the most difficult to understand (Give the number only, eg. 4 for If Statements): □

3. I am able to do simple exercises for a programming concept

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

4. I understand simple examples explaining how to apply a programming concept

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

5. I understand solutions provided for simple exercises

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

6. Most times I understand the theory explaining a programming concept

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

7. I am able to combine different programming concepts to solve a complex problem

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

8. I am able to apply programming concepts in larger, more complex exercises

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

9. I understand solutions provided for complex exercises

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

10. I am able to apply solutions from simple exercises to more complex exercises

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

11. I understand compiler error messages

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

12. I use compiler error messages to find syntax errors in my code

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

13. I use *showmessage* or similar to display variable values at critical points in my program to determine if the program is processing the data correctly

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

14. I use breakpoints in my code to evaluate variable values to determine if the program is processing the data correctly

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

15. I use different input values to test that my program will work for all possible user inputs

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

16. Most times I understand what is required for a particular programming question i.e. what the program is supposed to do

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

17. I am able to write the code required for to solve the programming questions i.e. write the solution in a way that the computer can understand it

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

18. I work out a solution (in any form) to a problem before coding

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

19. I write down a non-code solution before trying to write the code

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

20. I use comments in my program code to describe what different sections of the code are doing

1 Strongly Disagree	2	3	4	5	6	7 Strongly Agree

Thank you. Please complete the MSLQ questionnaire.

Appendix D

Multiple Choice Tests

D.1 Grade 10: *if*-statements

If Statement Questions

The following properties of *if* statements are to be evaluated:

Marking Sheet:

1 b
2 a
3 d
4 b
5 a
6 a
7 a
8 d
9 d
10 b
11 e
12 d (3marks)

If-Statements **Grade 10** | ***Multiple Choice Class Test*** ***25 Marks***

Name:

Please write your answer in the space provided on the right. Use the following responses if you are unable to select an answer:

X	Do not understand question
Y	Concept not covered in class
Z	Just don't know the answer

1. Consider the following code fragment:

```
iNum1 := 5;
iNum2 := 0;
if iNum1 > 0 then
  iNum2 := 1;
```

What value is in the variable *iNum2* after the code is executed?

a) 0   b) 1   c) 2   d) 5

1

2. Consider the following code fragment:

```
iNum1 := 0;
iNum2 := 4;
if iNum2 > 4 then
begin
  iNum1 := iNum1 + 1;
  iNum2 := iNum2 * 2;
end;
```

After the above code segment executes the values of *iNum1* and *iNum2* will be:

a) iNum1 = 0; iNum2 = 4   b) iNum1 = 1; iNum2 = 4
c) iNum1 = 0; iNum2 = 8   d) iNum1 = 1; iNum2 = 8

2

3. Consider the following code fragment:

```
iNum := 1;
if iNum <= 10 then
  iNum := iNum + 10;
```

3

After the above code segment finishes the value of *iNum* will be:

a) 1   b) 9   c) 10   d) 11

4. Consider the following code fragment:

```
iNum := 21;
if iNum < 21 then
  memOut.lines.add('underage')
else
  memOut.lines.add('adult');
```

What will be printed in the memobox component?

4

a) `underage`   b) `adult`

5. Consider the following code fragment:

```
iNum1 := 5;
iNum2 := 7;
if ((iNum1+10) > (iNum2*2)) then
  memOut.lines.add(‘greater than’)
else
  memOut.lines.add(‘less than’);
```

5

What will be printed in the memobox component?

a) greater than b) less than

6. The following code segment should check whether the number *n* is even or odd:

```
if (????????) then
  memOut.lines.add(‘Odd’)
else
  memOut.lines.add(‘Even’);
```

The ???????? in the above code should be replaced by:

6

```
a)  (iNum mod 2 = 1)            b)  (iNum mod 2 = 0)
c)  (iNum div 2 = 0)            d)  (iNum div 2 = 1)
```

7. Consider the following code fragment:

```
if (iNum > 40) and (iNum <= 50) then
  memOut.lines.add(‘Invalid’);
```

7

Which one of the following values of *iNum* will **not** print “Invalid”:

a) 40 b) 41 c) 50 d) None of the above

8. Consider the following code fragment:

```
if (iNum > 40) or (iNum <= 50) then
  memOut.lines.add(‘Invalid’);
```

8

Which one of the following values of *iNum* will **not** print “Invalid”:

a) 40 b) 41 c) 50 d) None of the above

9. The following code segment should take a number *iNum* and calculate the value that is half of *iNum*. The code should print “Even” if the value half of *iNum* is even and “odd” if the value half of *iNum* is odd.

```
if (?????????) then
  memOut.lines.add(‘Even’)
else
  memOut.lines.add(‘Odd’);
```

The ????????? in the above code should be replaced by:

9

```
a)  (iNum mod 2 = 0)            b)  (iNum div 2 = 0)
c)  ((iNum div 2) mod 2)        d)  ((iNum div 2) mod 2 = 0)
```

10. Consider the following code fragment:

```
if (iNum > 100) then
begin
  if (iNum < 200) then
    memOut.lines.add(‘A’)
  else
    memOut.lines.add(‘B’);
end
else
  if (iNum > 50) then
    memOut.lines.add(‘C’)
  else
    memOut.lines.add(‘D’);
```

If the value of *iNum* is 200, what will be printed in the memobox component:

a) A b) B c) C d) D

10

11. Consider the following code fragment:

```
if (iNum >= 100) then
begin
  if (iNum < 200) then
    memOut.lines.add(‘A’)
  else
    memOut.lines.add(‘B’);
end;
  if (iNum > 50) then
    memOut.lines.add(‘C’)
  else
    memOut.lines.add(‘D’);
```

If the value of *iNum* is 100, what will be printed in the memobox component:

a) A b) B c) C d) D e) A
C

11

12. Code is required that will use an integer value *iNum* (where 0<=iNum<=100) and print a message based on the value of iNum in a memobox:

iNum	Message
0 – 49	Below average
51-100	Above average
50	Average

Which one of the following code segments will **NOT** print the correct messages:

```
a) if (iNum <= 49) then
       memOut.lines.add(‘Below average’)
   else
     if (iNum=50) then
       memOut.lines.add(‘Average’)
     else
       memOut.lines.add(‘Above average’);
```

```
b) if (iNum > 50) then
     memOut.lines.add(‘Above average’)
   else
     if (iNum < 50) then
       memOut.lines.add(‘Below average’)
     else
       memOut.lines.add(‘Average’)

c) if (iNum=50) then
     memOut.lines.add(‘Average’)
   else
     if (iNum > 50) then
       memOut.lines.add(‘Above average’)
     else
       memOut.lines.add(‘Below average’)

d) if (iNum >= 50) then
     memOut.lines.add(‘Average’)
   else
     if (iNum < 50) then
       memOut.lines.add(‘Below Average’)
     else
       memOut.lines.add(‘Above Average’);
```

12

D.2 Grade 10: *for*-loops

***For* Loops**

Marking Sheet:

1 d
2 d
3 c
4 c
5 d
6 c
7 a
8 c

For Loops ***Grade 10***	***Multiple Choice Class Test*** ***16 Marks***
Name:	

Please write your answer in the space provided on the right. Use the following responses if you are unable to select an answer:

X	Do not understand question
Y	Concept not covered in class
Z	Just don't know the answer

1. Consider the following code fragment:

```
for iCounter := 1 to 5 do
  iNumber := iCounter;
```

What value is in the variable *iNumber* after the code is executed?

a) 1 b) 3 c) 4 d) 5

1 □

2. Consider the following code fragment:

```
iSum := 0;
for iCounter := 2 to 8 do
begin
  iSum := iSum + iCounter;
  iNum := iCounter;
end;
```

After the above code segment finishes the values of *iSum* and *iNum* will be:

a) iNum = 2; iSum = 8 b) iNum = 8; iSum = 8
c) iNum = 2; iSum = 35 d) iNum = 8; iSum = 35

2 □

3. Consider the following code fragment:

```
for iCounter := 3 to 9 do
  memOut.lines.add('hello');
```

How many times will "hello" be printed in the memobox component?

a) 3 b) 6 c) 7 d) 9

3 □

4. Consider the following code fragment:

```
for iStep1 := 1 to 5 do
  for iStep2 := 1 to 3 do
    memOut.lines.add(intToStr(iStep1) + 'x' + intToStr(iStep2));
```

What will the **last line** printed in the memobox component be?

a) iStep1 x iStep2 b) iStep1 + x + iStep2
c) 5 x 3 d) 15

4 □

5. Consider the following code fragment:

```
for iStep := 0 to 5 do
  memOut.lines.add(‘apples’);
memOut.lines.add(‘oranges’);
```

How many times are the words “apples” and “oranges” printed in the memobox component respectively?

a) apples = 5; oranges = 5 b) apples = 6; oranges = 6

c) apples = 5; oranges = 1 d) apples = 6; oranges = 1

5

6. The following pattern should be printed in a memobox component:

```
4x2=8
3x3=9
2x4=8
1x5=5
```

The following code segment is intended to print the pattern in the memobox component:

```
for a := 4 downto 1 do
  for ????????
    memOut.lines.add(intToStr(a)+’x’+intToStr(b)+’=’+intToStr(a*b));
```

The ???????? in the above code should be replaced by:

a) `b := 1 to 5 do` b) `b := 1 to 4 do`

c) `b := 2 to 5 do` d) `b := 2 to 4 do`

6

7. Consider the following code fragment:

```
iNumA := 10;
iNumB := 3;
iNumC := 4;
for iStep := iNumA to iNumB do
  iNumC := iNumC + 1;
```

7

After the above code segment finishes the value of *iNumC* will be:

a) 4 b) 10 c) 11 d) 12

8. The following code segment should print all the **even** numbers from 100 to 200 in a memobox component:

```
for iCounter := ???????? do
begin
  iEvenNumber := iCounter * 2;
  memOut.lines.add(intToStr(iEvenNumber));
end;
```

The ???????? in the above code should be replaced by:

8

a) `100 to 200` b) `1 to 100`

c) `50 to 100` d) `1 to 200`

D.3 Grade 10: *repeat*- and *while*-loops

Repeat and While Loops

Marking Sheet:

1 a and b
2 c
3 d
4 a
5 a
6 b
7 d
8 d
9 d
10 c
11 c
12 c

Repeat and While Loops ***Grade 10***	***Multiple Choice Class Test*** ***24 Marks***
Name:	

Please write your answer in the space provided on the right. Use the following responses if you are unable to select an answer:

X	Do not understand question
Y	Concept not covered in class
Z	Just don't know the answer

1. Consider the code to print all numbers which are smaller than 40 in the following sequence: 4,9,14,19,24,..

```
iNum := 4;
while (____________) do
begin
  memOut.lines.add(intToStr(iNum));
  iNum := iNum + 5;
end;
```

Which *while do* statement will result in the sequence printed correctly?

```
 a)  While (iNum < 40) do          b)  While (iNum <= 40) do
 c)  While (iNum = 40) do          d)  While (iNum = 39) do
```

1

2. Consider the following sequence: 50,38,26,14,.. . Complete the code below to print the first 15 numbers of this sequence.

```
iNum := 50;
iCounter := 1;
repeat
  memOut.lines.add(intToStr(iNum));
  iNum := iNum - 12;
  inc(iCounter);
until (___________);
```

Which *until* line statement will result in the sequence printed correctly:

```
 a)  until (iCounter = 15);        b)  until (iCounter >= 15);
 c)  until (iCounter > 15);        d)  until (iCounter <= 15);
```

2

3. Consider the following sequence: 1,3,9,27,81,.. . Complete the code below to print all numbers in the sequence that are smaller than 200.

```
iNum := 1;
while(iNum < 200) do
begin
  memOut.lines.add(intToStr(iNum));
  ???????????????
end;
```

The line ?????????????? above should be replaced by:

```
 a)  iNum := iNum + 3;             b)  iNum2 := iNum * 3;
 c)  iNum := 3 * 9;                d)  iNum := iNum * 3;
```

3

4. Consider the following sequence: 1,2,4,8,16,32,.. . Complete the code below to print the first 10 numbers of this sequence.

```
iNum := 1;
iCount := 0;
while (_____________) do
begin
  memOut.lines.add(intToStr(iNum));
  iNum := iNum * 2;
  inc(iCount);
end;
```

Which *while do* statement below will result in the sequence printed correctly:

```
a)  while (iCount < 10) do       b)  while (iNum < 10) do
c)  while (iCount > 10) do       d)  while (iNum > 10) do
```

4

5. If a person invests an initial (start) amount of R 100 in a bank account and the amount in the bank account increases by 12% per annum, how much money will be saved after 4 years? Consider the code below to calculate the amount:

```
rSavedAmt := 100;
iYears := 1;
repeat
  ????????????????
  inc(iYears);
until (iYears > 4);
memOut.lines.add(‘Amount saved after 4 yrs is R ‘ + FloatToStr(rSavedAmt));
```

The line ?????????????? above should be replaced by:

```
a)  rSavedAmt := rSavedAmt * 1.12;  b)  rSavedAmt := rSavedAmt * 100;
c)  rSavedAmt := 100 * 12;          d)  rSavedAmt := rSavedAmt * 0.12;
```

5

6. Consider the following code fragment:

```
n := 5;
m := 1;
while (m < n) do
begin
  memOut.lines.add(intToStr(m*m));
  inc(m);
end;
```

What number sequence is printed by the above code?

```
a)  1,4,9,16,32                  b)  1,4,9,16
c)  1,4                          d)  1,2,3,4,5
```

6

7. Consider the following sequence: 1,8,15,22,.. . Complete the code below to print all numbers smaller than 40.

```
iNum := 1;
repeat
  memOut.lines.add(intToStr(iNum));
  iNum := iNum + 7;
until (_____________);
```

Which *until* line statement will result in the sequence printed correctly:

```
a)  until (iNum < 40);           b)  until (iNum = 40);
c)  until (iNum <= 40);          d)  until (iNum >= 40);
```

7

8. Consider the following sequence: 1,2,4,7,11,16,.. . Complete the code below to print the first 20 numbers of the sequence.

```
iNum := 1;
iCounter := 0;
repeat
  ?????????????????
  inc(iCounter);
  memOut.lines.add(intToStr(iNum));
until (iCounter >= 20);
```

The line ?????????????? above should be replaced by:

```
a)  iNum := iNum + 1;            b)  iNum := iNum + 2;
c)  iNum := iNum * 2;            d)  iNum := iNum + iCounter;
```

8

9. Consider the following code fragment:

```
iNum := 1;
while (iNum < 30) do
begin
  memOut.lines.add(intToStr(iNum));
  iNum := iNum * (-2);
end;
```

What number sequence is printed by the above code?

```
a)  1,-2,-4,-6,-8,-10,-12,-14    b)  1,2,4,6,8,10,12,14
c)  1,-2,4,-8,16                 d)  1,-2,4,-8,16,-32
```

9

10. A certain plant grows by 2cm every month in the first two years. Thereafter, it grows by 0.5cm each month. Complete the code below to determine how tall (in cm) the plant will be after 5 years:

```
iYears := 1;
rGrowth := 0;
while (iYears <= 5) do
begin
  ???????????????
  inc(iYears);
end;
memOut.lines.add('The plant will be '+ floatToStr(rGrowth) + ' cm tall');
```

The line ?????????????? above should be replaced by:

```
a)  rGrowth := rGrowth+(12*2);   b)  rGrowth := rGrowth + (iYears * 2);

c)  if (iYears <= 2) then        d)  if (iYears <= 2) then
      rGrowth := rGrowth + 24          rGrowth := rGrowth + (12*2);
    else
      rGrowth := rGrowth + 6;
```

10

11. Consider the following code fragment:

```
iNum1 := 0;
iNum2 := 1;
iCount := 0;

n := strToInt(edtNumTimes.text);

while (n > iCount) do
begin
  memOut.lines.add(intToStr(iNum2));
  temp := iNum1 + iNum2;
  iNum1 := iNum2;
  iNum2 := temp;
  inc(iCount);
end;
```

If the user types in 8 (n=8), what sequence will be printed by the code above:

```
 a)  1,2,3,5,8,13,21,34           b)  1,2,3,4,5,6,7,8
 c)  1,1,2,3,5,8,13,21            d)  1,3,5,7,9,11,13,15
```

11

12. Consider the following code fragment that <u>should</u> print the sequence: 3,2,5,6,9,14,.. .

```
iMax := strToInt(edtCount.text);
iNum := 3;
iCount := 0;

repeat
  memOut.lines.add(intToStr(iNum));
  if (iNum mod 2 = 0) then
    iNum := iNum + 3
  else
    iNum := iNum * 2 - 4;
until (iCount = iMax);
```

iMax is a value typed in by the user to indicate how many numbers to print in the sequence. The code above causes an infinite loop. Which of the options below will fix the problem?

a) The *memOut* line should move to after the *else* line
b) `Until (iCount = iMax);` should be replaced by `until (iCount > iMax);`
c) The code is missing the line: `inc(iCount);`
d) The *repeat* loop should be replaced by a *while* loop

12

D.4 Grade 11: One-dimensional Arrays

Array (One Dimensional)

Marking Sheet:

1 b
2 b
3 c
4 c
5 a
6 b
7 c
8 d
9 c
10 b

Arrays (One-Dimensional) ***Multiple Choice Class Test***
Grade 11 ***20 Marks***

Name:

Please write your answer in the space provided on the right. Use the following responses if you are unable to select an answer:

X	Do not understand question
Y	Not done in class
Z	Just don't know the answer

The following represents an integer array:

3	**4**	**7**	**2**	**1**

The array is declared as follows:

```
    var myArray : array[0..4] of integer;
```

Use this array information to answer Questions 1-9 below.

1. What is the value of x (integer variable) after the following line of code is executed:

```
   x := myArray[1];
```

a) 3 b) 4 c) 2 d) 1

1

2. What is the value of y (integer variable) after the following lines are executed:

```
var a, y : integer;
begin
  a := 5;
  y := myArray[a-1];
end;
```

a) 2 b) 1 c) 0 d) 3

2

3. Which one of the following lines of code will change the value 7 in *myArray* to an 8?

```
 a)  myArray[7] := 8;           b)  myArray[3] := 8;

 c)  myArray[2] := 8;           d)  myArray[7] := myArray[8];
```

3

4. What values will the array *myArray* contain after the following code is executed?

```
var i : integer;
begin
  for i := 1 to 5 do
    myArray[i-1] := myArray[i-1]-1;
end;
```

a)	**-1**	**-1**	**-1**	**-1**	**-1**	b)	**3**	**4**	**7**	**2**	**1**
c)	**2**	**3**	**6**	**1**	**0**	d)	**0**	**1**	**2**	**3**	**4**

4

5. What values will the array *myArray* contain after the following code is executed?

```
var i : integer;
begin
  for i := 0 to 4 do
    myArray[i] := i;
end;
```

a)	**0**	**1**	**2**	**3**	**4**	b)	**3**	**4**	**7**	**2**	**1**
c)	**i**	**i**	**i**	**i**	**i**	d)	**1**	**2**	**3**	**4**	**5**

5

6. The following code segment should add all the numbers in the array:

```
var iCounter, iSum : integer;
begin
  iSum := 0;
  for iCounter := 0 to 4 do
    ?????????
  memOut.lines.add(‘The total is ‘ + intToStr(iSum));
end;
```

The ???????? in the above code should be replaced by:

```
 a)  iSum := iSum + iCounter;      b)  iSum := iSum + myArray[iCounter];

 c)  iSum := myArray[iCounter];    d)  iSum := icounter;
```

6

7. The following code segment should count the number of values in the array that are greater than 2:

```
var iStep, iCount : integer;
begin
  iCount := 0;
  for iStep := 0 to 4 do
    ?????????
  memOut.lines.add(‘Greater than 2: ‘ + intToStr(iCount));
end;
```

The ???????? in the above code should be replaced by:

```
 a)  iCount := iCount + 1;          b)  iCount := myArray[iStep] + 1;

 c)  if (myArray[iStep] > 2) then   d)  if (myArray[iStep] >= 2) then
           iCount := iCount + 1;              iCount := iCount + 1;
```

7

8. What values will the array *myArray* contain after the following code is executed?

```
var i : integer;
begin
  for i := 0 to 3 do
    if (myArray[i] > myArray[i+1]) then
      myArray[i+1] := myArray[i];
end;
```

a)	**3**	**4**	**7**	**2**	**1**	b)	**1**	**2**	**3**	**4**	**7**
c)	**3**	**4**	**2**	**1**	**7**	d)	**3**	**4**	**7**	**7**	**7**

8

9. The following line of code should display the first value in the array:

```
memOut.lines.add(intToStr(????????????));
```

The ???????? in the above code should be replaced by:

a) `myArray[1]` b) `1`

c) `myArray[0]` d) `0`

9

10. Which of the following lines of code declares an array that can contain 15 string variables?

```
a) var myArray : array[0..15] of string;
b) var myArray : array[0..14] of string;
c) var myArray : string;
d) var myArray : array[0..15] : string;
```

10

D.5 Grade 11: Procedures and Functions

Procedure/Function Questions

Marking Sheet:

1 c
2 a
3 b
4 d
5 b
6 d
7 c
8 d
9 b
10 c

Procedures and Functions ***Multiple Choice Class Test***

Grade 11 ***20 Marks***

Name:

Please write your answer in the space provided on the right. Use the following responses if you are unable to select an answer:

X	Do not understand question
Y	Concept not covered in class
Z	Just don’t know the answer

Use the following procedures and functions to answer the questions:

```
function myFunction1(iVal : integer) : integer;
begin
  result := 2*iVal;
end;

function myFunction2(iVal1, iVal2 : integer) : string;
begin
  result := ‘The answer is ‘+ intToStr(iVal1+iVal2);
end;

procedure myProcedure1(iNum : integer; sTxt : string);
var iCounter : integer;
begin
  for iCounter := 1 to iNum do
    memOut.lines.add(sTxt);
end;

procedure myProcedure2(sTxt : string; iNum : integer);
var iCount : integer;
    sTemp : string;
begin
  sTemp := ‘’;
  for iCount := 1 to length(sTxt) do
    sTemp := sTemp + sTxt[iCount] + intToStr(iNum);
  lblOutput.caption := sTemp;
end;

procedure myProcedure3(iNum1, iNum2 : integer);
begin
  lblOut.caption := ‘The answer is ‘ + inttoStr(a+b);
end;
```

1. Consider the following code fragment:

```
var iValue1, iValue2, iSum : integer;
begin
  iValue1 := strToInt(edtNum1.text);
  iValue2 := strToInt(edtNum2.text);
  iSum := getSum(iValue1, iValue2);
  lblOutput.caption := intToStr(iSum);
end;
```

Which of the following is the correct declaration of *getSum*?

```
a)  procedure getSum(iNum, iOtherNum : integer);
b)  procedure getSum(iNum, iOtherNum : integer) : integer;
c)  function getSum(iNum, iOtherNum : integer) : integer;
d)  function getSum(iNum, iOtherNum : integer);
```

1

2. Consider the following code fragment:

```
var iN1, iN2 : integer;
begin
  iN1 := 6;
  iN2 := 3;
  printProduct(iN1, iN2);
end;
```

Which of the following is the correct declaration of *printProduct*?

```
a)  procedure printProduct(iNum, iOtherNum : integer);
b)  procedure printProduct(iNum, iOtherNum : integer) : integer;
c)  function printProduct(iNum, iOtherNum : integer);
d)  function printProduct(iNum, iOtherNum : integer) : integer;
```

2

3. Which of the following code segments correctly uses *myProcedure1* (see page 1)*?*

```
a) iN := 1;
   sTemp := ‘abc’;
   myProcedure1(sTemp, iN);

b) iN := 1;
   sTemp := ‘abc’;
   myProcedure1(iN, sTemp);

c) iN := 1;
   sTemp := ‘abc’;
   sTextOut := myProcedure1(iN, sTemp);
   lblOut.caption := sTextOut;

d) iN := 1;
   sTemp := ‘abc’;
   myProcedure1(iNum, sTxt);
```

3

4. If a string variable `sSomeStr` and an integer variable `iANumber` are assigned values as follows:

```
sSomeStr := ‘xyz’;
iANumber := 9;
??????????
```

Which of the following lines will replace the ?????????? line and correctly use *myProcedure2* (see page 1)?

```
a)  myProcedure2(iANumber, sSomeStr);
b)  myProcedure2(sTxt, iNum);
c)  anotherStr := myProcedure2(sTxt, iNum);
d)  myProcedure2(sSomeStr, iANumber);
```

4

5. Consider the following code fragment:

```
var iNum, iNum2 : integer;
begin
  iNum := strToInt(edtIn.text);
  ????????????
end;
```

Which of the following lines will replace the ?????????? line and correctly use *myFunction1* (see page 1)?

```
a)  myFunction1(iNum);
b)  iNum2 := myFunction1(iNum);
c)  myFunction1(iVal);
d)  iNum2 := myFunction1(iVal);
```

5

6. The following code segment should print the sum of two numbers:

```
iNum1 := 5;
iNum2 := 6;
??????????
memOut.lines.add(‘The sum is ‘ + intToStr(iNum3));
```

The ???????? line in the above code should be replaced by (refer to page 1):

```
a)  iNum3 := myFunction2(iNum2, iNum1);
b)  iNum3 := myFunction2(iVal1,iVal2);
c)  myProcedure3(iNum1,iNum2);
```
d) None of the above

6

7. If you have to write a program that takes a string value and an integer value, eg. ‘computer’ and 5 respectively, and then inserts the integer value between each character of the string , e.g. `c5o5m5p5u5t5e5r5`

Which one of the following lines will do this (refer to page 1)?

```
a)  myProcedure1(5,’computer’);
b)  myProcedure1(‘computer’,5);
c)  myProcedure2(‘computer’,5);
d)  myProcedure2(5,’computer’);
```

7

8. Consider the following procedure/function:

```
procedure/function header
var iFact, iCounter : integer;
begin
  ifact := 1;
  for iCounter := 1 to iNum do
    iFact := ifact * iCounter;
  result := iFact;
end;
```

If *iNum* is an input parameter, which of the following will replace **procedure/function header** for this code:

a) procedure Factorial (iNum : integer);
b) function Factorial (iFact : integer) : integer;
c) function Factorial (iFact : integer);
d) function Factorial (iNum : integer) : integer;

8

9. Consider the following procedure/function:

```
procedure/function header
var iCounter1, iCounter2 : integer;
    sTemp : string;
begin
  for iCounter1 := 1 to iSize do
  begin
    stemp := ‘’;
    for iCounter2 := 1 to iCounter1 do
      sTemp := stemp + sPattern;
    memOut.lines.add(sTemp);
  end;
end;
```

If *sPattern* and *iSize* are input parameters, which of the following will replace **procedure/function header** for this code:

a) procedure printPatterns(sPattern, iSize : integer);
b) procedure printPatterns(sPattern : string; iSize : integer);
c) procedure printPatterns(sPattern, iSize : string);
d) procedure printPatterns(iSize : integer);

9

10. You are required to write a procedure/function that takes two integer input parameters and returns the biggest (largest) of these two values. Which one of the following would be the most appropriate procedure/function declaration?

a) function theBiggest(iNum1 : integer);
b) function theBiggest(iNum1, iNum2 : integer);
c) function theBiggest(iNum1, iNum2 : integer) : integer;
d) procedure theBiggest(iNum1, iNum2 : integer);

10

Appendix E

PAT Evaluation Questionnaire

E.1 RoboMind Questionnaire

Name:		Grade:	

Please complete the following questions. We would like to determine the usefulness of RoboMind. We would appreciate your co-operation. Remember that these are your honest views; there are no right or wrong answers. Mark the appropriate number or response with an **X.**

1. Have you installed RoboMind?		**Yes**	**No**
2. How many times do you use RoboMind per week?	**0 – 1 times**	**2-3 times**	**4 or more times**

3. RoboMind has helped me with my understanding of:

	Strongly Disagree				*Strongly Agree*	
a. Variables	*1*	***2***	***3***	***4***	***5***	*Not done yet*
b. If Statements	*1*	***2***	***3***	***4***	*5*	*Not done yet*
c. For Loops	*1*	***2***	***3***	***4***	*5*	*Not done yet*
d. Repeat Loops	*1*	***2***	***3***	***4***	*5*	*Not done yet*
e. While Loops	*1*	***2***	***3***	***4***	*5*	*Not done yet*
f. Procedures (Gr11)	*1*	***2***	***3***	***4***	*5*	*Not done yet*

4. I think RoboMind is easy to use.				
Strongly Disagree 1	*2*	*3*	*4*	*5 Strongly Agree*
5. I think most people would learn to use RoboMind very quickly.				
Strongly Disagree 1	*2*	*3*	*4*	*5 Strongly Agree*
6. I think I need assistance to be able to use RoboMind.				
Strongly Disagree 1	*2*	*3*	*4*	*5 Strongly Agree*
7. I feel very confident using RoboMind.				
Strongly Disagree 1	*2*	*3*	*4*	*5 Strongly Agree*

8. What do you like about RoboMind?
9. What do you **not** like about RoboMind?
10. What changes or improvements to RoboMind would you recommend?
11. If you are not using RoboMind, why not?

Would you be interested in additional help installing or using RoboMind? Yes / No

E.2 Scratch Questionnaire

Name:		Grade:	
Please complete the following questions. We would like to determine the usefulness of Scratch. We would appreciate your co-operation. Remember that these are your honest views; there are no right or wrong answers. Mark the appropriate number or response with an **X.**			

1. Have you installed Scratch?		**Yes**	**No**
2. How many times do you use Scratch per week?	**0 – 1 times**	**2-3 times**	**4 or more times**

3. Scratch has helped me with my understanding of:							
		Strongly Disagree				*Strongly Agree*	
a.	Variables	*1*	***2***	***3***	***4***	***5***	*Not done yet*
b.	Input	*1*	***2***	***3***	***4***	*5*	*Not done yet*
c.	Output	*1*	***2***	***3***	***4***	*5*	*Not done yet*
d.	If Statements	*1*	***2***	***3***	***4***	*5*	*Not done yet*
e.	For Loops	*1*	***2***	***3***	***4***	*5*	*Not done yet*
f.	Repeat Loops	*1*	***2***	***3***	***4***	*5*	*Not done yet*
g.	While Loops	*1*	***2***	***3***	***4***	*5*	*Not done yet*
h.	String Handling	*1*	***2***	***3***	***4***	*5*	*Not done yet*
i.	Arrays (Gr11)	*1*	***2***	***3***	***4***	*5*	*Not done yet*
j.	Procedures (Gr11)	*1*	***2***	***3***	***4***	*5*	*Not done yet*

4. I think Scratch is easy to use.				
Strongly Disagree 1	*2*	*3*	*4*	*5 Strongly Agree*
5. I think most people would learn to use Scratch very quickly.				
Strongly Disagree 1	*2*	*3*	*4*	*5 Strongly Agree*
6. I think I need assistance to be able to use Scratch.				
Strongly Disagree 1	*2*	*3*	*4*	*5 Strongly Agree*
7. I feel very confident using Scratch.				
Strongly Disagree 1	*2*	*3*	*4*	*5 Strongly Agree*

8. What do you like about Scratch?

9. What do you **not** like about Scratch?

10. What changes or improvements to Scratch would you recommend?

11. If you are not using Scratch, why not?

Would you be interested in additional help installing or using Scratch? Yes / No

E.3 B# Questionnaire

Name:		Grade:	
Please complete the following questions. We would like to determine the usefulness of B#. We would appreciate your co-operation. Remember that these are your honest views; there are no right or wrong answers. Mark the appropriate number or response with an **X.**			

1. Have you installed B#?		**Yes**	**No**
2. How many times do you use B# per week?	**0 – 1 times**	**2-3 times**	**4 or more times**

3. B# has helped me with my understanding of:

	Strongly Disagree			*Strongly Agree*		
a. Variables	*1*	***2***	***3***	***4***	***5***	*Not done yet*
b. Input	*1*	***2***	***3***	***4***	*5*	*Not done yet*
c. Output	*1*	***2***	***3***	***4***	*5*	*Not done yet*
d. If Statements	*1*	***2***	***3***	***4***	*5*	*Not done yet*
e. Case Statements	*1*	***2***	***3***	***4***	*5*	*Not done yet*
f. If Statements	*1*	***2***	***3***	***4***	*5*	*Not done yet*
g. Repeat Loops	*1*	***2***	***3***	***4***	*5*	*Not done yet*
h. While Loops	*1*	***2***	***3***	***4***	*5*	*Not done yet*

4. I think B# is easy to use.

Strongly Disagree 1	*2*	*3*	*4*	*5 Strongly Agree*

5. I think most people would learn to use B# very quickly.

Strongly Disagree 1	*2*	*3*	*4*	*5 Strongly Agree*

6. I think I need assistance to be able to use B#.

Strongly Disagree 1	*2*	*3*	*4*	*5 Strongly Agree*

7. I feel very confident using B#.

Strongly Disagree 1	*2*	*3*	*4*	*5 Strongly Agree*

8. What do you like about B#?

9. What do you **not** like about B#?

10. What changes or improvements to B# would you recommend?

11. If you are not using B#, why not?

Would you be interested in additional help installing or using B#? Yes / No

Appendix F

Determination of PAT Rankings Based on Selection Criteria

F.1 Introduction

The selection criteria (Section 4.4) are used to evaluate the PATs and select the PATs most suitable for use by IT learners. The PATs are evaluated on three categories of selection criteria: programming concepts, programming knowledge and programming skills. Criteria in each of these categories are listed in order of priority with the most important criteria listed first. A method of determining the overall ranking of PATs was required based on the evaluation of the PATs in the three categories.

PATs should be ranked based on the number of selection criteria that are addressed by the PAT taking into consideration the priority of the selection criteria that are addressed. In each of the three categories, the selection criteria are weighted according to their priority ranking. The ranking of the PATs based on selection criteria for programming knowledge (Section F.2), programming skills (Section F.3) and programming concepts (Section E.4) are provided. The overall ranking of PATs is obtained by combining scores for each of the three categories (Section F.5).

F.2 Programming Knowlege Rankings

The first knowledge criteria item has a weighting of five (Table F.1). The priority weighting values decrease by one for the criteria that follow. Only PATs that support the Delphi programming language meet the first criteria to assist with programming language syntax.

The PAT scores are calculated by summing the weighting value multiplied by the value for the PAT. A value of one (1) indicates that the PAT meets the criteria. A blank cell is equivalent to zero and indicates that the PAT does not meet the selection criteria.

Table F.1: Calculation of PAT ranking score based on programming knowledge criteria

Criteria	**Weighting**	**RoboMind**	**BlueJ**	**Greenfoot**	**Scratch**	**B#**	**Jeliot**	**Ville**	**PlanAni**	**Alice**	**jGRASP**
Assists with the learning of the Delphi programming language syntax	5	1				1			1		
Assists with developing knowledge of programming principles & concepts	4	1	1	1	1	1	1	1	1	1	1
Constructivist to promote self-study	3	1	1	1	1	1	1	1	1	1	1
Assists with the application of programming knowledge	2						1	1	1		
Assists with the understanding of code execution	1	1	1	1	1	1	1	1	1	1	1
Total score		13	8	8	8	13	10	10	15	8	8

F.3 Programming Skill Rankings

The programming skill criteria are also weighted from five for the first priority item to one for the last (Table F.2). The calculation of the total PAT scores are the same as for the programming knowledge rankings. Half values ($\frac{1}{2}$) are assigned to PATs that do not cater for the the selection criteria due to the techniques used by the PAT (Section 5.4).

Table F.2: Calculation of PAT ranking score based on programming skill criteria

Criteria	**Weighting**	**RoboMind**	**BlueJ**	**Greenfoot**	**Scratch**	**B#**	**Jeliot**	**Ville**	**PlanAni**	**Alice**	**jGRASP**
Promotes problem solving & planning	5				1	1				1	
Provides simple error messages to assist with debugging	4	1			$\frac{1}{2}$	1			$\frac{1}{2}$	$\frac{1}{2}$	
Develops code comprehension	3				1	1	1	1	1	1	
Feedback to guide solution creation	2				1	1				1	
Feedback regarding errors	1	1	1	1	$\frac{1}{2}$	1	1	1	$\frac{1}{2}$	$\frac{1}{2}$	1
Total score		5	1	1	12.5	15	4	4	3	12.5	1

F.4 Programming Concept Rankings

The programming concept criteria are weighted from 18 to 1 based on the ranking of the criteria (Table F.3). The calculation of the total score is the same as for the programming knowledge and programming skills scores.

Table F.3: Calculation of PAT ranking score based on programming concept criteria

Criteria	**Weighting**	**RoboMind**	**BlueJ**	**Greenfoot**	**Scratch**	**B#**	**Jeliot**	**Ville**	**PlanAni**	**Alice**	**jGRASP**
Two-dimensional arrays	18		1	1			1				1
String handling	17		1	1	1		1	1			1
One-dimensional arrays	16		1	1	1		1	1	1		1
Procedures	15	1	1	1	1		1	1		1	1
Functions	14		1	1			1				1
repeat-loops	13	1	1	1	1	1	1	1	1	1	1
while-loops	12	1	1	1	1	1	1	1	1	1	1
Objects & classes	11		1	1	1		1			1	1
for-loops	10	1	1	1	1	1	1	1	1	1	1
if-statements	9	1	1	1	1	1	1	1	1	1	1
Correct use of parameters	8	1	1	1			1	1			1
SQL statements	7		1	1							1
Accessing database	6		1	1							1
case-statements	5		1	1		1	1				1
File handling	4		1	1							1
Variables	3	1	1	1	1	1	1	1	1	1	1
Input	2	1	1	1	1	1	1		1	1	1
Output	1		1	1	1	1	1	1	1	1	1
Total score		72	171	171	109	55	154	104	66	76	171

F.5 Overall Rankings

The overall rankings are obtained by calculating a total score based on the scores obtained in the three different selection criteria categories (Table F.4). The formula to calculate the total score for a PAT is:

$$PAT\ Overall\ Score = Knowledge\ score + Skill\ score + 0.1{*}Concept\ score$$

Only one tenth of the concept score is considered to minimise the effect of the number of criteria items (18) in the programming concept category compared to the five items in each of the programming knowledge and programming skill categories. Only the PATs that support Delphi have been considered for evaluation by participating IT learners in the study.

Table F.4: Overall PAT rankings

	PAT	Knowledge	Skills	Concepts*0.1	Total	Delphi
1.	Jeliot	10	4	15.4	29.4	
1.	Greenfoot	8	1	17.1	26.1	
	BlueJ	8	1	17.1	26.1	
	jGRASP	8	1	17.1	26.1	
4.	B#	13	15	5.5	33.5	✓
5.	Scratch	8	12.5	10.9	31.4	✓
6.	Alice	8	12.5	7.6	28.1	✓
7.	RoboMind (Adapted)	14	5	7.2	26.2	✓
8.	PlanAni	15	3	6.6	24.6	✓
9.	Ville	10	4	10.4	24.4	

Appendix G

PAT Supporting Documentation

G.1 RoboMind

http://www.robomind.net/en/index.html

RoboMind is a programming tool that allows you to program a robot to move through a map world.

Install:

The files are included on the CD. Simply copy the *RoboMind* folder to your hard drive. First install the *Java Runtime Environment* (**jre-6u24-windows-i586** is the install file in the *RoboMind folder* – just double-click). This is the only file you need to install. To run the program, open the *prjRobo folder* (inside *RoboMind* folder) and double-click the **prjRobo.jar** file.

Support:

Several example programs are included to help you learn how to use *RoboMind*. Click on open file and select a program.

You can also find information on the *RoboMind* website.

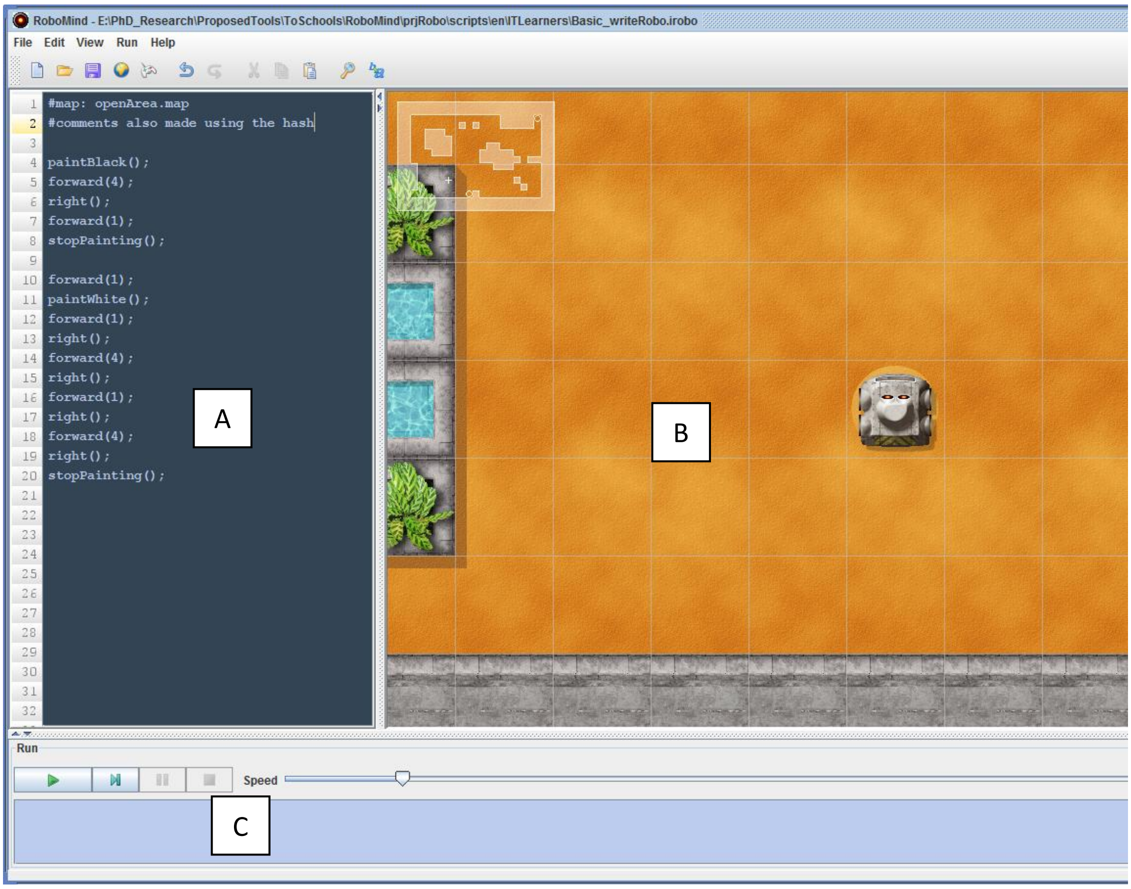
Concepts:

You can implement the following IT programming concepts in *RoboMind*:

- Variables (integer)
- *If*-statements (nested)
- Looping (RoboMind does not implement a normal *for*-loop like Delphi. You must use *for1to( )* where for example, for1to(8) repeats something 8 times. *Repeat..until* and *while..do* loops are also included.
- Procedures

RoboMind has additional commands to control the robot such as movement, turning, grabbing beacons and painting lines. The robot can also see (to check if the path is clear) and flip a coin (to make a decision).

Several different maps are included in *RoboMind*. Instructions are included to explain how you can create your own maps.


RoboMind - E:\PhD_Research\ProposedTools\ToSchools\RoboMind\prjRobo\scripts\en\ITLearners\Basic_writeRobo.irobo
File
Edit
View
Run
Help
A
B
Run
Speed
C

```
#map: openArea.map
#comments also made using the hash

paintBlack();
forward(4);
right();
forward(1);
stopPainting();

forward(1);
paintWhite();
forward(1);
right();
forward(4);
right();
forward(1);
right();
forward(4);
right();
stopPainting();
```

The screenshot above is the interface you will work with when using *RoboMind*.

- The code to control the robot is typed in the editor (Window A).
- To run the code, click on the green arrow ▶ below the editor. If there are no errors in the code the program will run and the robot will move through the world.
- Error messages will appear in the blue window at the bottom (C).
- There are several example exercises which can be accessed by opening a file 📂. The ITLearner folder has additional exercises.
- *RoboMind* has several built-in commands for movement and to check whether or not there are obstacles around it. Use the Edit -> Insert menu to see which commands are available or look at the exercises. The website can also be used for more information.
- **Please note:** There are differences between the concepts provided on the website and in the exercises. This is because the version you have received has been adapted to make the code look similar to Delphi. Use the exercises to find out more about using variables, the different loops and procedures.
- There are several maps available (See File->Open Map). You can also create your own maps. If no map is specified in the code, it will use the one currently viewed in window B. To specify a map, at the top of the code type the name of the map file you want to use, for example, **#default.map** where *default.map* is the name of the map file. The file *RoboMapFormat.pdf* has been included and explains how you can create your own map.

G.2 Scratch

SCRATCH

http://scratch.mit.edu/

Scratch is a programming tool that allows you to easily create your own interactive stories, animations, games, music and art. You can also post your projects on the web or download projects that other people have created.

Install:

The install file is included on the CD. Simply double-click the **ScratchInstaller1.4.exe** file to begin the installation.

Support:

In the Scratch Help menu, use the Scratch Help Page and Help Screens to learn more about using Scratch.

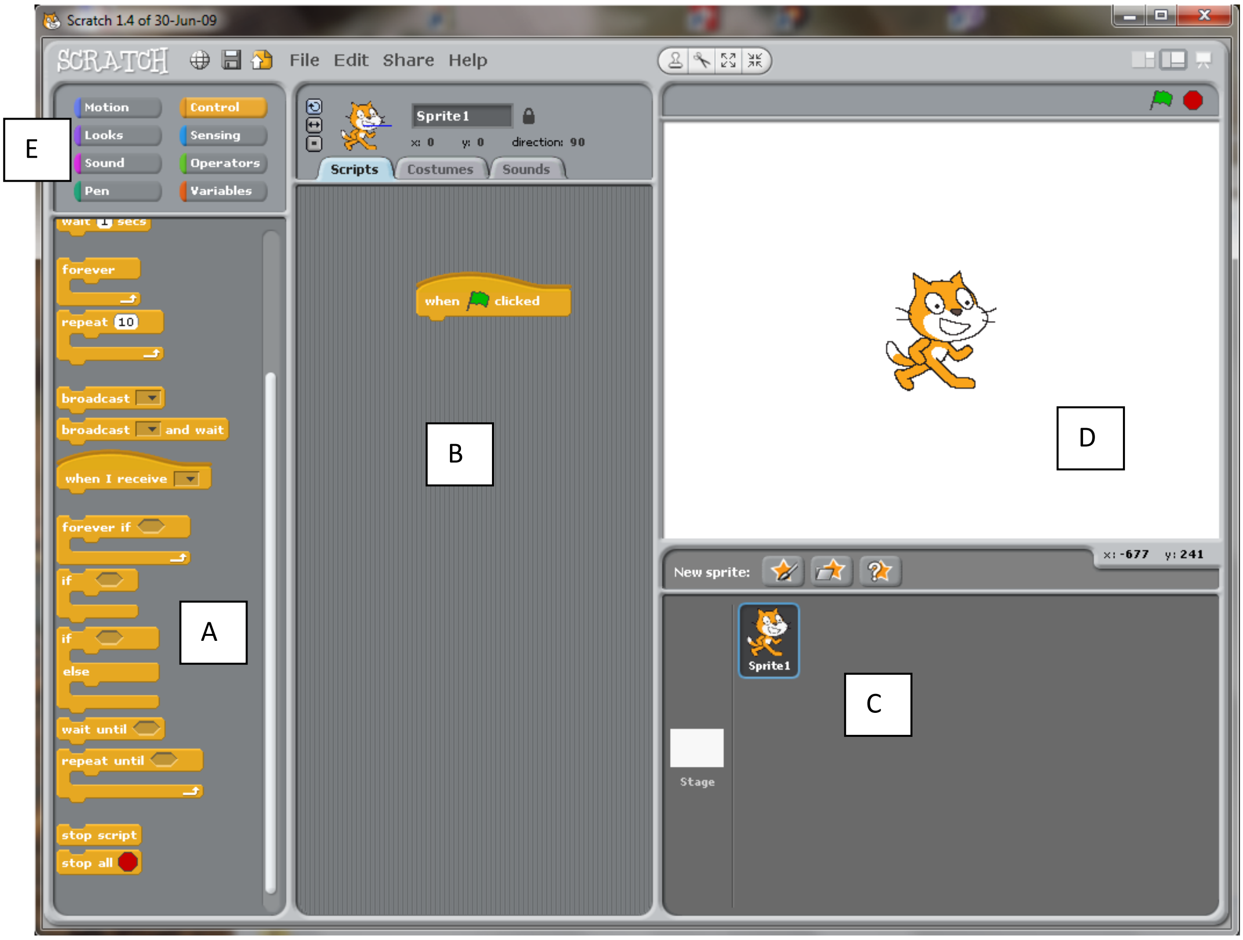
Scratch installs several Example projects for you to use. An *ITLearner* folder is included on the CD with projects that show you how to use Scratch for programming concepts you will learn about in your IT classes.

You can also find tutorials and example projects on the Scratch website.

Concepts:

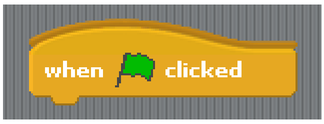
You can implement the following IT programming concepts in Scratch:

- Variables (integer)
- Input from the user including text or number input to answer a question or using the keyboard arrow keys, etc. to control characters (called sprites) on the screen
- Output to the user
- *If*-statements
- Looping (Scratch does not implement a normal for loop like Delphi. Instead you would use a repeat 10 for example to repeat something 10 times.
- Simple string handling such as joining words, getting the length of a word, finding a specific character in a word.
- Arrays (lists in Scratch)



Using Scratch:

The screenshot above is what you will work with when using Scratch.

- To create a “program”, drag items from window A to window B. The statements can lock together (you will know they are locked when they all move around together).
- If you want to remove statements, either right-click on the statement and click Delete or simply drag from window B to window A.
- There are many different statements available to you. You will find them by selecting the different categories listed at the top left (E).
- Use  to perform actions when the user clicks the green flag – so this will be the start event.
- To change the sprite, use the New sprite options in window C.
- You can also move or rotate the sprite in the main window D
- At the top of window B are 3 tabs. Scripts are the programs that will control the sprite. Costume allows you to change the appearances of a sprite, for example, to make it look like it is walking. You can also associate sounds with the sprite.

- Remember, the different sprites are listed in window C. You can have more than one. Each sprite in window C can have its own script or program, that is, what to do when the flag is clicked or the user uses the arrow keys.

G.3 B#

B#

B# is a programming tool developed at by the NMMU Department of Computing Sciences. B# allows you to create a program simply by creating a flowchart. The flowchart is created by dragging appropriate items into the flowchart area in a specific order and indicating how variables and values are calculated and what is displayed.

Install:

The files are included on the CD. Simply copy the *B#* folder to your hard drive. In the folder, double-click the B#.msi file to install the program.

Support:

Several example programs are included on the CD to help you learn how to use *B#*. Open any of these file in B# to see the flowchart, the corresponding code and to see what happens when the program runs.

Concepts:

You can implement the following IT programming concepts in *B#*:

- Variables (integer)
- *If*-statements (nested)
- *case* statements
- Looping (*for*, *repeat..until* and *while..do* loops)
- User input and output

B# creates Pascal code which is the same code used in Delphi. However, *B#* uses a console application (black command line interface) and not a graphical user interface like Delphi. The only difference in the code generated by B# is when input is received from the user or output is displayed to the user.

When using B#, write/writeln are used to display text to the user, while readln is used to get input from the user. the following lines will get a number from the user and assign it to the integer variable *num*:

```
write(‘Enter a number’);
readln(num);
```

This would be similar to having a label with caption ‘Enter a number’ and an edit box (edtNum), then when the user clicks a button (for example) the following code will be used to assign the number entered in the edit box to num:

```
num := strToInt(edtNum.text);
```

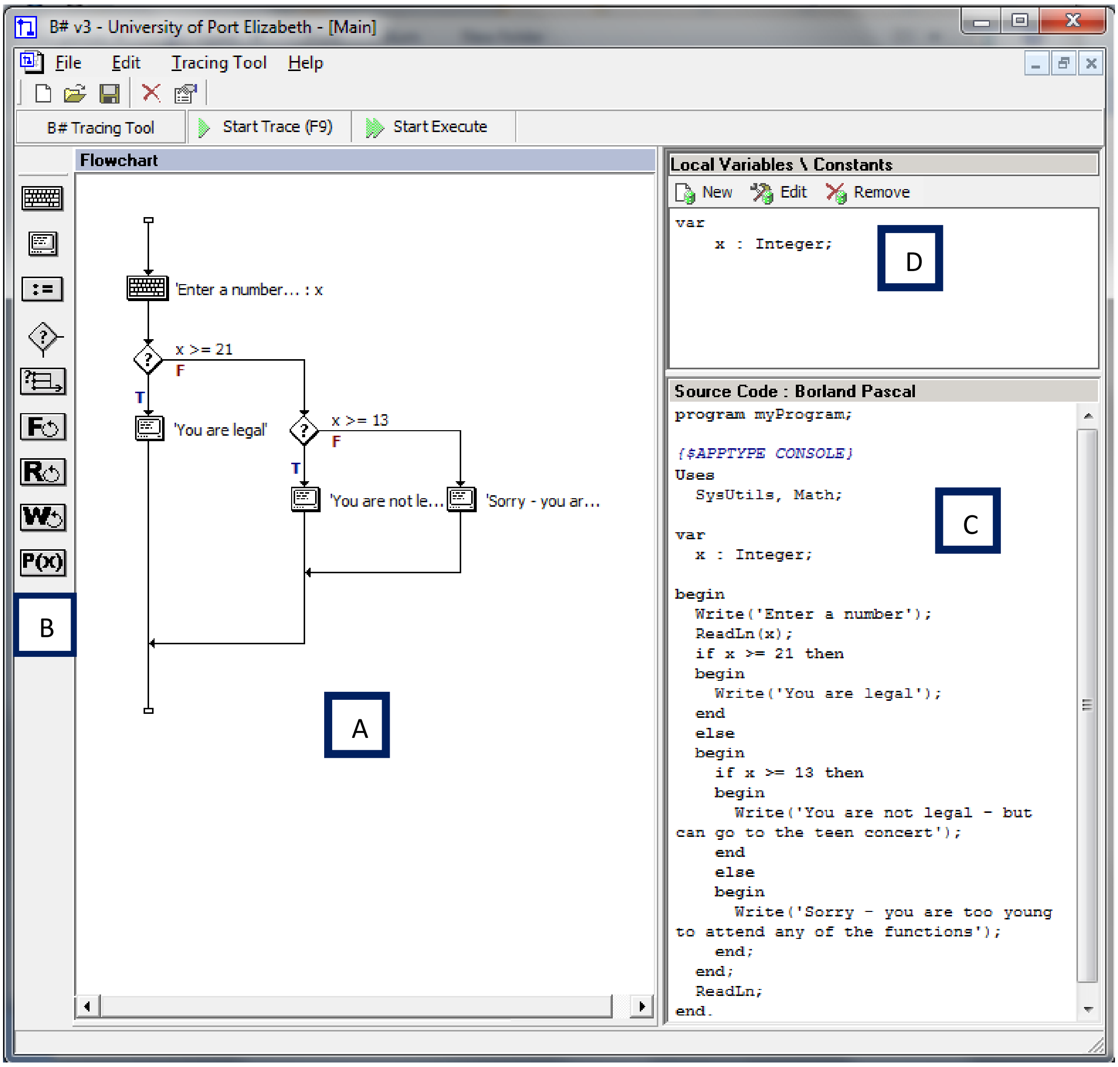
Note that when using a console interface the user needs to be asked a question (write/writeln), the program must wait for the answer and assign it to an appropriate type variable (readln).

Similarly, to display output to the user, write or writeln is used:

```
writeln('The user entered', num);     //B#
lblOutput.caption := 'The user entered' + intToStr(num);   //Delphi
```

Notice that in the writeln automatically combines a string and integer variable, while in Delphi the integer variable type must be converted to a string.

Please remember these conversions with regards to input and output if first doing exercises using B# and copying the generated code to Delphi.



```
var
    x : Integer;
```

```
program myProgram;

{$APPTYPE CONSOLE}
Uses
  SysUtils, Math;

var
  x : Integer;

begin
  Write('Enter a number');
  ReadLn(x);
  if x >= 21 then
  begin
    Write('You are legal');
  end
  else
  begin
    if x >= 13 then
    begin
      Write('You are not legal - but
can go to the teen concert');
    end
    else
    begin
      Write('Sorry - you are too young
to attend any of the functions');
    end;
  end;
  ReadLn;
end.
```

The screenshot above is the interface you will work with when using *B#*

- To create a program, create a flowchart by dragging icons from the left hand side (B) to the flowchart work area (A) in the correct order.
- The code for the program is generated on the right hand side (C)

- Before assigning values to variables or using variables, the variable must be created. Variables that can be used are listed in D. New variables can be created here and existing variables can be edited or removed.
- Remember to use the single quotes when typing messages to display to users.
- The program can be executed to check if it is working properly.
- The trace tool goes through the program step by step so that each step can be checked to see if it is correct.

Appendix H

Papers Originating from this Research Study

H.1 E-Skills 2010 Conference

The Impact of IT at Schools on E-Skills Development

Melisa KOORSSE[1], André P. CALITZ[2], Charmain B. CILLIERS[3]
Nelson Mandel Metropolitan University (NMMU), P.O. Box 77000, Port Elizabeth, 6031, South Africa
[1]*Tel: +27 41 372 2193, Fax: +27 41 504 2831, Email: Melisa.Koorsse@nmmu.ac.za*
[2] *Tel: +27 41 504 2639, Fax: +27 41 504 2831, Email: Andre.Calitz@nmmu.ac.za*
[3] *Tel: +27 41 504 2235, Fax: +27 41 504 2831, Email: Charmain.Cilliers@nmmu.ac.za*

Abstract: Attention and resources are focused on the development and upliftment of e-skills in South Africa. Business development, government and education have been identified as categories where the shortage of ICT skills requires urgent attention. The objective of this paper is to report on the impact of the current Information Technology (IT) subject in South African secondary schools on e-skills developments. This paper presents feedback indicating that the IT subject is having a direct impact on the number of future ICT professionals. The number of Grade 9 learners selecting IT as a subject is decreasing. The number of learners that start IT in Grade 10 but drop the subject before Grade 12 is increasing. Very few Grade 12 IT learners decide to study further in a Computer Science or Information Systems degree programme after matriculating or pursue a career in a computer related field. This paper considers research related to the current IT curriculum and ICT skills in South Africa, as well as the opinions of IT teachers and subject advisors. Issues that need to be addressed are the relevance and scope of topics in the current learning outcome guidelines. Plans for e-skills development in South Africa may not have the desired impact if the Information Technology subject and its role in e-skills development are not addressed by all stakeholders.

Keywords: E-skills, ICT skills shortage, Information Technology, Education in Secondary School

1. Introduction

The importance of Information and Communication Technology (ICT) skills and professionals for business and the economy is generally accepted [4, 8, 14]. The South African government identified the importance of e-skills development in 2007 and a council was tasked to address this issue [13]. However, unless the content framework of the current Information Technology (IT) subject presented in South African secondary schools is addressed, e-skills development initiatives may be in vain. Pilot research in schools supported by the views of educators that problems exist with the subject guidelines and implementation of the IT subject indicate that the implementation of IT in South African secondary schools could have serious negative impacts on the future of e-skills development in South Africa.

The term *e-skills* refers to a range of knowledge, skills and competences which incorporates three main categories [15]: ICT practitioner skills, ICT user skills and e-business skills. Together these three categories allow the term *e-skills* to encompass the capabilities required to develop, design, manage, maintain, support, install and service ICT systems, effectively apply ICT systems and devices as tools to support the work, such as business functions within industry, of individual users and to identify and exploit the opportunities provided by ICT to improve the efficiency and performance of organisations.

In 2007, then President of South Africa, Thabo Mbeki, stated that it would be futile to improve South Africa's ICT infrastructure if there was a shortage of skilled people to use the technology [14]. Statistics presented by Oracle supported this statement by indicating that "the skills shortage in the manufacturing sector alone will lag by 14 000 ICT specialists in 2010" [14]. The reason for this shortage needs to be addressed.

The European Forum [15] identified three deficiencies in terms of e-skills: a lack of skilled ICT professionals, a gap between the current and required levels of competence of ICT staff within organisations and a mismatch between the competence of graduates and the competence level required by business due to course misalignment. A report by the ACM describes the global trend in the decline of interest in computer related professions [1]. Job opportunities still exist yet there is a shortage of qualified computer and ICT specialists. Surveys and interviews with students are being administered to determine the reason and to find ways of encouraging students to consider computer related fields as a profession. Schools and universities worldwide are trying to determine the reason for the lack of interest.

This paper addresses the following research question: **What impact is IT at secondary school having on the development of e-skills in South Africa?** Evidence is presented of the negative impact that IT in schools is having on the future career choice of learners. Related research on the reasons for the decrease in the number of people choosing Computer Science or Information Systems as a profession will be presented. The main purpose of this paper is to initiate discussions and debates amongst key stakeholders regarding the problems and way forward for IT as a subject in South African schools.

This paper presents a background of e-skills development in South Africa (Section 2), the methodology used to gather and present information (Section 3), a description of the IT subject at school (Section 4), a summary of educators' opinions on the current issues regarding IT in schools (Section 4), suggested action to ensure that the problems can be addressed (Section 6) and concluding remarks (Section 7).

2. Background

Computers and the Internet play a very important role in social, economic and cultural aspects of our everyday lives. ICT infrastructure provides industries with a competitive edge if properly leveraged. The impact of ICT on health care, education, government provisioning and service delivery, to name but a few, is the reason why countries around the world are calling for initiatives and focus on the development of e-skills.

The past 10 years has recorded an increase followed by a decrease in interest in computing as a profession [2, 11, 12]. At the turn of the century interest in the computing industry was increased [2]. However, after 2001 interest in computing professions started to decrease due to several negative factors [2]. Countries across the world are facing another challenge with regards to the *Baby Boom* generation which is set to retire within the next 10 to 15 years. Results have indicated that the number of Computer Science (CS) and Information System (IS) graduates from tertiary education institutions is not sufficient to replace them [2]. This is in contradiction to the increase in demand for skilled ICT professionals. The computing profession currently has a very low unemployment figure that is indicative of a labour market under pressure.

Research by Babin, *et al.* [2] and Biggers, *et al.* [3] have focused on identifying the reasons for the lack of interest in computing as a career choice particularly at tertiary level. Survey results have indicated that students are uninformed or misinformed about the actual job descriptions of a computing professional. In many cases the curriculum or subject content is not presented correctly and the result is that students perceive a career in computing as being asocial, only focused on programming with little connection to the outside world [3]. Students have also been misinformed about certain industry business practices. In the USA, for instance, a misconception about outsourcing of positions has lead many students to question job security in the IT field [2].

South Africa is no exception to the global trend of the skilled IT professional shortage. The South African Government has formulated a medium term strategic plan [16] to guide the process to achieve strategic objectives that will result in a democratic, non-racial, non-sexist and prosperous society. The development of ICT skills and infrastructure are key goals in several strategic priorities including the speeding up of economic growth and the creation of decent work and sustainable livelihoods, the building of economic and social infrastructure and the improvement of the delivery and quality of public services. These strategic priorities may not be achieved successfully if there is a shortage of properly trained ICT skilled professionals.

A problem which will influence the achievement of the strategic priorities outlined in [16] is that fewer learners in South African secondary schools are selecting the Information Technology subject at school. Of the learners that take the subject, very few continue with Computer Science (CS) or Information Systems (IS) courses at tertiary level. South Africa has the opportunity to foster an interest in ICT at school level where learners can be exposed to the socio-economic importance of ICT and the different career opportunities it will provide. However, research results seem to indicate that IT at school is having the opposite effect – after being exposed to the IT curriculum students then decide not continue a career in computing [11]. Careful consideration needs to be given to what the IT curriculum is exposing or not exposing learners to that they lose interest in the ICT profession at the end of their secondary schooling.

The research presented in this paper is important as it plays a role in making government, business stakeholders, tertiary institutions and curriculum decision makers aware of the impact that the IT subject in South African schools is having on the future of the ICT profession and the national growth and development plans of South Africa. It is important for this research to raise awareness and encourage further investigation into secondary school learners' lack of interest in IT, shortcomings of the current IT subject framework and learning programme guidelines and positive changes that should be implemented to renew interest in ICT careers.

3. Methodology

The research approach is ethnographic using interpretivist methods [5]. There has been growing concern amongst all involved with IT at secondary schools that the subject curriculum is having a negative impact on the future of skills development. These concerns have emerged at various forums including computer studies mailing groups, meetings with IT educators to address concerns, Computer Science education conferences and interviews with IT teachers.

An ethnographic approach has been adopted as it is a method where the world view of participants is investigated and represented. The ethnographic approach to this study allows the views of people involved with IT to be interpreted. IT educators have 'insider knowledge' of the shortcomings of the current curriculum and how it is affecting the future of e-skills development in South Africa.

The primary source of data for this study was unstructured interviews with IT teachers. The interviews took place during December 2009 and February 2010. Qualitative content analysis was used to analyse the interview data.

4. IT in South Africa

Information Technology (IT) is one of the 29 subjects included in the National Curriculum Statement of the South African Department of Education [6] offered at secondary school level for Grades 10 to 12. The IT subject is defined in the National Curriculum Statement (NCS) for South African schools [6] as follows:

> "*Information Technology focuses on activities that deal with the solution of problems through logical thinking, information management and communication. It also focuses on the development of computer applications using current development tools. The subject develops awareness and an understanding of the social, economic and other implications of using computers.*"

The purpose of IT, as described in the Learning Programme Guidelines [7] is to afford learners the opportunity to learn about and work with ICTs. IT is designed to develop higher-order thinking skills, technology skills, information skills, problem-solving skills, creative skills, collaborative skills and lifelong learning skills. IT learners will gain a deeper understanding of the concepts and principles of ICT hardware and software, be taught to use digital technology to solve problems and learn about programming as a process of designing, developing and implementing software solutions. IT at school has four learning outcomes as specified in the IT National Curriculum Statement [6] to meet these ideals:

- Learning Outcome (LO) 1 : Hardware & System Software
- Learning Outcome (LO) 2 : e-Communication
- Learning Outcome (LO) 3 : Social & Ethical Issues
- Learning Outcome (LO) 4 : Programming and Software Development

These learning outcomes originate from the broader knowledge domain of ICTs.

The general vision for IT and the skills it will teach learners incorporates valuable skills and knowledge that could be considered core and the basic principles to a computer related curriculum. However, IT teachers are struggling to implement this vision in their classrooms due to the depth of content that needs to be covered for the concepts and topics constituting the four learning outcomes. The result is that the majority of Grade 12 IT learners on completing their final year lack the skills and knowledge outlined in the learning programme guidelines. Furthermore, very few are motivated to pursue a career in a computing related field.

Research studies in the Western Cape and Grahamstown (Eastern Cape) by Seymour *et al.* (2005) [in 11] and Jacobs and Sewry (2008) [11] respectively, were done to determine Grade 12 learners' inclinations to study Computer Science and Information Systems at tertiary level. It was found that a student's previous experience with computers affected their attitudes toward any future use [11]. Both studies found that learners with no access to computers at school were more inclined to study Computer Science than those with access to computers. The studies also found that learners that have negative perceptions of IT jobs available are less inclined to study Information Systems or Computer Science. Both studies found that learners do not know what Information Systems as a field of study is about although the perceptions of Computer Science were more accurate. Jacobs and Sewry (2008) [11] conclude that educational institutions need to promote accurate representations of IT related subjects and career fields to learners.

This paper analyses IT educator views to identify the problems faced by IT teachers and the reasons for the lack of interest in computing at tertiary level.

5. Findings

The lack of interest in IT as a subject at secondary school and as a degree course at tertiary education level has been noticed by IT educators, Department of Education subject co-ordinators and academics at universities across the country. In addition, industry is feeling the impact of this lack of interest in IT careers. Many industries have great difficulty filling ICT related positions and the demand for ICT professionals is increasing.

In an attempt to address this lack of interest and ensure the continued existence of IT as an FET subject within the National Curriculum Statement of South Africa, stakeholders are attempting to identify and address the current problems with the subject curriculum. Meetings for concerned IT educators are being arranged to discuss the issues and suggested short and long term solutions for the IT subject. However, in many cases, the meetings only contain a small representative sample of IT educators as the location and time of the meetings make it impossible for the majority of stakeholders, situated in all corners of the country, to attend.

Some success has been achieved through a computer studies mailing group where issues currently affecting IT can be discussed and debated in a public forum. Many more IT educators can participate in this mailing group as location and time are not an issue.

Interviews with IT teachers as part of a case study have highlighted many of the same comments and concerns raised on the mailing list and at various informal meetings with IT educators. The opinions, comments and suggestions of IT teachers have been considered and summarised for presentation in this paper in an attempt to highlight the impact the current IT subject at school is having on the future of ICT skills development.

Interviews with IT teachers have identified factors that are contributing to the lack of interest in IT as a subject at school and the current situation where many IT learners are struggling to cope with the subject content. A contributing factor to the lack of interest in IT as a subject choice in Grade 9 and the difficulties experienced by some IT learners after selecting the subject is that IT is a new subject in Grade 10 – many IT learners are not exposed to the content earlier. There are schools that do not provide computer skills classes to their Grade 8 and Grade 9 learners. These learners, at the end of Grade 9 are not exposed to any of the topics covered in IT. The result is that Grade 9 learners do not have a proper understanding of the content and learning outcomes of the IT subject and this may result in an uninformed decision to take or not to take IT.

Another factor that may deter Grade 9 learners from selecting IT as a subject is the lack of trained IT teachers. Younger teachers find better jobs in industry while the more experienced teachers are retiring. There is a shortage of suitably trained replacements for these teachers. Universities are experiencing a lack of interest by student FET teachers to become IT teachers. Qualified IT teachers capable of teaching the content in the current IT subject curriculum are hard to come by. The result of all these issues is that the IT teacher may not remain at the school for an extended period of time. A constantly changing subject teacher may make Grade 9 learners hesitant to select the IT subject.

Similarly, due to the lack of qualified FET teachers, teachers are appointed as IT teachers without the necessary qualifications or knowledge of the subject content. This is to the detriment of IT learners. A teacher without the required level of knowledge for the IT subject will also be a deterrent for Grade 9 learners and may result in learners who selected IT to drop the subject during Grade 10 or Grade 11.

Another very important factor deterring Grade 9 learners and causing IT learners to drop the subject is that IT is difficult. The majority of IT teachers believe that the current IT curriculum is overloaded or that too much is expected. However, there is no agreement on what topics are important and what content should be included in the curriculum.

Certain IT teachers have admitted to focusing mainly on Learning Outcome 4 which includes spreadsheets, databases and programming (either Java or Delphi is used as the programming language, depending on the province in which the school is located). Content in the other 3 learning outcomes, mainly theory, are covered very briefly if not left as a reading assignment or self-study. The programming content takes up most of the teaching time in the school year due to the amount of topics that need to be covered and the nature of the concepts. Many of the concepts are abstract and new to learners requiring IT teachers to dedicate much time to ensure that learners develop a proper understanding of the programming concepts and principles.

A suggestion by several IT teachers is to reduce the level of programming and provide more variety in the content. There is agreement that IT learners' perception is that IT is programming. Thus a Grade 12 learner considering studying further is deciding whether or not to continue programming as a career. This is a misconception created by the IT subject – due to the time spent on the programming content, learners are not properly exposed to all ICT related topics and career fields. The suggestion to reduce the level of programming by only focusing on basic programming principles and provide more variety, is aimed at exposing learners to other areas of computing such as ERP systems, business analysis, robotics, 3D graphics, etc. This should allow Grade 12 learners to make a better informed decision on whether or not to pursue a career in Computer Science or Information Systems (BSc or BCom).

A counter argument to reducing the level of programming and including other computer topics is that the subject content will still be overloaded and the result will be a grade 12 learner with a shallow knowledge of many different topics as the topics would not be able to be covered in great depth. *If IT should be made easier – by how much?* If the content of the IT subject and/or the level of difficulty are reduced, does this benefit the learners? Most IT teachers will state that learners either have an aptitude or motivation for IT – "*they just get it"* – or they don't. However, after taking IT as a subject a Grade 12 learner should be able to decide whether or not to follow a career in computing based on the content matter and whether or not it is appealing to them. Currently, Grade 12 learners finishing IT never want to experience programming again and this is the basis of their decision.

During the interviews and meetings with IT teachers, one positive factor was the determination and passion displayed by IT teachers toward the improvement of IT as a subject. IT teachers want to ensure that IT is a subject that will teach learners valuable and useful skills and knowledge that they can apply long after they have left school. It is important for business and government to realise that IT should prepare learners for and excite them about a career in ICT. However, steps need to be taken soon to stop the current decline in interest in IT as a subject in school and ICT as a career.

6. The Way Forward

A question posed by a mailing list contributor to guide the forum towards finding a solution is: *What is the purpose of IT at school?* The task team that originally compiled the IT subject curriculum intended for IT to *give the learner a view of IT and what it is all about; exposure to different developmental platforms and how these interact with the real world*. This is what an IT curriculum in school should do. However, this is currently not the result. Either the curriculum is not being implemented as intended or the curriculum cannot be implemented as intended and thus needs to be revised.

The findings summarised above indicate that IT teachers recognise the problems related to the IT subject. It is accepted that the IT subject curriculum needs to change and clear implementation guidelines need to be formulated to ensure a common understanding of implementation by IT teachers. However, no agreement on exactly what needs to change – what to remove and what to include – can be reached. The purpose of IT in school and the role of the subject in relation to business skills required and level of knowledge required by tertiary institutions as well as meeting the strategic objectives of Government, need to be clearly identified.

It is accepted that a revision of the IT subject curriculum is urgently required. It may, however, not be possible to implement curriculum changes in the very near future (next five years). Several short term solutions are suggested by related literature and based on the views and opinions of IT teachers. Firstly, the shortage of skilled IT teachers needs to be addressed and this should be a key objective in any plan or framework to develop e-skills in South Africa. Industries need to be made more aware of the importance and benefit of skilled and qualified IT teachers on the number and quality of IT learners pursuing a career in ICT. Funding and financial support for IT teacher training would provide incentives for FET teachers to train as IT teachers. This would increase the number of qualified IT teachers able to teach IT in schools.

IT teacher skills and knowledge can also be improved through workshops where IT teachers can meet and brainstorm different aspects of the curriculum that are problematic. This would allow IT teachers to form a common understanding of what is required, provide support for younger, inexperienced teachers and provide a knowledgebase of ideas and suggestions on ways of presenting and assessing different topics within the subject curriculum. The support provided for IT teachers in this way would have a positive effect on the presentation of subject content in class and hopefully motivate learners' interest in IT as a subject.

The workshops however will have to be arranged in areas accessible to teachers. The result would be many workshops for different clusters (possibly several within one city or town area) across the country. Each workshop group may formulate their own understanding of the curriculum and implementation thereof. Curriculum problems and lack of interest to study Computer Science among students in the US prompted the development of programs to tackle the problems [9]. The Computer Science Teachers Association (CSTA) is one such program that was launched by the ACM in 2005 to support Computer Science teaching. A similar association for IT educators in South Africa could ensure common objectives and provide IT educators with an official representative to voice their concerns and ensure wider participation and involvement of all IT educators in addressing IT subject concerns.

In addition to creating more interest in the IT subject amongst learners, the IT subject curriculum content should provide IT learners with the skills to meet the strategic objectives identified by the South African Government [16] to ensure national development. Appropriate ICT skills training within the IT subject curriculum should provide IT learners with appropriate e-skills, namely ICT practitioner skills, ICT user skills and e-business skills, in order to meet the socio-economic goals of the country. IT learners should be provided with knowledge and skills of how ICT can contribute to infrastructure development in terms of manufacturing and the transmission and processing of information as well as the use of economic and social infrastructure that will provide applications and services, such as e-government, to meet the needs of the country and people.

7. Conclusions

IT in South African secondary schools is having a negative impact on Computer Science and Information Systems degree choices after school [2, 11]. The skills and knowledge the IT subject tries to provide Grade 12 IT learners is decreasing their motivation to pursue a computer-related career and creating misconceptions about what different computer-related career fields entail. This, in turn, is contributing to an ICT skills shortage – impacting the development of ICT skills in South Africa. The difficulties faced by and lack of motivation of current IT learners, together with a shortage of skilled IT teachers, are deterring Grade 9 learners from selecting IT as a subject in Grade 10.

The skill level of learners entering IT at Grade 10 is not consistent, especially across schools. Schools are not offering learners in the lower grades a standard background knowledge that would assist a better understanding of IT concepts in Grade 10. The shortage of trained IT teachers is another serious negative impact on IT at school. The current situation in schools – overloaded curriculum and dwindling learner numbers – is worsening the situation. Student teachers are hesitant to train as IT teachers due to lack of learner interest in IT as a subject and the level of difficulty of teaching the subject due to its abstract nature.

The most important problem facing IT is the subject content matter. The subject must provide an IT learner with important skills and knowledge based on the four learning outcomes. However, there is a limit to the amount of time available in which this can be done. The current curriculum seemingly attempts to include too much content within the three FET phase years (Grade 10-12) – to the detriment of learners and the subject. There is no consensus amongst IT teachers as to the depth and breadth of the topics that should be included, especially with regards to the software development learning outcomes.

The IT curriculum needs to be addressed as a matter of urgency. A more structured approach is required where representatives of all stakeholders – IT educators, IT subject advisors, tertiary institutions, industry and Government – come together to discuss the way forward. The role of the IT subject in relation to industry and tertiary institutions and the skills required by a Grade 12 learner need to be clearly defined before the content can be revised. The shortage of skilled IT teachers also needs to be addressed. Incentives are needed to attract teachers to train as IT subject teachers.

The inclusion of ICT skills education at secondary school level through the IT subject will provide learners with work opportunities and quality of life due to the ICT skills and knowledge which they will obtain from the IT subject. If more learners have the opportunity and interest to develop and improve their ICT skills and knowledge at secondary school level it will contribute to a “digitally literate” workforce [15] and a future generation where the digital divide will be smaller.

The following question was posed by a mailing list contributor: *Should IT remain a subject at school?* Considering the impact IT is currently having on the future of ICT skills development in South Africa, perhaps the answer is **no**. On the other hand, the IT subject should be an opportunity to improve a learner’s skills and knowledge in different areas of computing and the three categories included as e-skills [15] for a future career in computing or simply to improve their quality of life, opportunities and benefits as a citizen of South Africa. It is in the interest of ICT skills development and the economic growth and development of South Africa, to ensure that the subject IT not only equips learners with valuable skills and knowledge but exposes learners to the opportunities that ICT skills and career fields can offer.

The development of e-skills in South Africa will not be a simple task and should be addressed on several fronts. Further research and investigation into the IT subject at schools as well as the need for structured, consistent programme to develop e-skills in learners earlier than Grade 10, should be addressed. Difficulties as well as gaps in the knowledge provided to IT learners need to be identified. Infrastructure to support and train IT teachers should be in place to ensure quality education. Tertiary institutions and industry should be consulted to identify the required e-skills competence level of Grade 12 IT learners. The knowledge, skills and competences required to meet the strategic objectives outlined by Government [16] should also be woven into the IT curriculum. The development of a National e-skills action plan for South Africa would need to address these different issues, as well as learn from the approaches of other countries [15], in order to fulfil the role of e-skills in the development of South Africa.

References

[1] ACM, Computer Science Curriculum 2008: An Interim Revision of CS 2001. December 2008.

[2] R. Babin, K. Grant, L. Sawal, Identifying Influencers in High School Student ICT Career Choice. 2008.

[3] M. Biggers, A. Brauer, T. Yilmaz, Student Perceptions of Computer Science: A Retention Study Comparing Graduating Seniors vs. CS Leavers. SIGCSE’08, 2008.

[4] N.G. Carr, IT Doesn’t Matter. Harvard Business Review, Vol. 81 No. 5, 41, 2003.

[5] L. Cohen, L. Manion, K. Morrison, Research Methods in Education. 6th Edition, Published by Routledge. ISBN 978-0-415-36878-0, 2007.

[6] Department of Education (DoE), National Curriculum Statement Grades 10-12: Information Technology. Pretoria: Government Printers, 2003.

[7] Department of Education (DoE), Learning Programme Guidelines: Information Technology. January 2008.

[8] N. Evans, Leading Information Technology in South Africa: A Unique Challenge. SIGMIS-CPR'06, 2006.

[9] O. Hazzan, J. Gal-Ezer, L. Blum, A Model for high School Computer Science Education: The Four Key Elements That Make IT!. SIGCSE'08, USA, 2008.

[10] E. Henning, Finding Your Way in Qualitative Research. Van Schaik, 2004.

[11] C. Jacobs, D.A. Sewry, Learner Inclinations to Study computer Science or Information Systems at Tertiary Level. Submitted to SACJ, 2009.

[12] L.E.C. Potter, L.A. von Hellens, S.H. Nielsen, Childhood Interest in IT and the Choice of IT as a Career: The Experiences of a Group of IT Professionals. SIGMIS-CPR'09, Ireland, 2009.

[13] SouthAfrica.Info [website], South Africa pushes 'e-skills'. Online article, 27 August 2007. Available online: http://www.southafrica.info/business/economy/development/skills-270807.htm (Accessed on 25 February 2010).

[14] J. Tandeur, J. van Braak, M. Valcke, Curricula and the use of ICT in Education: Two worlds apart? In: British Journal of Educational Technology, 2006.

[15] The European E-Skills Forum. *E-Skills for Europe: Towards 2010 and Beyond*. Synthesis Report, September 2004.

[16] The Presidency, RSA. Medium Term Strategic Framework: A Framework to Guide Government's Programme in the Electoral Mandate Period (2009-2014). Issued by Minister in the Presidency, June 2009.

H.2 SAICSIT 2010 Conference

Motivation and Learning Preferences of Information Technology Learners in South African Secondary Schools

Melisa Koorsse
Nelson Mandela Metropolitan University
P.O. Box 77000
Port Elizabeth, 6031
+27 41 372 2193
Melisa.Koorsse@nmmu.ac.za

Charmain B. Cilliers
Nelson Mandela Metropolitan University
P.O. Box 77000
Port Elizabeth, 6031
+27 41 504 2235
Charmain.Cilliers@nmmu.ac.za

André P. Calitz
Nelson Mandela Metropolitan University
P.O. Box 77000
Port Elizabeth, 6031
+27 41 504 2639
Andre.Calitz@nmmu.ac.za

ABSTRACT

The Information Technology (IT) subject presented in South African secondary schools is considered to be a difficult subject. The programming component of IT is believed to be the main cause of this difficulty. Learners who struggle with programming are unable to obtain above average marks in IT, as the programming component has the largest weighting in the IT subject framework. The aim of the research upon which this paper is based is to identify factors related to learner achievement in programming and the IT subject. The two areas that this paper investigates are learner motivation towards programming and the learning preferences of IT learners. The Motivated Strategies for Learning Questionnaire (MSLQ) is used to determine learner motivation and the Visual, Aural, Read/Write and Kinesthetic (VARK) Questionnaire is used to determine the learning preferences of IT learners. Both questionnaires provide interesting results and observations. The *self-efficacy for learning and performance* and *control of learning beliefs* motivational subscales seem to influence the performance of learners at the different schools. The VARK questionnaire results for this study indicate that the learning preferences of IT learners may influence understanding of programming concepts and also that it is best to present content to IT learners using a balance between all four modal groups (visual, aural, read/write and kinesthetic) to ensure that the learning preferences of all learners are met. The findings of this research indicate the impact that motivation and learning preferences possibly have on the understanding of programming concepts and overall achievement in programming and the IT subject. The contribution of this paper is the identification of the MSLQ and VARK questionnaires as methods that can be used to improve teaching strategies in South African secondary schools.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *Computer science education, Curriculum.*

General Terms

Performance, Human Factors

Keywords

Information Technology, Introductory Programming, Motivated Strategies for Learning Questionnaire (MSLQ), VARK Questionnaire.

1. INTRODUCTION

The Information Technology (IT) subject in South African schools has earned a reputation as a difficult subject because of the difficulty IT learners experience in understanding programming [8, 9]. The negative impression created amongst learners and their lack of interest in the subject affects the number of learners wishing to pursue a career in IT related professions [8].

IT learners, as novice programmers, are faced with similar difficulties to other novice programmers worldwide who are learning introductory programming concepts. The abstract nature of programming concepts, application of programming concept knowledge and the correct mental model of program code execution are a few of the factors that contribute to the difficulty faced by novice programmers [4, 5, 10, 11]. Learners that struggle to understand programming concepts become frustrated and are more likely to change to another "easier" subject [7, 17].

Motivation towards a subject plays an important part in influencing learner understanding of subject knowledge and their achievement in the subject [13]. Motivated learners are more eager to attempt challenging tasks, overcome any difficulty and enjoy their achievement. Learners with positive attitudes are also more likely to put more effort into their learning and involvement in learning tasks.

Different learning preferences of learners may also influence their understanding of subject knowledge [2]. A learner's preference for a particular learning modality is an indication of how a learner should learn, process and integrate information in different situations. If teachers are not able to deliver subject content knowledge in a format appropriate to cater for the different

SAICSIT '10, October 11–13, 2010, Bela Bela, South Africa.
Copyright 2010 ACM 978-1-60558-950-3/10/10…$10.00

learning preferences of learners, it could impact learner performance in the subject.

The determination of the influence of motivation and learning preferences on subject achievement is important in order to distinguish different factors that may be responsible for lack of learner performance in the subject.

This study determines the motivation of IT learners towards programming in particular using the Motivated Strategies for Learning Questionnaire (MSLQ) [15]. The motivational subscale values for different achievement groups (top 25%, middle 50% and bottom 25%) are compared and analysed to determine whether or not certain motivational subscales have an influence on learner achievement in the subject IT.

This study also determines the learning preferences of IT learners using the Visual, Aural, Read/Write and Kinesthetic (VARK) questionnaire [6]. The learners are ranked based on their final IT grade and divided into four different achievement groups (top 25%, upper middle 25%, lower middle 25%, bottom 25%). The average learning preferences for the four groups are compared to determine whether or not the learning preference of IT learners impacts their achievement in the subject IT.

The methodology used to investigate these issues is presented in the next section (Section 2). Section 3 presents the results of the MSLQ followed by a discussion and interpretation of the results. The findings and an interpretation of the results of the VARK Questionnaires completed by IT learners are presented in Section 4. The paper is summarised and future work is presented in Section 5.

2. METHODOLOGY

The main aim of this study is to investigate the contribution that can be made by determining the motivation and learning preferences of IT learners with regards to programming in South African secondary schools. The results presented are important to demonstrate how the MSLQ and VARK Questionnaire instruments can be used to provide information in addition to normal assessment of knowledge using formal assessment methods.

This paper sought to address the following primary research question:

> How does IT learner motivation towards programming and learning preferences relate to performance in the IT subject?

In order to answer this research question the research focused on answering the following two secondary research questions:

1. How does IT learner motivation toward programming compare to their performance in the IT subject?
2. How do IT learner learning preferences compare to their performance in the IT subject?

The research approach adopted by this study, which includes a discussion of the MSLQ and VARK Questionnaire instruments, is presented in the next section (Section 2.1). This is followed by an overview of the participants selected for this study (Section 2.2).

2.1 Research Approach

The results of the MSLQ and VARK questionnaires administered to IT learners are presented and the researcher's interpretations of the results are discussed in this paper (Section 3 and 4). The questionnaires used in the research study are not included with this paper due to their length; however, copies of the questionnaires can be obtained from the authors.

2.1.1 Motivated Strategies for Learning Questionnaire (MSLQ)

The Motivated Strategies for Learning Questionnaire (MSLQ) [15] is used to determine IT learner motivation toward programming. This questionnaire is designed as an instrument that can be used to investigate the nature of learner motivation and use of learning strategies. The questionnaire consists of 81 items divided into two sections: motivation and learning strategies. Items are rated using a 7-point Likert scale where 1 represents "not at all true of me" and 7 represents "very true of me". Only the motivation section which consists of 31 items was administered to IT learners participating in this study (Section 2.3).

The items in the motivation section of the MSLQ can be categorised according to six subscales:

- Intrinsic goal orientation (IG) – degree to which learner perceives participation in programming tasks for reasons such as challenge, curiosity and mastery.
- Extrinsic goal orientation (EG) – degree to which learner perceives participation in programming tasks for reasons such as grades, rewards, performance and evaluation by others.
- Task value (TV) – learner evaluation of how interesting, important and useful programming tasks are.
- Control of learning beliefs (CB) – learner belief that their own efforts to learn will result in positive outcomes.
- Self-efficacy for learning and performance (SLP) – learner expectancy for success and self-appraisal of their ability to master a task.
- Test anxiety (TA) – measure of learners' worry or emotionality component that could result in performance decrement.

The mean value of the ratings provided for the items in each of the six subscales, are calculated for each IT learner.

2.1.2 VARK Questionnaire

The VARK Questionnaire, designed by Neil Fleming [6] is used to identify preferred modes of learning of IT learners. A person's learning preference is the manner in which they most efficiently and effectively perceive, process, store and recall information about what they are attempting to learn [18]. The acronym VARK originates from four different learning preferences identified by Fleming [6]: **Visual** (sight), **Aural** (hearing), **Read/Write** and **Kinesthetic** (doing). The VARK questionnaire identifies learning *preferences,* which is unrelated to learning *strategies* or methods of learning (e.g. in groups or individually, at home or school).

The VARK Questionnaire administered to IT learners is the *Younger* version, which includes questionnaire items aimed at young participants such as learners. The questionnaire consists of

16 multiple-answer questions presenting everyday situations and participants can indicate one or more answers for each item.

2.2 Participant Selection

The research study took place in schools offering IT as a subject and the results of three of the participating schools are presented in this paper. The results for each of the schools were analysed independently due to the different school environments and IT teachers.

Table 1. Learner Distribution in Schools

School	No. Grade 11 learners
School A	11
School B	14
School C	16
Total	41

Table 1 indicates the number of Grade 11 IT learners who participated from each school. The number of learners participating was dependent on the class size and voluntary consent from learners and their parents/guardians.

3. IT LEARNER MOTIVATION TOWARDS PROGRAMMING

3.1 Findings

This section presents the findings of the MSLQ [15] that was administered to IT learners at the three participating schools. The discussion of the MSLQ results from IT learners is aimed at highlighting how the MSLQ may be used to provide possible explanations for learner performance in IT based on their motivation towards programming.

The motivational subscales of the MSLQ are analysed using exploratory data analysis. The final Grade 10 IT marks obtained by learners (now Grade 11) were divided into three groups, namely top 25%, middle 50% and bottom 25%. The bottom 25% group includes all marks below the lower quartile (quartile 1), the middle 50% group includes all marks equal to and above the lower quartile but below the upper quartile (quartile 3) and the top 25% group includes all marks equal to and above the upper quartile (quartile 3). For each subscale, the mean of the subscale values obtained by learners in each group were calculated as specified in the MSLQ manual [15]. Figure 1, Figure 2 and Figure 3 represent these mean values for each school as well as the overall mean value for each subscale.

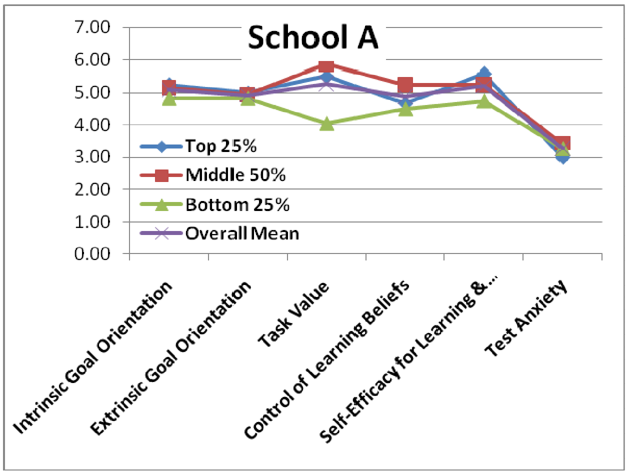


Figure 1. Mean Values for Motivational Subscales at School A

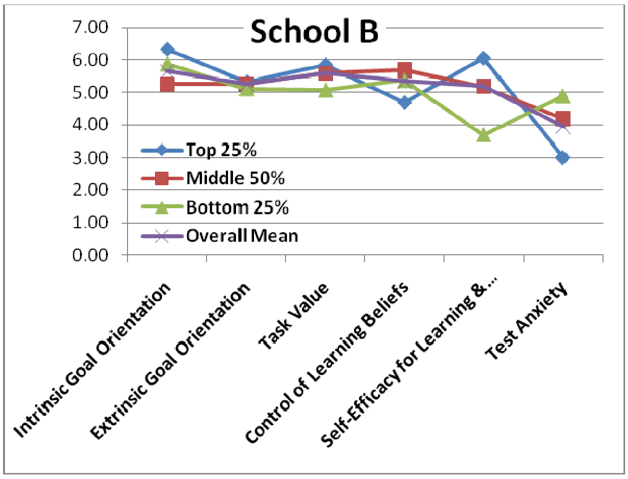


Figure 2. Mean Values for Motivational Subscales at School B

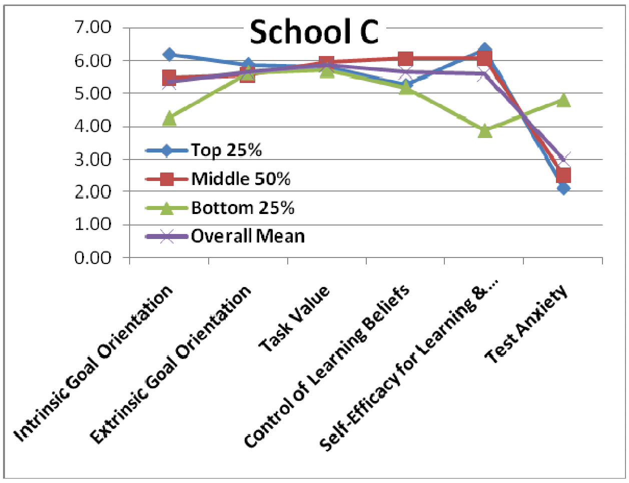


Figure 3. Mean Values for Motivational Subscales at School C

For the mean values of two of the subscales, namely *extrinsic goal orientation* and *self-efficacy for learning and performance* from all three schools, the bottom 25% had the lowest mean value while the top 25% had the highest mean value. The middle 50%

group at all the schools had the highest mean value for the *control of learning beliefs* subscale. The middle 50% group at School A (Figure 1) and School C (Figure 3) also had the highest mean value for the *task value* subscale. The middle 50% group at School A had a higher *test anxiety* value than the lower 25% group.

Self-efficacy for learning and performance is the subscale with the highest mean value for the top 25% group in both School A (Figure 1) and School C (Figure 3) and the second highest mean value for the top 25% group in School B (Figure 2). It is also the category with the highest and second highest mean value for the middle 50% group in School C and School A, respectively. This category has the lowest mean value for the bottom 25% group from School B and School C.

Control of learning beliefs is the subscale with the highest mean value for the middle 50% group in School B and the second highest mean value in School A and School C. *Intrinsic goal orientation* is the subscale with the highest mean value for the bottom 25% group in School B and the second highest mean value for the bottom 25% group in School A and School C.

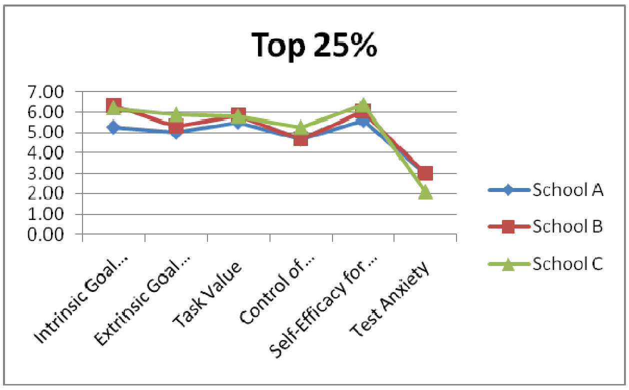


Figure 4. Top 25% Motivational Subscale Mean Values

Figure 4 indicates interesting trends in the lines representing motivational subscale mean values for the top 25% group at each of the three schools. *Intrinsic goal orientation* is rated higher than *extrinsic goal orientation*. Excluding *test anxiety, control of learning beliefs* is the lowest rated motivational subscale. *Task value* is also rated above *extrinsic goal orientation* and *control of learning beliefs.*

The trend lines representing the motivational subscale mean values for the middle 50% group at each of the three schools (Figure 5) do not indicate the lower mean value of *control of learning beliefs* as it appears for the top 25% group. The difference in mean values between the *intrinsic* and *extrinsic goal orientation* subscales is also smaller.

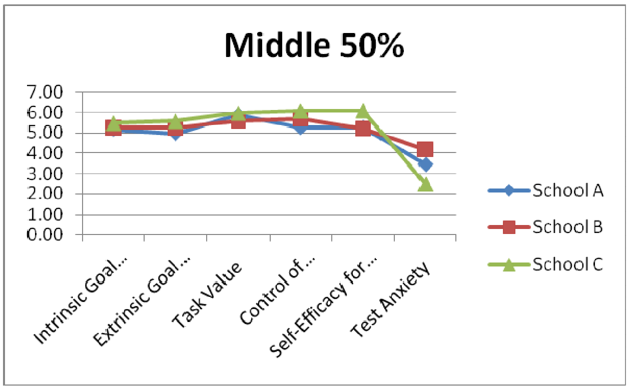


Figure 5. Middle 50% Motivational Subscale Mean Values

The trend lines representing the motivational subscale mean values for the bottom 25% group indicate a clear difference in mean values from the other two groups for specific subscales at each of the three schools. At School A (Figure 1) the bottom 25% group's mean value for the *task value* subscale differs greatly from the other two groups. At School B (Figure 2) and School C (Figure 3) the bottom 25% group's mean value for the *self-efficacy for learning and performance* subscale differs greatly from the other two groups. In addition, the *intrinsic goal orientation* subscale mean for the bottom 25% group at School C differs greatly from the other two groups.

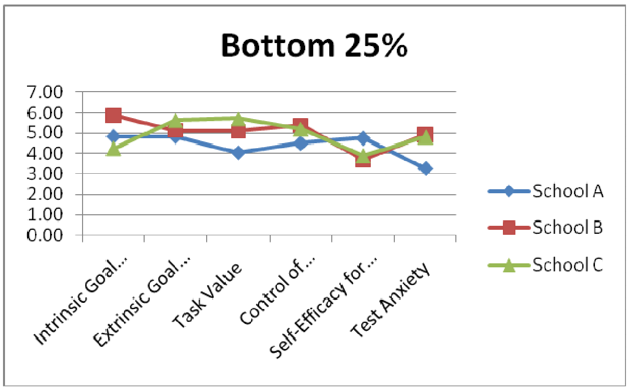


Figure 6. Bottom 35% Motivational Subscale Mean Values

The trend lines representing the motivational subscale mean values for the lower 25% group at each of the three schools (Figure 6) differ from each other unlike the trend lines of the top 25% group (Figure 4) and the middle 50% group (Figure 5). The lower SLP subscale value for School B and School C is, however, evident.

3.2 Discussion

The results of the MSLQ provide interesting analysis with regards to the different schools as well as the motivational subscale values of the three performance groups at each school.

The top 25% group at each of the three schools had *self-efficacy for learning and performance* as the motivational subscale with the highest (School A and School C) or second highest (School B) mean value. The middle 50% group at the three schools also had high mean values for the *self-efficacy for learning and performance* motivational subscale. Grade 11 IT learners with a final IT assessment mark in Grade 10 in the top 25% or middle

50% generally indicated a high expectancy for success and had a high regard of their own ability to master a task.

However, the top 25% group, unlike the middle 50% group, indicate less belief that their own efforts to learn will achieve success as *control of learning beliefs* was rated the lowest after the *test anxiety* subscale for the top 25% group but was the subscale with the highest mean value for the middle 50% group in School B and second highest mean value in School A and School C.

Control of learning beliefs is an indication of the belief by a learner that their own efforts to study will make a difference in their academic performance, as opposed to external factors such as the teacher. This could indicate that, although top 25% and middle 50% learners have confidence in their own abilities to achieve success, top 25% learners believe that good performance in IT is dependent on other factors beyond their own efforts to study unlike the middle 50% learners who believe that they alone control their academic performance.

The bottom 25% group at School B and School C had the lowest mean value for the *self-efficacy for learning and performance* motivational subscale which may indicate less belief in their abilities to master content knowledge than the other two groups. The bottom 25% group indicates a strong belief that their own efforts to study will impact their academic performance as indicated by the mean value for *control of learning beliefs* which is similar to that of the middle 50% group.

Table 2.

Group	**SLP** (Confidence in own abilities)	**CB** (Own efforts to study will impact performance)
Top 25%	✓	✗
Middle 50%	✓	✓
Bottom 25%	✗	✓

Table 2 summarises the preceding discussion with respect to the belief by each group of the impact of *self-efficacy for learning and performance* (SLP) and *control of learning beliefs* (CB) on academic achievement.

A possible explanation for the results (Table 2), however, indicating that the top 25% and middle 50% learners have strong confidence in their own abilities but that bottom 25% learners have less confidence in their own abilities is that programming is a difficult task that requires learners to understand abstract concepts [11, 12]. A lack of confidence in their own abilities to master programming concepts and skills could also be interpreted as frustration and dissatisfaction at not being able to master the programming content which could lead to a decline in interest for the subject [7].

Inability to get programs to "run" (compile and execute) successfully also decrease learner confidence in their own programming abilities. Learners unable to debug their programs [3, 12] or formulate the correct solutions to solve the problems [1] will have a lesser chance of obtaining the correct program solutions. IT learners also use programming development environments intended for use by professional programmers. These development environments make it very difficult for IT learners as novice programmers to successfully find errors in their code or assist in the development of proper code solutions [2].

Related to the lower confidence indicated by the bottom 25% IT learners is that the bottom 25% and middle 50% learners indicate a strong belief that their performance in IT is based on their own efforts to study, whereas the top 25% learners do not. A possible explanation for this result is that the top 25% learners may believe that they require the teacher's assistance and explanations to understand the abstract programming concepts presented and they may be more likely to ask questions in class and request the teacher's assistance more frequently when working on programming exercises. The middle 50% and bottom 25% learners may possibly be hesitant to ask the teacher for assistance, possibly due to a need to master the task or problems themselves, a slower work pace compared to the higher achievers thus running out of time to request the teacher's assistance or hesitance to indicate their lack of understanding to the teacher or their peers. If this is the case, a possible solution to this problem is the use of programming assistance tools.

Programming assistance tools are software programs designed specifically to assist novice programmers to understand introductory programming concepts. These tools can include features such as animations or step by step code execution to improve a novice programmer's understanding of programming concepts. If IT learners accept that they do not have to struggle to master the programming concepts and skills themselves but should use assistance, either in the form of the teacher, programming assistance tools or other self-instruction resources such as online tutorials, their understanding of programming and performance in IT may improve.

Another difference identified between the top 25% and middle 50% groups is the difference between the *intrinsic* and *extrinsic goal orientation* mean values. *Intrinsic goal orientation* has a definite higher mean value than *extrinsic goal orientation* for the top 25% group (Figure 4) whereas the difference is much smaller between these two subscales for the middle 50% group. *Intrinsic goal orientation* is on average rated higher by the top 25% learners compared to the middle 50% learners.

The top 25% group thus indicates a greater belief than the middle 50% group that their participation in the subject is for reasons such as it being challenging (IG) rather than for academic achievement, awards or acknowledgment (EG). A possible reason for this result is that the top 25% group has a greater confidence in their own abilities to master the content knowledge and perform well in the IT subject. This group may therefore be more confident to accept more challenging problems to test their programming knowledge and skills without it impacting their performance in the subject. The middle 50% group also indicate a strong confidence in their own abilities to master the programming concepts; however, they may possibly want to achieve better performance marks in the subject in order to achieve the level of the top 25% group.

The results from the different schools for the bottom 25% group differ with regards to *intrinsic* and *extrinsic goal orientation.* The *intrinsic* and *extrinsic goal orientation* relationship for the bottom 25 % group at School A is similar to the middle 50% group, while at School B the *intrinsic* and *extrinsic goal orientation* relationship for the bottom 25% group is similar to the top 25% group. At School C, however, *extrinsic goal orientation* is rated, on average, higher than *intrinsic goal orientation* by the bottom 25% group. A possible explanation is that better academic performance and achievement in the IT subject may be more important for the bottom 25% group at School C.

The MSLQ has provided interesting data with regards to performance and IT learner motivation. The significance of the differences between the top 25%, middle 50% and bottom 25% groups and the similarities of the of the top 25% and middle 50% groups at all three schools is not clear at this stage and not enough data exists from the results to determine learner performance based on their motivational subscale profile.

Further research would include increasing the sample size and administering the MSLQ to IT learners at other schools to determine whether or not the same similarities occur. Feedback from participants of this study to determine the accuracy of the MSLQ results and explore possible explanations for the similarities would increase the value of the contribution of this research.

4. LEARNING PREFERENCES OF IT LEARNERS

4.1 Findings

The presentation and interpretation of the VARK questionnaire results aims to indicate how the VARK questionnaire can be used to provide possible explanations for learner performance in IT based on their learning preferences.

Each item of the VARK questionnaire has four possible responses and respondents can select one or more of the responses. Each of the four responses describes one of the four learning preferences (visual, aural, read/write and kinesthetic). The number of selections for each learning preference is accumulated providing a total value for each. The learning preference of the IT learners, in this case, is thus indicated by the examining the totals for each learning preference.

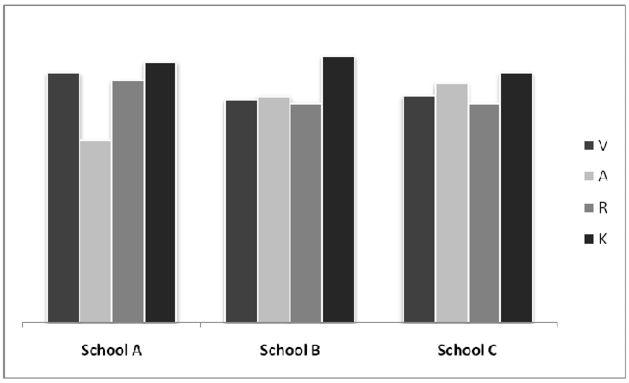


Figure 7. Learning Preference Profile for Grade 11 learners at each school

Figure 7 indicates the overall learning preference profile for each school. The scores obtained by IT learners for each learning preference were totaled to obtain an overall profile for IT learners at each school. The responses to the VARK Questionnaire indicate that IT learners at School A have a preference for visual, read/write or kinesthetic learning modes although the aural learning preference is not extremely weak.

The learning preference profiles of IT learners at School B and School C are evenly balanced. IT learners at School B have a slight preference toward the kinesthetic learning mode, while at School C there is a preference toward the kinesthetic and aural learning modes.

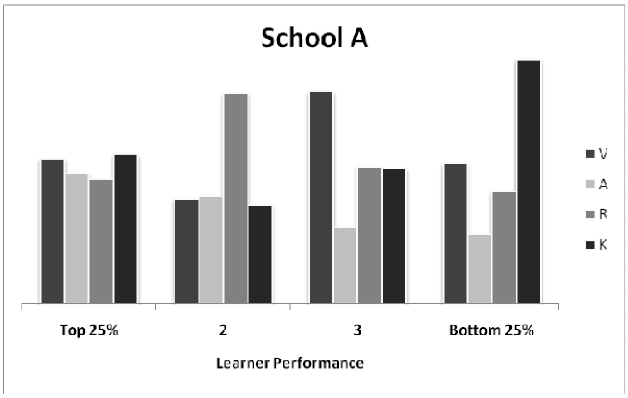


Figure 8. Learning Preference Profiles at School A

Figure 8, 9 and 10 provide further analysis of the IT learner learning preferences at each of the three schools. The IT learners are divided into four groups based on their final mark for the IT subject at the end of Grade 10. The learning profiles for each of these groups are calculated by adding the scores for each modality for all the learners in the group. The learning profiles for each group can then be compared (Figure 8, 9 and 10).

At School A (Figure 8), the learning preference profile of the top 25% group is balanced compared to the other three groups. The top 25% group does indicate a preference toward the visual and kinesthetic modalities. Group 2 indicates a strong preference for the read/write modality. Group 3 indicates a strong preference for the visual modality, while the aural modality is not strongly preferred. The bottom 25% group has a strong preference for the kinesthetic modality while the aural modality is also not strongly preferred. There is also a definite decline in the preference of the aural modality from the top 25% through to the bottom 25% group.

It would seem from Figure 8 that a preference for the aural modality is strongly related to better achievement (top 25%). A weaker preference for the aural modality but a strong read/write preference would still ensure above class average performance (group 2). A stronger visual preference with some preference toward the read/write modality characterises learners in group 3. The bottom 25% learners have a stronger preference toward the kinesthetic modality with aural and read/write preferences, which are strong for the top two groups, being the lesser modalities for this group.

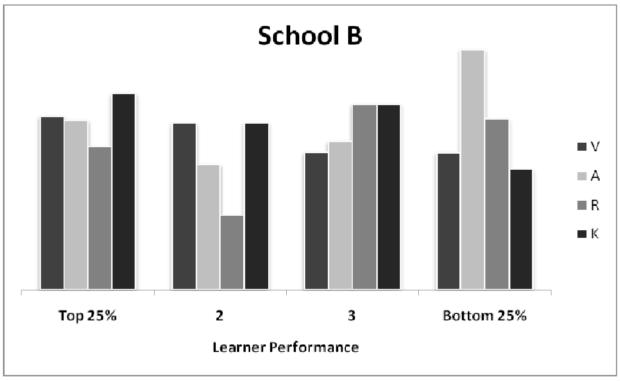


Figure 9. Learning Preference Profiles at School B

Figure 9 indicates the learning preference profiles of IT learners at School B. The top 25% group is generally balanced except for a slight preference for the kinesthetic modality, followed by the visual and aural modalities. Group 2 indicates a preference for the visual and kinesthetic modalities while the read/write modality is not preferred. Group 3 is also generally balanced but with stronger preferences for the read/write and kinesthetic modalities. The bottom 25% group indicates a strong preference for the aural modality.

At School B it would seem that a stronger preference for the visual and kinesthetic modalities as opposed to the read/write modality impacts performance. The teacher learning preference profile could not be obtained before the write-up of these results.

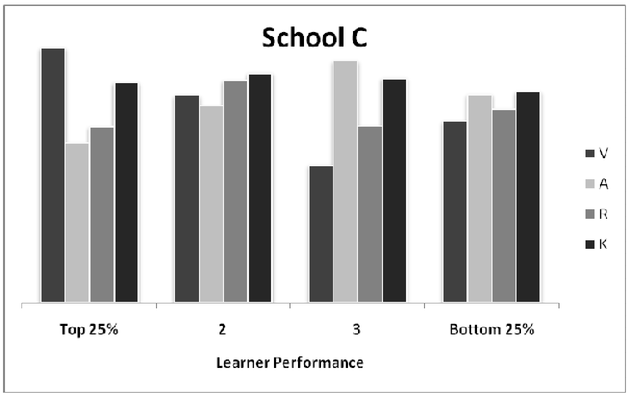


Figure 10. Learning Preference Profiles at School C

The learning preference profiles of learners at School C are presented in Figure 10. The top 25% group indicates a preference for the visual and kinesthetic modalities although the other two modalities are also strong. Group 2 indicates a balanced learning profile with slight preference for kinesthetic and read/write modalities. The aural and kinesthetic modalities are preferred by Group 3 with the visual modality being weaker than the other three. The bottom 25% group is also balanced with slight preference for the kinesthetic and aural modalities.

Group 3 and the bottom 25% group do have a preference for the aural and kinesthetic learning modalities. Neither the aural modality nor the kinesthetic modality is strongly preferred above any other modality in the bottom 25% group.

4.2 Discussion

The use of the VARK questionnaire to identify learning preferences of learners can be beneficial for learners and teachers. Learners that are aware of their learning preferences can make use of their preferred learning modality to better understand concepts and recall information for IT tests and examinations [6]. Teachers that are aware of the preferred learning preferences of different learners would be able to explain programming concepts and assist learners more effectively by using their preferred learning modalities [6].

The teacher at School A (Figure 8) would be advised to explain subject content using all four modalities with more emphasis on graphics (visual), demonstration (kinesthetic) and activities in the textbook (read/write), while lecturing or oral presentation (aural) should still be used. This may assist the lower three groups to improve their understanding of programming concepts and improve their performance in the IT subject.

At School B (Figure 9), the top two groups indicate a strong preference for the visual and kinesthetic learning modalities, group 3 indicates a strong preference for the read/write and kinesthetic modalities and the bottom 25% indicates a very strong preference for the aural followed by the read/write modality. The read/write and aural preferences of the last two groups may possibly be an indication that language may be a factor.

Learners who prefer reading information or listening to someone explain concepts to them but are struggling with IT may possibly be having difficulty understanding the English terms or their own language equivalent, used throughout the subject. Teachers would be encouraged to identify whether or not learners, who do not have English as their home language, are struggling with the English terms, particularly in the programming section. Alternatively, these learners may simply require more oral explanations of subject concepts. A possible method to overcome a learner's difficulty in understanding the English terms would be the use of the kinesthetic or visual modalities, depending on which is more strongly preferred. Demonstrations of how programming examples are solved and coded using an overhead projector together with animations to explain different programming concepts such as looping or the use of arrays would are possible kinesthetic methods that can be used. Programming assistance tools that provide animated explanations of how code is executed may also be useful to improve learner performance (e.g. Jeliot 2000 [2], Ville [16]).

At School C (Figure 10), a comparison of the learning profiles of all four learner groups does not seem to indicate an association between learning preferences and a learner's performance. However, if the bottom 25% profile, which is fairly evenly balanced, is ignored and only the first three groups are compared, two observations can be made. The first is that the preference of the visual modality is strong for the top 25% group and decreases in preference for group 2 and group 3. The second observation is that the aural modality is the least preferred for the top 25% group and is the preferred modality for group 3. The kinesthetic modality is also a strong preference for all four groups. The teacher at School C would thus be encouraged to use demonstrations of solving coding problems to explain programming concepts together with oral presentations of programming concepts to assist group 2 and group 3 learners improve their performance in IT.

The bottom 25% group at School C indicated a fairly balanced learning preference profile. This group should thus have no great difficulty understanding programming concepts no matter which modality is used by the teacher to explain the concepts to the learners. A possible conclusion is that the learning modality used will have no impact to improve the performance of learners in this group and that these learners and the teacher would need to focus on other factors which may be affecting achievement in the IT subject.

There will possibly be cases when the learning preference of learners does not have an impact on their performance. If all four groups of learners at a school all have balanced learning preference profiles then a change in the learning modality or combination of learning modalities used by the teacher or learner may not influence the learner's achievement in the subject. Instead, other issues such as learning strategies or motivation in the subject may need to be addressed. A similar case may exist if there is no significant correlation between a particular modality or combination of modalities and learner performance.

Further research to understand and explain the results of the VARK questionnaire would include administering additional questionnaires to IT learners to determine factors that may influence their performance such as home language.

5. CONCLUSION

This research study has focused on two secondary research questions to compare the performance of MSLQ and the VARK questionnaire results to IT learner performance in programming. These two research questions aim to answer the primary research question: How does IT learner motivation towards programming and learning preferences relate to performance in the IT subject? The discussion of the results of the MSLQ and VARK questionnaires in Sections 3.2 and 4.2, respectively, address these research questions.

The MSLQ provided results that may explain the influence of motivation towards programming on IT learner performance. In order to improve their performance with regards to programming, possible conclusions from the results indicate that IT learners should have a strong belief in their own ability to learn and understand programming concepts. IT learners also need to gain knowledge and insight from the IT teacher and other resources such as programming assistance tools to develop a better understanding of programming concepts. These findings address secondary research question 1, indicating that the MSLQ questionnaire can be used to compare IT learner motivation to performance in the subject. Further investigation is required to determine the significance of the difference in results obtained for the three performance groups.

The VARK Questionnaire results for each of the three participating schools have also provided interesting observations. Comparisons of the learning profiles of learners in different performance groups have identified possible reasons for learner performance based on the learning preferences of each group. Possible suggestions for improving learner performance have also been discussed. Generally, it is noted that the use of all four learning modalities by the teacher and learners is best to ensure proper understanding of programming concepts.

The findings indicate that the VARK questionnaire can be used by teachers to compare the performance of top learners to those who are struggling and devise teaching strategies accordingly. The findings have therefore addressed secondary research question 2.

This study has shown how the MSLQ and VARK questionnaires can be used to identify possible reasons for the difference in performance between IT learners other than the difficulty related to learning abstract programming concepts.

Shortcomings in the research exist, however, these have helped to identify areas for further research. The similarities in the motivational mean values between top learners and the differences between bottom learners can not accurately be explained. The accuracy of the findings of this research will be investigated by trying to increase the sample size and collecting data from IT learners at other schools. Additional biographical information and feedback from IT learner participants in this study would provide more insight into the results of the questionnaires.

The contribution of this investigation is to encourage IT teachers at these schools to be more aware of how to assist their IT learners to improve learner achievement in IT. In addition, other researchers are encouraged to use the MSLQ and VARK questionnaires in their research to investigate the impact of motivation and learning preference on subject or course achievement.

6. ACKNOWLEDGMENTS

We would like to thank the schools, teachers and learners who participated in this research study for their valuable contributions. We would also like to thank the National Center for Research to Improve Postsecondary Teaching and Learning, Ann Arbor, MI, for granting permission to use the MSLQ for this research.

7. REFERENCES

[1] Al-Imamy, S. Alizadeh, J. and Nour, M.A. 2006. On the Develoment of a Programming Teaching Tool: The Effect of Teaching by Templates on the Learning Process. *Journal of Information Technology Education*, Vol. 5, pp 271-283.

[2] Bednarik, R. and Fränti, P. 2004. Survival of Students with Different Learning Preferences. In *Proceedings of the Fourth Finnish/Baltic Sea conference on Computing Science Education*, Koli, Finland, October 1-3.

[3] Bednarik, R. and Tukiainen, M. 2006. An Eye-Tracking Methodology for Characterizing Program Comprehension Processes. In *Proceedings of the 2006 Symposium on Eye-Tracking Research Applications*, San Diego, California, March 27-29.

[4] Ben-Ari, M., Levy, R. and Uronen, P.A. 2000. An Extended Experiment with Jeliot 2000. In *Proc. Of the Program Visualization Workshop*, Porvoo, Finland.

[5] De Raadt, M. 2008. *Teaching Programming strategies explicitly to Novice Programmers*. Doctoral dissertation. University of Southern Queensland.

[6] Fleming, N. and Baume, D. 2006. *Learning Styles Again: VARKing up the right tree!* Educational Developments, SEDA Ltd., Issue 7.4, pp 4-7, November 2006.

[7] Gomes, A. and Mendes, A. J. 2007. An environment to improve programming education. In *Proceedings of the 2007 international Conference on Computer Systems and Technologies* (Bulgaria, June 14 - 15, 2007). B. Rachev, A.

Smrikarov, and D. Dimov, Eds. CompSysTech '07, Vol. 285. ACM, New York, NY, pp 1-6.

[8] Havenga, M. and Mentz, E. 2009. The School Subject Information Technology: A South African Perspective. In the *Proceedings of the SACLA'09 Conference*, Mpekweni Beach Resort, South Africa, June 29-July 1, 2009.

[9] Jacobs, C. and Sewry, D.A. 2009. Learner Inclinations to Study Computer Science or Information Systems at Tertiary Level. Submitted to *South African Computer Journal*.

[10] Kumar, A.N. 2006. Explanation of step-by-step execution as feedback for problems on program analysis, and its generation in model-based problem-solving tutors. In *Proceedings of Technology, Instruction, Cognition and Learning (TICL) Journal*.

[11] Lahtinen, E., Ala-Mutka, K., and Järvinen, H. 2005. A study of the difficulties of novice programmers. In *Proceedings of the 10th Annual SIGCSE Conference on innovation and Technology in Computer Science Education* (Caparica, Portugal, June 27 - 29, 2005). ITiCSE '05. ACM, New York, NY, pp 14-18.

[12] Lister, R., Simon, B., Thompson, E., Whalley, J. L., and Prasad, C. 2006. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Bologna, Italy, June 26 - 28, 2006). ITICSE '06. ACM, New York, NY, pp 118-122.

[13] Liu, M. 2005. Motivating Students Through Problem-based Learning. NECC.

[14] Mannila, L. and de Raadt, M. 2006. An objective comparison of languages for teaching introductory programming. In *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006* (Uppsala, Sweden, February 01 - 01, 2006). Baltic Sea '06, Vol. 276.

[15] Pintrich, R., Smith, A., Garcia, T. and McKeachie, W.J. 1991. A Manual for the Use of the Motivated Strategies for Learning Questionnaire (MSLQ). National Center for Research to Improve Postsecondary Teaching and Learning. Ann Arbor, MI: University of Michigan Press.

[16] Rajala, T. Laakso, M-J., Kailo, E. and Salakosi, T. 2007. Ville-A Multi-language Tool for Teaching Novice Programming. TUCS Technical Report No. 827, June 2007.

[17] Shuhidan, S., Hamilton, M. and D'Souza, D. 2009. A Taxonomic Study of Novice Programming Summative Assessment. In *Eleventh Australasian Computing Education Conference (ACE2009)*, (Wellington, New Zealand, January 2009). Conferences in Research and Practice in Information Technology (CRPIT), Vol. 95. M. Hamilton & T. Clear, Eds.

[18] Wehrwein, E.A. Lujan, H.L. and DiCarlo, S.E. 2007. Gender Differences in Learning Style Preferences among Undergraduate Physiology Students. In *Advanced Physiology Education*, Vol. 31, pp. 153-157, June 2007.

H.3 SACLA 2010 Conference

Programming in South African Schools: The Inside Story

Melisa Koorsse
Nelson Mandela Metropolitan University
P.O. Box 77000
Port Elizabeth, 6031
+27 41 372 2193
Melisa.Koorsse@nmmu.ac.za

André P. Calitz
Nelson Mandela Metropolitan University
P.O. Box 77000
Port Elizabeth, 6031
+27 41 504 2639
Andre.Calitz@nmmu.ac.za

Charmain B. Cilliers
Nelson Mandela Metropolitan University
P.O. Box 77000
Port Elizabeth, 6031
+27 41 504 2235
Charmain.Cilliers@nmmu.ac.za

ABSTRACT

The Information Technology (IT) subject presented in South African schools is considered a difficult subject. The number of Grade 9 learners opting to take IT as a subject in Grade 10 is decreasing and the number of learners continuing with IT to Grade 12 is declining. The main cause of the difficulty of the IT subject is the programming component. The programming component has the largest weighting in the IT subject framework resulting in learners who struggle with programming unable to obtain above average marks for the subject. The subject workload and lack of teacher support makes it difficult for IT teachers to present the programming content. The allocated class time for IT is not sufficient to ensure that learners grasp the basic concept knowledge required to develop a proper understanding of programming. This paper reports on the reasons why Grade 10 IT learners decided to choose IT as a subject. The programming concepts that Grade 11 learners find difficult and their approaches to programming are also presented in this paper. The research presented by this paper aims to identify problems faced by IT learners with regards to programming in South African secondary schools as the first phase of a study to assist IT learners to understand and apply programming concepts.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *Computer science education, Curriculum.*

General Terms

Performance, Human Factors

Keywords

Information Technology, Novice Programmers, Introductory Programming.

1. INTRODUCTION

Information Technology (IT) is one of the 29 school subjects included in the National Curriculum Statement of the South African Department of Education, offered at secondary school level from Grade 10 to 12 [5]. IT includes problem solving activities that require logical thinking, information management and communication. Computer application development using current object-oriented, net-centric development tools is also an important area of focus in IT [6].

IT in South African secondary schools has earned a reputation as a difficult subject because of the difficulty learners experience in understanding programming [9, 10]. Learners either avoid the subject IT or many learners that do attempt the subject change to another, "easier" subject such as Computer Applications Technology (CAT) before Grade 12 (their final year). The lack of interest in the subject and the negative impression created amongst learners affects the number of learners wishing to pursue a career in computer science or IT related professions [9]. This follows the trend of a decline in interest in computing professions recorded globally over the past 10 years [1, 10, 16].

The IT subject content is categorised into four main learning outcomes, as indicated in Table 1. The weighting for Learning Outcome 4 (Programming and Software Development) is the highest weighting of the four learning outcomes.

Table 1. Learning Outcome Time Allocation [6]

Learning Outcome	Weighting
Learning Outcome 1: Hardware & Systems Software	20%
Learning Outcome 2: E-Communication	10%
Learning Outcome 3: Social & Ethical Issues	10%
Learning Outcome 4: Programming & Software Development	60%

A learner's understanding and application of programming concepts impacts the learner's achievement in the IT subject [9]. Learners unable to understand programming concepts are frustrated and more likely to change to another subject [8, 18].

South African IT learners are faced with the same difficulties when compared to novice programmers worldwide. Studies have indicated that novice programmers may understand the syntax of programming concepts, but struggle to apply the concept knowledge when solving problems [4, 12]. The abstract nature of programming results in novice programmers struggling to develop the correct mental model of how a computer executes code. This has also been identified as a contributing factor to programming difficulty [2, 11]. A result of this misunderstanding is that code instructions are not written in the correct order [17].

Another important contributing factor to the inability to write code to solve a programming problem is the difficulty novice programmers have in reading and understanding code. A novice programmer, who is unable to read and understand code, will not be able to learn from examples or solutions, and thus be unable to generate code solutions to new problems [12, 16]. Teachers need to develop additional skills such as code comprehension techniques and not simply programming concept knowledge.

The difficulties faced by IT learners are worsened by the school teaching environment which differs from tertiary education facilities [9]. Four hours per week is the recommended time allocation for IT in the school timetable as indicated in the IT Learning Programme Guidelines [6]. This time is allocated for the entire programming subject content – not just Learning Outcome 4 (Table 1). IT teachers find it difficult to teach all the subject content in the required depth within the allocated time [9].

The IT teacher is usually the only “assistant” available to learners during practical programming sessions. IT teachers struggle to assist all learners who are having difficulty during a class period, which usually lasts 45-50 minutes and this includes time taken for learners to enter the class and prepare for the lesson. Learner involvement in sport and cultural activities after school make it difficult for IT teachers to arrange a time for learners to practice programming exercises with the teacher’s assistance. Currently, school subject timetables do not cater for the time required by programmers to master programming content [9].

People involved with the IT subject at school, Computer Science and Information Systems lecturers at universities across the country and businesses requiring skilled computer professionals are concerned about the lack of interest in IT as a subject at school. There are concerns with respect to the impact a shortage of skilled computer professionals may have on the South African economy and the country’s ability to remain competitive in the global market [14].

It would only be possible to implement any revisions to current IT subject content in five years time [informal meeting to address IT subject concerns, 29 October 2009, Pretoria, South Africa]. In the interim, ways of assisting IT learners to cope with and understand the current curriculum are needed to ensure that interest in IT as a subject at school can be maintained until improvements can be implemented.

The problems investigated in this study are the decline in interest in IT as a subject and the difficulty of the IT subject, mainly due to the programming component. The methodology used to investigate these problems is presented in the next section (Section 2). Section 3 presents the results of the IT learner surveys followed by a discussion and interpretation of the results in Section 4. The paper is summarised and future work is presented in Section 5.

2. METHODOLOGY

2.1 Purpose of the Study

The main aim of this study is to investigate the factors influencing the learning of IT programming in South African secondary schools. The results presented are important to identify ways of assisting IT learners to understand programming concepts within the South African secondary school environment.

This research sought to explore 2 primary questions:

1. What are the reasons why learners decide to take IT?
2. What do IT learners find difficult with regards to programming?

2.2 Research Approach

The research approach adopted by this study was influenced by the research methodology of the overall doctoral study of which this study forms a part. The doctoral study is focused on providing South African IT learners with techniques and tools to assist their understanding of programming concepts.

Close-ended surveys using questionnaires were used to gather information from IT learners. Grade 10 IT learners were approached to complete a close-ended questionnaire aimed at identifying factors influencing learners’ decisions to take IT. The questionnaire was designed by the researcher based on questions used by Biggers *et al.* [3] in a similar study to investigate university student perceptions of Computer Science (CS) and the reason why students decided to major in CS. Grade 10 learner perceptions of programming were not investigated in this research study as the survey was done in the beginning of the academic year. IT is selected as a subject for three years from Grade 10 to Grade 12 and Grade 10 IT learners generally have no prior programming experience.

A close-ended questionnaire to determine learner perceptions of different aspects of programming was designed by the researcher and completed by Grade 11 IT learners based on their experience of programming in Grade 10. The questions in this questionnaire were designed to investigate learner perceptions of the difficulty of different programming concepts, their comprehension of programming examples and exercise solutions, and the application of solutions for simple problems to similar, but more complex problems. Another aim of the questionnaire was to determine learners’ ability to debug errors in code solutions and to plan solutions to programming problems. The results are presented and the researcher’s interpretations of the results are discussed in this paper.

The questionnaires used in the research study were not included with this paper due to their length; however, copies of the questionnaires can be obtained from the authors.

2.3 Sample

The research study is conducted in schools offering IT as a subject that are situated in the Port Elizabeth metropolitan area, Eastern Cape Province, South Africa. The amount of interaction required by the researcher with schools participating in the doctoral study has restricted the selection of schools participating to the Port Elizabeth area, due to the proximity to the researcher.

In South Africa, two programming languages are used to teach programming to learners taking IT, namely Java and Delphi. The programming language used is determined by the province the school is situated in. In the Eastern Cape, Delphi is the programming language used.

The results of three of the participating schools are presented in this paper. The results for each of the schools were analysed independently due to the different school environments and IT teachers as these may influence learner perceptions and the generalisability of certain of the results.

Table 2. Learner Distribution in Schools

School	Grade 10	Grade 11
School A	23	11
School B	18	16
School C	18	14

Table 2 indicates the number of learners who participated in each grade from each school. The number of learners participating was dependent on the class size and voluntary consent from learners and their parents/guardians. Unfortunately, no schools from disadvantaged communities in the Port Elizabeth metropolitan area could be included in the study as none of these schools offered IT as a subject.

3. FINDINGS

3.1 Why Learners Selected IT as a Subject?

Learners were asked to indicate factors influencing their decisions to select IT as a subject. Learners could select more than one option from a list provided (Table 3). The frequency distribution graph in Figure 1 summarises the learner responses. It is clear from the graph (Figure 1) that learners are responsible for their own subject choice although others may provide input into the decision making process. Most learners indicated an interest in computers and in programming.

Table 3. List of Possible Factors Influencing the Decision to Take IT (more than one could be selected)

IT was one of my first choice subjects
I had to decide between IT and another subject
Taking IT was my own decision
IT was recommended/suggested by someone else
If IT was taught by another teacher I might not have selected it
I enjoy working on a computer (e.g. games, typing, creating documents)
I have some knowledge of programming or what it is all about
I am interested in programming

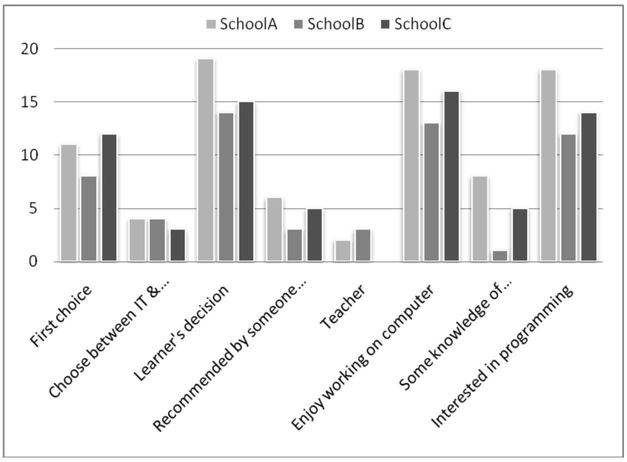


Figure 1. Factors Influencing Learner Decision to Take IT

Table 4. List of Possible Main Reasons for Taking IT

To understand more about the different aspects and topics related to computing in general
To be able to understand how a computer works (hardware)
To improve my understanding and use of different computer programs or applications
To be able to develop my own software programs
To play LAN games with friends in class
To have more access to a computer at school
Other

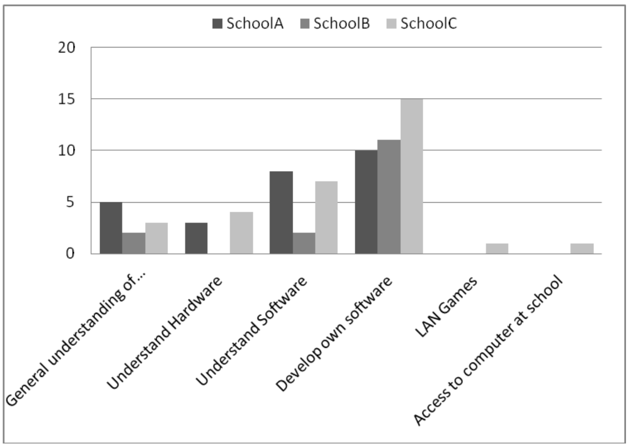


Figure 2. Main Reason for Selecting IT

The frequency distribution graph in Figure 2 summarises the main reason why learners selected IT as a subject. Table 4 provides the list of possible reasons provided to learners. It is clear from Figure 2 that most learners selecting IT did so in order to learn how to develop their own software programs.

Learners were also asked to indicate, using a 7-point Likert scale where 1 is "*Strongly Disagree*" and 7 is "*Strongly Agree*", the perceived difficulty of IT when compared to other subjects and whether they believed they would do well in IT. It should be remembered that, in general, Grade 10 learners have no experience of programming. Figures 3 and 4 are the frequency distribution graphs for the perceived difficulty of IT and expected success in IT, respectively, as indicated by Grade 10.

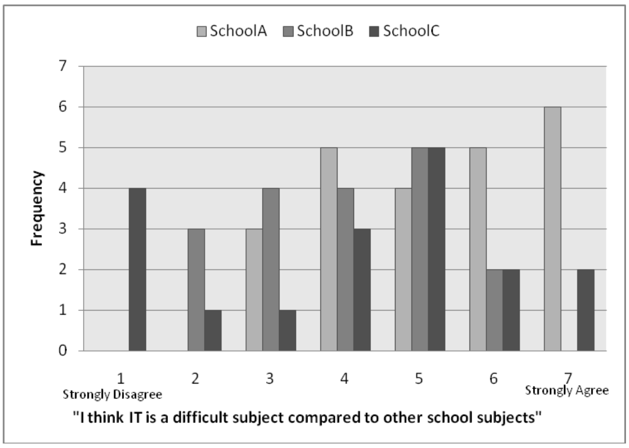


Figure 3. Perceived Difficulty of IT

A comparison of the perception of the difficulty of IT at each of the three schools indicates a different profile for each school (Figure 3). If each school is analysed individually it is noted that participants from School A tend to perceive IT as difficult compared to other subjects. Participants from School B did not strongly agree or disagree, with responses clustering around being undecided. Participants from School C indicated mixed responses with two "spikes" in the results. Quite a few students tended to strongly disagree with the statement that IT is difficult compared to other subjects, while other students tended towards agreeing with the statement but not as strongly as the disagreements. If the number of participants from School C selecting a value less than 4 is compared with the number selecting a value greater than 4, slightly more participants tended to agree with the statement (9 participants as opposed to 6 disagreements).

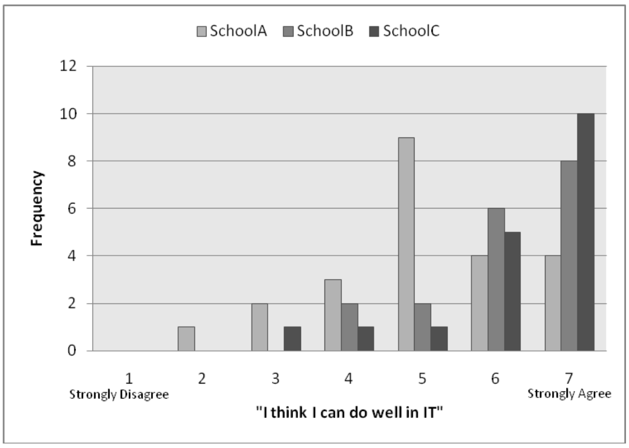


Figure 4. Perceived Success in IT

The results of the participants' perceived success in IT indicates a general trend amongst participants from all three schools. Except for 4 participants disagreeing with the statement, most participants agreed with the statement that they could do well in IT. Participants from two of the schools indicated this very strongly while participants from School A seemed to be more apprehensive.

3.2 What Aspects of Programming are Difficult?

Learners were asked to rate the difficulty of 22 different programming concepts (Table 5) and skills derived from the IT Learning Programme Guidelines [6] using a 7-point Likert scale where 1 is "*Extremely Easy"* and 7 is "*Extremely Difficult"*. Each participant was also asked to indicate the concept that they perceived as the most difficult. Most learners indicated that *procedures* is the most difficult concept to understand. Debugging of code was also indicated as difficult by the learners.

Table 5. Programming Concepts Rated by Grade 11 Learners

Variables
Input (Getting information from the user)
Output (Displaying information to the user)
If statements
Case (Delphi) or switch (Java) statements
For loops
While loops
Repeat (Delphi) or do While (Java) loops
String handling
One dimensional arrays
Two dimensional arrays
File handling
Accessing a database
SQL statements
Procedures
Functions
Correct use of parameters
Objects & classes
Problem solving
Algorithms
Planning (use of pseudocode to plan solution before coding)
Debugging (finding errors in the code)

The remaining questions were formulated in such a way as to determine the degree to which learners were able to understand simple code exercises and solutions, apply their understanding of simple solutions to more complex problems, debug errors in code and plan solutions to programming problems. Participants had to rate their responses using a 7-point Likert scale where 1 is "*Strongly Disagree"* and 7 is "*Strongly Agree*".

Table 6 summarises the results for the three schools by providing mean and mode values. It is clear that participants believe they are able to do simple exercises and understand provided solutions for simple exercises. In general, participants indicated a positive belief in their abilities to combine programming concepts to solve complex problems. As would be expected, the ability to apply simple solutions to complex problems was rated less than the ability to understand simple exercises; however, there is only a slight difference. Further investigation is needed to determine how significant the difference is.

Table 6. Results of Questionnaires to Grade 11 Learners

	Understanding of simple code	Application to complex problems	Debugging	Planning
Median				
School A	5.70	4.82	4.03	5.73
School B	5.90	5.36	5.2	4.3
School C	6.05	5.84	4.85	4.51
Mode				
School A	6	5	1	5
School B	6	5	6	5
School C	7	7	7	7

The mode and median results describing the ability to debug errors in code showed varied responses from participants. The analysis of results for specific questions used provided more insight. Two of the questions related to debugging were: *I understand compiler error messages* and *I use compiler error messages to find syntax errors in my code*. The mean and mode values of the ratings provided for these two questions were generally higher for the second question than the first. It can be deduced that participants attempt to use compiler error messages to debug code but this is difficult if they are not able to understand the message.

The two questions related to debugging that were on average given the lowest ratings were related to the use of output to display variable values at critical points in the program and the use of breakpoints to step through code. Both these questions investigate methods that novice programmers can use to find errors in the logical processing of their code.

Planning of solutions has also been identified as a factor influencing novice programmer ability to program successfully [7, 17]. In the feedback received learners indicated that most of the time they understood what the expected program was required to do and that they were then able to write code to solve the programming problem. However, most participants disagreed with the statement *I write down a non-code solution before trying to write code* and they disagreed with the statement *I use comments in my program code to describe different sections of code.*

4. DISCUSSION

4.1 Selecting IT as a Subject

An important result of the questionnaire to Grade 10 learners regarding their choice of IT as a subject, is that learners make the subject choice decision. Input may be provided by parents, teachers and friends but the final decision is made by the learner. Campaigns to advertise IT as a subject for Grade 9 learners during the subject choice process should be focused on the learners. If learners are motivated to take IT as a subject and have a clear understanding of what the subject entails, chances are good that the learner will choose IT as a subject. The impact of programming on the IT subject choice decision is quite clear. Learners exposed to programming before Grade 10 may be more inclined to select IT as a subject

The indication of learners wanting to develop their own software as the main reason for IT being selected as a subject raises two points. The first point to consider is that the choice of IT for software development is fueled mainly by future career fields, including programming and mechatronics. For others, a computing career may not be their first choice at present but they are keeping their options open. Learners who end up struggling with IT and programming will then think twice about pursuing a career in programming or a computer related field.

The second point that needs to be considered is that the interest in developing their own programs as the main reason for selecting IT clearly indicates the learners' association of IT with programming. The high weighting of programming in the presentation and assessment of IT means that learners struggling with programming would be struggling with IT. If learners are struggling with or are no longer interested in programming, there is not enough focus or time spent on other aspects of computer related topics in the IT subject to allow the leaner to find another area of interest within the computing field as well as maintaining the same level of achievement as is expected of the learner. Learners struggling with programming thus have very few options but to discontinue the subject, especially if academic achievement is affected [9].

Participants' perceptions of the difficulty of IT as a subject when compared to other subjects, is varied. This is to be expected as the questionnaire was completed in the first quarter of the academic year. The learners do not have any experience in the subject and are only starting with programming concepts such as input and output methods and variables. The results of this question will be compared to the results of the same question at the end of Grade 10. The difference in learners' perceptions of difficulty can then be compared to what they have experienced in the subject during the year.

Similarly, Grade 10 learners' perceptions of their success in the subject will be compared to their actual success (assessment marks) and self-reported evaluation of their success at the end of the academic year. These results would provide insight into the impact that IT and programming have on learners' motivation and interest in the subject.

4.2 Difficulty of Programming

An interesting result from the questionnaire survey to Grade 11 learners is the indication by learners that *procedures* (or methods in Java) is a difficult programming concept to understand. In the results of this research study it was indicated to be the most difficult concept. *Procedures* and *functions* together with the use of *parameters* have traditionally been difficult concepts to grasp for learners in South African schools.

Initially, this result was interesting as the work schedule of the IT teachers indicated that *procedures* and *functions* would only be taught after the questionnaire was given to learners. There was doubt as to whether or not learners had been taught this concept and, if not, whether learners then misinterpreted the meaning of *procedures* on the questionnaire. The questionnaire only provided the commonly used term *procedures* and no description was provided.

Grade 11 learners at two of the participating schools identified *procedures* as difficult. After analysing the results of the questionnaire, the respective teachers were consulted about whether or not they had already taught *procedures* to their learners. School B indicated that the learners were busy with the concept of *procedures* when the questionnaire was completed, while School C indicated that learners were introduced to *procedures* and *functions* at the end of Grade 10 in preparation for a more thorough presentation of the concepts in Grade 11. The feedback from the schools thus clarified that learners were referring to the *procedures* programming concept as intended by the questionnaire.

Procedures and *functions* require learners to change from coding in an event handler (such as a button click or form create) to creating their own methods and calling these methods when needed. A proper understanding of *procedures* and *functions* requires learners to have an understanding of code reusability and modular code. These are concepts that can be extended further in object oriented programming. Further investigation is needed to identify whether the difficulty in understanding *procedures* and *functions* is related to the actual implementation of these concepts in a programming language or a lack of proper understanding of the purpose of these concepts.

A more definite indication by participants with regards to difficult concepts or programming skills was that participants regarded debugging code as difficult. Programming knowledge and debugging ability are important factors determining whether or not a programmer would be able to correct any code errors [2]. Inability to debug code could lead to frustration on behalf of the learner and lower their motivation and interest in the subject [18]. Learners thus need assistance or techniques to find errors in their code.

One form of assistance is when the learner uses techniques and tools to identify the errors themselves. Compiler error messages assist programmers to find syntax errors. Logical errors can be determined using strategic outputs of variable values or breakpoints to step through or into the code. However, if learners are unable to understand the compiler error messages or are unable to use breakpoints or output variable values, as indicated by their responses in the questionnaire, they will be unable to solve many of these errors successfully without other forms of assistance. The software development tools used in the schools are designed for use by professional programmers. The compiler error messages and the functions provided to assist programmers to debug their code are often too complicated for novice programmers to understand or use [2, 15, 17].

The second form of assistance which learners turn to is the teacher or their peers. If too many learners in the class are unable to solve the problems themselves, the result is the teacher spending the lesson moving from one learner to the next identifying and explaining errors. Assisting learners in the class does provide teachers with an opportunity to identify what different learners are struggling with and assess their knowledge of the concepts. However, teachers would need to beware of assisting learners too quickly with simple errors such as missing semicolons. Without a strategy of assisting learners to debug code themselves, teachers are not enabling learners to develop their debugging skills.

It is also sometimes impossible for the IT teacher to assist all learners within the class lesson which is usually less than an hour long. Many learners tend not to continue with the exercises if they are struggling to find errors. The teacher is also sometimes not able to go through all the solutions with the class and it is thus left to the learners to ensure that they understand the solutions and attempt to complete the exercises again on their own. The time allocated to the subject and the large amount of content that needs to be covered thus makes it very difficult for IT teachers to ensure that learners have enough practice doing programming examples and exercises.

Regarding the planning of programming solutions, most participants indicated that they did not first formulate non-code solutions. The fact that learners do not use non-code solutions can not be interpreted as an indication that those learners are struggling with programming. Research studies have discussed the impact of planning on programming success [7, 17]; however, the results in this research study would need to be correlated with assessment marks to determine the impact that not using non-code solutions has on programming ability. The formulation of non-code solutions to plan the solution to a programming problem can assist struggling novice programmers to decompose a problem into simpler parts which may then be easier to solve. The planning of programming solutions is a critical problem solving step that many novice programmers tend to omit [17, 18] and skill that IT teachers cover very briefly when introducing programming to Grade 10 learners.

5. CONCLUSION

This research study has focused on two research questions, namely, to identify the reasons for learners deciding to take IT as a subject and the difficulties with regards to programming concepts experienced by IT learners. The questionnaire responses received from Grade 10 and Grade 11 participants provided preliminary findings to answer these questions.

An interest in programming and software development was identified as the main reason why learners decided to take IT. A possible follow up study would be to identify whether the number of learners taking IT would increase or decrease if programming was not the main focus. The learners who participated in this research study will be surveyed again towards the end of the year to analyse the effect that exposure to the programming content has had on their interest in the subject and their perceptions of the difficulty of IT compared to other subjects.

The use of *procedures* and finding bugs in code were identified as two of the three most difficult issues in programming by respondents in a survey conducted by Lahtinen *et al.* [12]. These results are supported by our research where *procedures* and debugging were identified as difficult programming concepts by participants. Further investigation will be conducted into the reason why *procedures* are identified as difficult by learners.

Debugging is a skill that is essential to ensuring that the correct programming solutions are produced. This study has highlighted that this debugging needs to be focused on and learners can be assisted by providing techniques and tools that can be used to evaluate code and find errors.

Grade 12 learners may be approached to complete the questionnaire provided to the Grade 11 learners. The Grade 12 learners would be able to report on their experiences of programming in Grade 10 and Grade 11 and thus provide more insight into the programming difficulties experienced.

It is hoped that this study has produced interesting and useful results which will help IT teachers to be more aware of how IT learners experience and approach programming and be better able to assist learners. In addition, it is hoped that this study has made others aware of the difficulties related to the teaching of IT in South African secondary schools and encourages further research on this topic.

6. ACKNOWLEDGMENTS

We would like to thank the schools, teachers and learners who participated in this research study for their valuable contributions.

7. REFERENCES

[1] Babin, R., Grand, K. and Sawal, L. Identifying Influencers in High School Student ICT Career Choice, 2008.

[2] Ben-Ari, M., Levy, R. and Uronen, P.A. An Extended Experiment with Jeliot 2000. In *Proc. Of the Program Visualization Workshop,* Porvoo, Finland, 2000.

[3] Biggers, M., Brauer, A. and Yilmaz, T. Student Perceptions of Computer Science: A Retention Study Comparing Graduating Seniors v. CS Leavers. In *Proceedings of the SIGCSE'08 Conference*, 2008.

[4] De Raadt, M. *Teaching Programming strategies explicitly to Novice Programmers*. Doctoral dissertation. University of Southern Queensland, 2008.

[5] Department of Education. *National Curriculum Statement. Grades 10-12 (General).* Information Technology, 2003.

[6] Department of Education. *National Curriculum Statement. Grades 10-12 (General).* Learning Programme Guidelines. Information Technology, 2008.

[7] Fidge, C. and Teague, D. Losing Their Marbles: Syntax-Free Programming for Assessing Problem-Solving Skills. In *Proc. Eleventh Australasian Computing Education Conference* (ACE 2009), Wellington, New Zealand. CRPIT, 95. Hamilton, M. and Clear, T., Eds. ACS, pp 75-82, 2009.

[8] Gomes, A. and Mendes, A. J. An environment to improve programming education. In *Proceedings of the 2007 international Conference on Computer Systems and Technologies* (Bulgaria, June 14 - 15, 2007). B. Rachev, A. Smrikarov, and D. Dimov, Eds. CompSysTech '07, Vol. 285. ACM, New York, NY, pp 1-6, 2007.

[9] Havenga, M. and Mentz, E. The School Subject Information Technology: A South African Perspective. In the *Proceedings of the SACLA'09 Conference,* Mpekweni Beach Resort, South Africa, June 29-July 1, 2009.

[10] Jacobs, C. and Sewry, D.A. Learner Inclinations to Study Computer Science or Information Systems at Tertiary Level. Submitted to *South African Computer Journal*, 2009.

[11] Kumar, A.N. Explanation of step-by-step execution as feedback for problems on program analysis, and its generation in model-based problem-solving tutors. In *Proceedings of Technology, Instruction, Cognition and Learning (TICL) Journal*, 2006.

[12] Lahtinen, E., Ala-Mutka, K., and Järvinen, H. A study of the difficulties of novice programmers. In *Proceedings of the 10th Annual SIGCSE Conference on innovation and Technology in Computer Science Education* (Caparica, Portugal, June 27 - 29, 2005). ITiCSE '05. ACM, New York, NY, pp 14-18, 2005.

[13] Lister, R., Simon, B., Thompson, E., Whalley, J. L., and Prasad, C. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Bologna, Italy, June 26 - 28, 2006). ITICSE '06. ACM, New York, NY, pp 118-122, 2006.

[14] Merkofer, P. and Murphy, A. The E-Skills Landscape in South Africa. In *Zeitschrift für Politikberatung*. Published by VS Verlag für Sozialwissenschaften. ISSN: 1865-4789, 2010.

[15] Pendergast, M.O. Teaching Introductory Programming to IS Students: Java Problems and Pitfalls. In *Journal of Information Technology Education*, Vol. 5, pp 491-515, 2005.

[16] Potter, L.E.C., von Hellens, L.A. and Nielsen, S.H. Childhood Interest in IT and the Choice of IT as a Career: The Experiences of a Group of IT Professionals. In the *Proceedings of the SIGMIS-CPR'09 Conference*, Ireland, 2009.

[17] Rongas, T., Kaarna, A. and Kalviainen, H. Advanced Learning Technologies. In *Proceedings of IEEE International Conference on Advanced Learning Technologies* (30 Aug.-1 Sept, 2004) *ICALT'04,* pp 678 – 680, 2004.

[18] Shuhidan, S., Hamilton, M. and D'Souza, D. A Taxonomic Study of Novice Programming Summative Assessment. In *Eleventh Australasian Computing Education Conference (ACE2009),* (Wellington, New Zealand, January 2009). Conferences in Research and Practice in Information Technology (CRPIT), Vol. 95. M. Hamilton & T. Clear, Eds, 2009.

H.4 SACLA 2011 Conference

Programming Assistance Software Tools to Support the Teaching of Introductory Programming

Melisa Koorsse
Nelson Mandela Metropolitan University
P.O. Box 77000
Port Elizabeth, 6031
+27 41 372 2193
Melisa.Koorsse@nmmu.ac.za

André P. Calitz
Nelson Mandela Metropolitan University
P.O. Box 77000
Port Elizabeth, 6031
+27 41 504 2639
Andre.Calitz@nmmu.ac.za

Charmain B. Cilliers
Nelson Mandela Metropolitan University
P.O. Box 77000
Port Elizabeth, 6031
+27 41 504 2235
Charmain.Cilliers@nmmu.ac.za

ABSTRACT

Novice programmers find learning to program difficult and debugging has also been identified as a difficult task for novice programmers. Novice programmers struggle to develop accurate mental models of programming concepts and processes, have difficulty understanding how a computer executes instructions and struggle to apply the syntax rules of high-level programming languages. Different programming assistance software tools have been developed to assist novice programmers with their understanding of programming concepts. Programming assistance tools use different techniques to assist novice programmers, including visualisation and animation techniques, and drag and drop interfaces. A number of programming assistance tools has shortcomings, for example, not supporting all introductory programming concepts.

This paper identifies several different programming assistance software tools that are freely available for use by novice programmers. The programming assistance tools are evaluated using selection criteria that can be used to select programming assistance software tools for use in introductory programming courses. The selection criteria are formulated from a literature review of introductory programming as well as research conducted with Information Technology (IT) learners in South African secondary schools. The research presented in this paper aims to provide IT teachers and introductory programming lecturers with a list of programming assistance software tools that are freely available for introductory programming courses and subjects, selection criteria that can be used to evaluate the programming assistance tools and a discussion of some of the shortcomings of programming assistance tools that need to be considered when selecting tools for introductory programming courses.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education – *Computer Science education, Curriculum.*

General Terms

Performance, Human Factors, Languages.

Keywords

Introductory Programming, Novice Programmers, Information Technology, Programming Assistance Tools.

1. INTRODUCTION

Expert or professional programmers possess problem solving abilities [1, 2] that are essential for developing software that is fast and scalable. These abilities are developed from programming experience gained over an average of 10 years [30].

Novice programmers, including Information Technology learners at South African secondary schools, find learning to program to be a difficult task [32]. The reason for this is that novice programmers need to learn how to understand and solve a problem, formulate a solution in a structured form (algorithm) and then write the algorithm in a specific programming language [34]. Programming can be a difficult task if programmers are unable to plan solutions [31], lack understanding of programming concepts due to the abstract nature of these concepts [23] and lack understanding of how a computer executes code [5].

The teaching and learning of programming concepts can be supported with programming assistance tools. Research in novice programming has suggested and developed programming tools [26] to enhance comprehension of algorithms and computer programs, assist with code debugging and assess programming knowledge and skills. The programming assistance tools use different techniques such as visualisation, animation or drag-and-drop interfaces to improve the conceptual understanding of programming concepts [2].

Educators and students may be unaware of the different programming assistance tools that can be used to support understanding of programming concepts. Certain tools have educational deficiencies and do not support all of the content presented in an introductory programming course.

This paper discusses the difficulties novice programmers experience when learning to program (Section 2). Criteria that can be used to select programming assistance tools to support novice programmers are formulated (Section 3) and used to evaluate several programming assistance tools freely available for use by novice programmers (Sections 4 and 5). The paper is concluded and future work is presented in Section 6.

2. PROGRAMMING DIFFICULTIES

Programming is a complex activity that requires a novice to learn non-trivial facts, skills and concepts that are new to them [2].

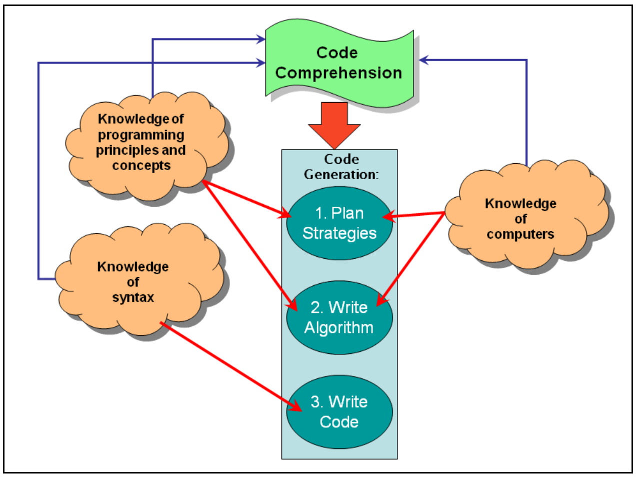


Figure 1. Knowledge and skills required by programmers.

Figure 1 summarises the relationship between code comprehension and generation and the different types of knowledge a novice programmer requires.

Code generation involves 3 main steps (Figure 1):

1. A given problem statement or requirements set must be considered in order to decide upon the programming strategy to use.
2. An algorithm to solve the problem should be formulated, often using pseudocode.
3. The algorithm is translated into the code of the programming language being used. The program is tested and changed as necessary until the original set of requirements to solve the problem, are met.

The three steps outlined above can only be achieved by a programmer who is able to apply programming knowledge and strategies and who has the ability to comprehend as well as generate code [8, 30].

2.1 Programming Knowledge

Programming knowledge includes knowledge of programming concepts and principles, knowledge of computers and knowledge of programming language syntax (Figure 1) [26]. Knowledge of programming concepts and principles is an understanding of how different concepts are implemented and why. For example, how a *for*-loop works or the purpose of a variable. Knowledge of computers includes an understanding of how computer events occur and can be handled by the code (for example a mouse-click or button press). Knowledge of syntax is required in order to implement a solution in a particular programming language.

All three of the above knowledge areas are important for code generation. If a programmer is unaware of the different programming concepts it would be difficult to plan a suitable solution. A well designed solution will not run successfully if the syntax used is incorrect. Code not executed in the correct order or when certain events occur will result in an incorrect program solution, regardless of whether there are no syntax errors or if the correct programming concepts have been implemented.

2.2 Programming Strategies

A programming strategy is the way in which programming knowledge is applied to solve a particular problem [8, 30]. A suitable programming strategy is required for the first step of code generation (Figure 1). A programmer who has an understanding of programming knowledge, but who is unable to use their knowledge to solve or transfer solutions from simple problems to more complex problems, lacks an understanding of programming strategies [8, 23]. Similarly, a programmer lacks programming strategy or problem solving ability if the programs compile and run yet do not solve the problem due to logic errors [1].

In general, more time is spent teaching programming language knowledge than programming strategy [1, 26]. Novice programmers tend to combine the steps of the code generation process (Figure 1) as they attempt to write the algorithm in a particular programming language [11, 31].

2.3 Code Comprehension and Generation

A novice programmer should have adequate knowledge in all three programming knowledge areas (Section 2.1) to be able to comprehend and generate code successfully (Figure 1). A novice programmer that lacks ability in one or more of the knowledge areas will struggle to generate a code solution.

Program comprehension is described as a "*cognitively complex skill*" [3, 24]. Novice programmers reading code should be able to identify the knowledge such as concepts used in the solution and the strategy applied to produce the solution [8].

A correlation has been shown to exist between code comprehension and code generation [8, 25]. Code comprehension is regarded as an important skill required for successful programming [3]. Novice programmers that are not able to read and understand code are unable to write similar code [25]. This can be a problem as new concepts are explained to novice programmers using practical examples. If a novice programmer is unable to read and understand code solutions, they will be unable to build their knowledge of programming concepts and strategies to solve real-world problems [24, 25]. The ability to read and understand code is moreover an important skill required for finding logical errors in code [25].

2.4 Other Contributing Factors

Other factors that can contribute to make learning to program difficult include the teaching approach used, the programming language and environment and specific programming concepts that are difficult to understand.

2.4.1 Teaching Approach

A novice programming student is guided by the teacher presenting the programming course when learning to program. Teachers need to present the course with a balance between the "how to" and "why" explanations. Overemphasising "how to", for example, how to use an *if*-statement in a particular problem, may result in students being unable to transfer what they have learnt. Their underlying skills and concept knowledge would be lacking [2]. Overemphasising "why" would provide students with a theoretical knowledge of the underlying programming principles. However, the theory would need to be accompanied by practical experience demonstrating how the principles are applied.

Teachers also need to assist novice programmers to create a proper mental model of different programming concepts, especially the more abstract concepts [32]. Each person has a preferred learning style, differing abilities, learning speeds and attitudes or motivations toward the subject [22, 23] which would need to be taken into consideration by the teacher when assisting individual students. However, many teachers use one teaching approach for all students [1] thus not catering for the learning requirements of individual students.

2.4.2 *Programming Language and Environment*

The programming language and development environment tool used by novice programmers when learning to program can also contribute to the difficulties experienced [26]. Certain programming languages are too complex to use to explain or teach introductory programming concepts [5, 17, 27]. If a novice programmer is having difficulty understanding a programming concept, an explanation of the concept using a code example should not be further confusing. Professional programming environments may also overwhelm novice programmers by presenting them with functionality and interfaces that are only used by professional programmers [5, 17, 27].

2.4.3 *Specific Programming Concepts*

Certain programming concepts are more abstract than others and are thus more difficult for novice programmers to understand [16]. Abstract concepts have no related explanation in real life, making it difficult for novice programmers to develop an accurate mental model of these concepts.

A literature survey has identified several specific programming concepts which novice programmers struggle to understand. *Recursion* is identified by three different research studies [14, 23, 32] as a difficult concept for novice programmers to understand. *Abstract data types* such as *arrays* are identified by two separate research studies [14, 23]. Novice programmers also seem to have difficulty understanding and using *methods* or *procedures and functions* [12, 14].

Another concept that teachers specifically find difficult to teach is object-oriented programming (OOP). The difficulty related to OOP has been associated with the paradigm shift from structured methods and not the actual concepts [17]. Two approaches have been recommended for teaching OOP: “objects first” and “objects last”. The “objects last” approach is the most common instruction method. This approach starts with simple concepts and programs and gradually advances to more difficult concepts [7]. This provides a gentle learning curve which allows novice programmers to incrementally build their programming knowledge.

A problem with teaching OOP last is that a paradigm shift is needed for students to switch from the procedural style of programming to the OOP style of programming. This shift has been identified as the cause of the difficulties related to teaching OOP last [7, 17]. The solution is the “objects first” approach. The “objects first” approach introduces concepts such as string handling and looping within the OOP environment. However, a novice programmer’s first experience of programming is a difficult mental challenge because they are faced with learning the basic programming concepts and syntax together with the complexities of OOP [7].

A review of the literature has indicated greater support for the “objects first” approach [7, 17, 18, 20]. This is evident from many tools that have been developed to promote “objects first” and ensure that students are not impacted by the difficulties of “objects first” highlighted earlier.

3. SELECTION CRITERIA

Table 1 lists selection criteria that can be used to select a programming assistance tool to support novice programmers learning to program. All items followed by an asterisk are derived from the programming difficulties identified in Section 2. The remaining items are derived from the results of surveys administered to Information Technology (IT) learners and teachers in South African secondary schools [21, 22].

The left hand column (Table 1) lists programming concepts that an introductory programming course should include. This list is derived from the list of programming concepts that Information Technology (IT) learners in South African secondary schools are required to learn [9]. The bold items in the list are programming concepts that have been identified as difficult to understand based on the results of a survey administered to IT teachers and IT learners [21].

The programming skills and knowledge items in the right hand column of Table 1, originate from the programming skills required by novice programmers identified in the literature survey presented in Section 2. Programming assistance tools should allow novice programmers to develop code comprehension skills, promote problem solving ability using appropriate strategies, assist understanding of code execution and allow syntax knowledge and knowledge of programming principles and concepts to be improved.

Every person has a preferred learning preference. Four learning preferences are identified by Fleming and Baume [10], namely visual, aural, read/write and kinesthetic. A persons learning preference could be only one or a combination of the four. The IT teacher/learner survey results indicated that programming assistance tools should cater for at least 2 of the learning preferences to assist individual users.

A programming assistance tool should be constructivist to allow a novice programmer to “build” their knowledge of programming concepts using the tool and promote self-study. Novice programmers also need assistance to formulate a code solution before actually writing the program code.

A tool that uses visualisation and/or animation techniques can increase interest and motivation [29]. Error handling refers to whether or not a tool can detect errors in code, that is, compile a code solution. Simple errors messages and/or suggestions to correct errors refer to the way in which the tool assists novice programmers to detect and correct errors in the code.

Novice programmers also struggle to apply programming strategies used to solve simple exercises, to solve more complex exercises.

Table 1. Selection criteria for Programming Assistance Tools.

Selection Criteria	
Concepts	***Programming skills & knowledge:***
Variables	Code comprehension*
Input (getting information from the user)	Promotes problem solving using strategies*
Output(displaying information to the user)	Code execution*
If-statements	Syntax knowledge*
Switch statements	Knowledge of programming principles & concepts*
For-loops	***Teaching/Learning approach:***
Repeat-until/do-while loops	Constructivist (promote self-study)
While-do loops	Feedback to guide solution creation
String handling	***Learning Preferences:***
Procedures*	Visual
Functions*	Aural
One-Dimensional Arrays*	Read/Write
Two-Dimensional Arrays*	Kinesthetic
File handling	***Other:***
Accessing a database	Simple & complex examples (scaffolding)
SQL statements	Error handling
Correct use of parameters	Simple error messages/suggestions to correct errors
Objects & classes*	Visualisation/Animation
Problem solving*	***Programming Language (e.g. Java, C#, Delphi (Pascal), etc.)***
Debugging*	

The programming assistance tools should be able to assist novice programmers with the syntax of whichever programming language the novice programmer is learning to program in. Care should be taken to select a tool which implements code or can be adapted to implement code that is the same or similar to that of the programming language being used by the novice programmers. Differences in syntax or the manner in which concepts are implemented can be confusing and result in making the learning process more difficult.

4. PROGRAMMING ASSISTANCE TOOLS

Programming assistance tools (PATs) are specifically designed to support novice programmers learning to program [26]. PATs can assist novice programmers to develop their understanding of programming concepts. [31] states that the ideal PAT would be able to support several features including problem solving, algorithm design, assist with learning syntax for a particular programming language, and partial compiling to quickly check output and operation of a code block.

PATs can also make use of visualisation and animation techniques. Most programming concepts, data structures and algorithms are abstract [31]. Visualisation techniques can be used to help novice programmers develop an accurate mental model of programming concepts (Section 2).

Several PATs have been identified from literature and will be discussed briefly.

4.1 RobotProg

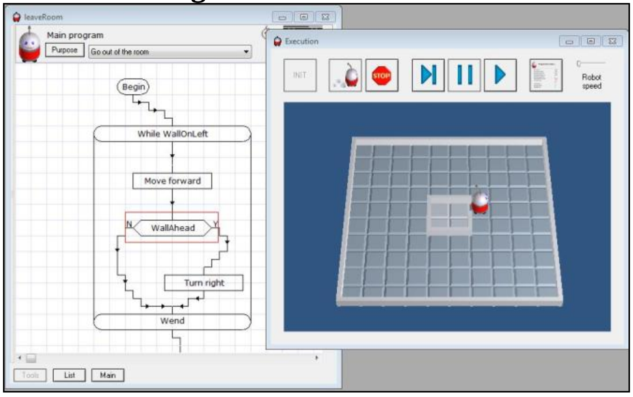


Figure 2. Execution of RobotProg flowchart.

RobotProg is a PAT in which the user creates a flowchart (Figure 2) by dragging and dropping icons representing programming concepts. When the program created is executed, it controls a robot to perform specific tasks (Figure 2).

Different levels of difficulty can be specified in RobotProg. The simple programming concepts are available in the lowest level.

More complex programming concepts are available for use in the flowchart as the level is increased.

Users are also challenged to complete tasks such as finding a corner or picking up a ball. The RobotProg tool is able to detect whether or not the task has been completed successfully. The RobotProg interface can be complicated for novice programmers to understand in the beginning. Users are not able to view any code generated by the flowcharts.

4.2 BlueJ

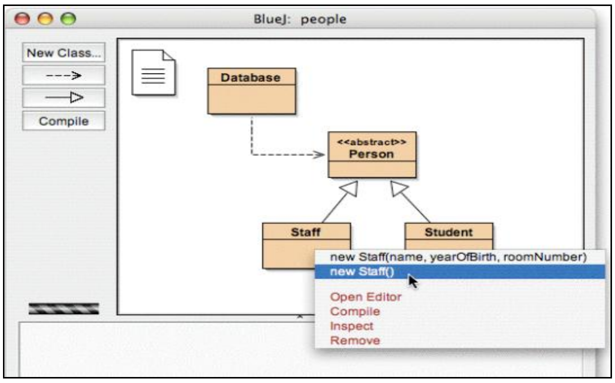


Figure 3. Creating UML-like class diagrams in BlueJ.

BlueJ is a tool that uses an objects first approach (Section 2.4) to introduce novice programmers to the concept of objects and classes. In BlueJ, the objects and classes concepts are demonstrated without the user having to write any code [17, 19].

The advantages of BlueJ are that it is interactive and simple to use. It uses visualisation to help novice programmers understand objects and classes. UML-like class diagrams are used to provide a graphical overview of project structures (Figure 3). A disadvantage is that exercises would need to be designed by the teacher, based on the functionality provided by BlueJ.

BlueJ generates code in Java. If users want to view the corresponding code implementations associated with the objects and classes visualisations, an understanding of the Java programming language is required. BlueJ also does not provide assistance with the understanding of other programming concepts such as conditional statements or looping, although they can be implemented.

4.3 Greenfoot

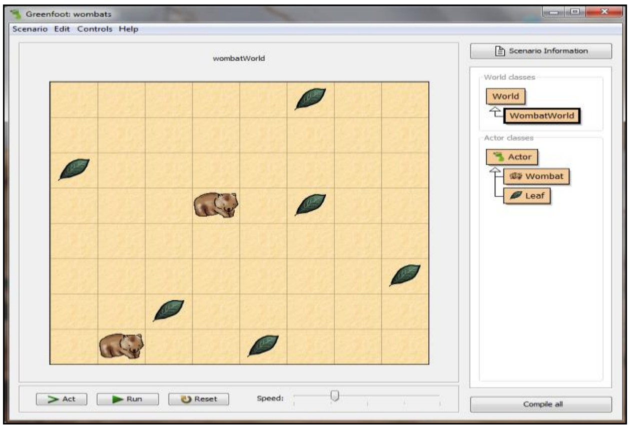
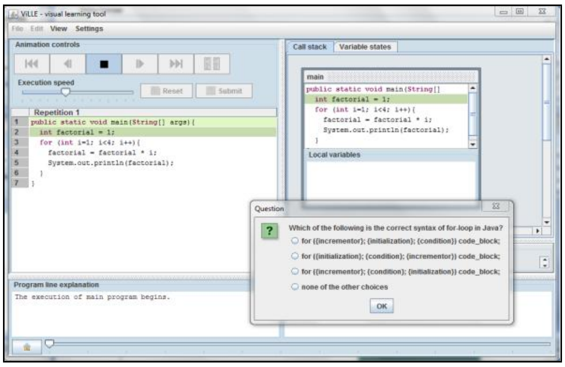


Figure 4. Greenfoot main screen with objects.

Greenfoot is a PAT that is used to teach object-oriented programming to secondary school learners [15]. Users can easily create different microworlds that are visually appealing and easy to interact with.

Users can interact with Greenfoot objects directly (Figure 4). Changes in the position and appearance of objects can be observed directly. Classes associated with Greenfoot objects are visible on the main screen (Figure 4) and code for the different objects can also be modified by the user. The coding language used is Java. Similar to BlueJ, Greenfoot only assists with the understanding of the implementation of objects and classes.

4.4 Ville



```
public static void main(String[] args){
  int factorial = 1;
  for (int i=1; i<4; i++){
    factorial = factorial * i;
    System.out.println(factorial);
  }
}
```

The execution of main program begins.

Which of the following is the correct syntax of for-loop in Java?

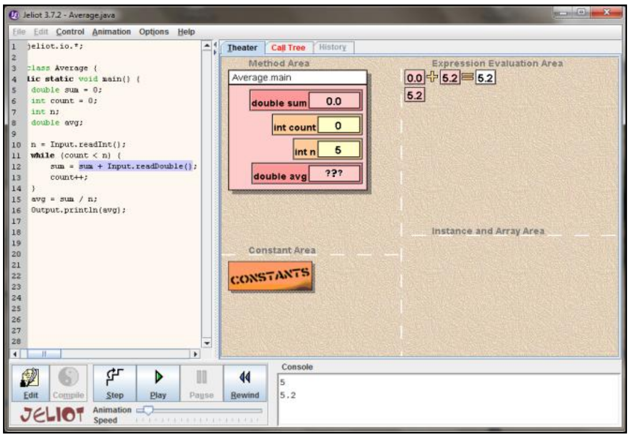
○ for ({incrementor}; {initialization}; {condition}) code_block;
○ for ({initialization}; {condition}; {incrementor}) code_block;
○ for ({incrementor}; {condition}; {initialization}) code_block;
○ none of the other choices

Figure 5. Execution of program in Ville with question being posed to user.

Ville is a language-independent programming tool [28]. Code execution is demonstrated using visualisation techniques. Ville has the syntax rules for several programming languages built in, including Java, Python, PHP, javascript, C++ and pseudocode. New languages can be added using the syntax editor.

A user can control the speed of execution as well as step forward or back through the program code. Explanations for program lines are provided and Ville can be set up to ask the user questions about the current code being executed (Figure 5).

4.5 Jeliot



```
jeliot.io.*;

class Average {
lic static void main() {
 double sum = 0;
 int count = 0;
 int n;
 double avg;

 n = Input.readInt();
 while (count < n) {
     sum = sum + Input.readDouble();
     count++;
 }
 avg = sum / n;
 Output.println(avg);
```

```
5
5.2
```

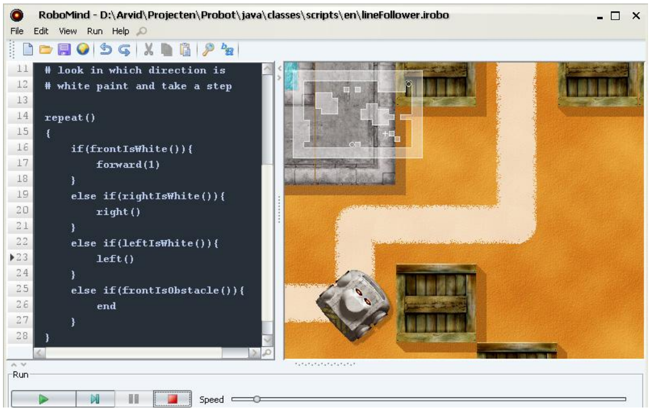
Figure 6. Visualisation of program execution using Jeliot.

Jeliot animates programs to assist novice programmer understanding of introductory programming concepts. Jeliot is capable of animating most of the Java language, including object allocation [4].

Visualisation and animation techniques are used in Jeliot to assist novice programmers to develop an accurate mental model of programming concepts during code comprehension [4]. The current line of execution is indicated to users during execution of the program (Figure 6). Four areas in the animation are used to indicate current variable values, expression evaluations, value of constants and the allocation of and reference to objects and arrays.

The standard Jeliot program assists novice programmers with the understanding of Java programs [4]. Jeliot 3 has been redesigned to separate the interpretation and animation of Java programs. This means that Jeliot 3 can be used to animate programs written in another programming language.

4.6 RoboMind


RoboMind - D:\Arvid\Projecten\Probot\java\classes\scripts\en\lineFollower.irobo
File Edit View Run Help
Run
Speed

```
# look in which direction is
# white paint and take a step

repeat()
{
    if(frontIsWhite()){
        forward(1)
    }
    else if(rightIsWhite()){
        right()
    }
    else if(leftIsWhite()){
        left()
    }
    else if(frontIsObstacle()){
        end
    }
}
```

Figure 7. RoboMind program executing.

RoboMind has been designed as a tool that can be used as a first introduction to programming without any prerequisites. User program a robot to move around and interact with objects in a map world using a simple educational programming language called ROBO (Figure 7).

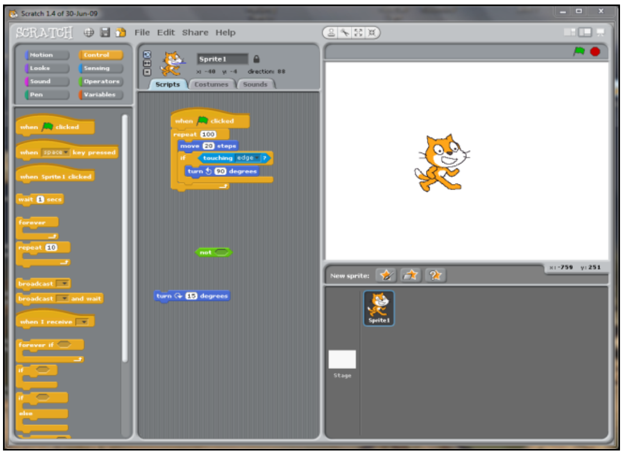
The RoboMind environment is freely available and the RoboMind 2.2 development environment is available as open source. RoboMind can thus be adapted to change the implementation of programming concepts to suit a particular programming language or to add additional functionality.

4.7 Scratch

The purpose of Scratch is to provide children with a tool that will allow them to start programming earlier [33]. Scratch allows people of different backgrounds and interests to easily create their own interactive games, animations, stories and simulations [29, 33].

In Scratch a building block metaphor is used whereby graphical blocks are combined to build scripts (Figure 8). This allows novice programmers to focus on finding problem solutions as syntax errors are eliminated. Scratch is also visually appealing and promotes active learning.

A problem that novice programmers using Scratch may encounter is that it will be difficult for them to move directly to a traditional programming environment after using Scratch. The use of an intermediate software tool to provide a link between the concepts introduced in Scratch and the methods of implementing these concepts in a programming language is suggested [29].



```
when clicked
repeat 100
    move 20 steps
    if touching edge ?
        turn 90 degrees
```

Figure 8. Scratch main screen with code area in the middle.

4.8 Additional Programming Assistance Tools

Several other PATs were also identified by this research study. PlanAni [6], for example, is a tool that uses animation to demonstrate the roles of variables. Alice [7, 15] is a tool similar to Scratch that can be used to create 3D animations and games by dragging object properties and methods to build the program code. In B# [13], users create a program by dragging and dropping icons to create a flowchart. The program can be executed and equivalent Pascal code is generated. jGrasp [17] automatically generates UML class diagrams to allow users to visualise objects, data structures and primitive variables.

5. EVALUATION OF PATS

This section demonstrates how the selection criteria formulated in Section 3 can be used to evaluate programming assistance tools. The programming assistance tools presented in Section 4 are evaluated. Table 2 and Table 3 provide an indication of which selection criteria each of the tools meet.

BlueJ, Greenfoot, BlueJ and Jeliot allow users to write programs in Java code, while B# generates Pascal code from the flowchart created by the user. The remaining tools are not programming language specific. Scratch, Alice and RobotProg can be used to teach any programming languages even though none of the tools explicitly teaches syntax for these languages. All three tools allow users to learn about different programming concepts using a drag-and-drop interface. The statements used are similar to the statements used by most programming languages even though they do not conform to the syntactical rules of any particular language. RoboMind can be adapted to compile code in any programming language.

BlueJ, Greenfoot, jGrasp and Jeliot allow the user to implement all of the programming concepts listed in Table 2. These are Java tools that are able to open and compile any java source files. The remaining tools allow users to implement certain of the programming concepts.

Table 2. Evaluation of programming assistance tools using selection criteria: Programming Concepts.

✓ = tool meets the criteria **● = tool can be adapted to meet the criteria**	**RoboMind**	**BlueJ**	**Greenfoot**	**Scratch**	**RobotProg**	**B#**	**Jeliot**	**Ville**	**PlanAni**	**Alice3D**	**JGrasp**
Programming Language (s=specific, n=non-specific)	n	s	s	n	n	s	s	n	n	n	s
Concepts											
Variables	●	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Input (getting information from the user)	●	✓	✓	✓	✓	✓	✓		✓	✓	✓
Output(displaying information to the user)	●	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
If-statements	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Case (Delphi)/Switch(Java) statements		✓	✓				✓				✓
For-loops	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Repeat loops	●	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
While loops	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
String handling		✓	✓				✓	✓			✓
Procedures	✓	✓	✓	✓	✓		✓	✓		✓	✓
Functions	●	✓	✓				✓				✓
One-Dimensional Arrays		✓	✓	✓			✓	✓	✓		✓
Two-Dimensional Arrays		✓	✓				✓				✓
File handling		✓	✓				✓				✓
Accessing a database		✓	✓				✓				✓
SQL statements		✓	✓				✓				✓
Correct use of parameters		✓	✓				✓	✓			✓
Objects & classes		✓	✓	✓			✓			✓	✓
Problem solving	✓	✓	✓	✓	✓		✓			✓	✓
Debugging	✓	✓	✓				✓	✓			✓

B#, Jeliot, PlanAni, Ville and RobotProg can assist users with code comprehension and code execution through the use of visualisation and animation (Table 3). All of the tools, except Alice and Scratch, can assist users to improve their knowledge of programming language syntax. The statements used by the original RoboMind are similar to Java but the editor can be adapted to compile statements that users are more accustomed to using in a particular programming language. All of the tools have been developed to support user understanding of programming principles and concepts. The tools also all promote self-study and self-exploration by users.

Scratch and Alice help users to create a solution correctly. In Scratch and Alice the statements used indicate to users where conditions or variables must be inserted or if other statements must be included within a loop or control structure. RobotProg allows users to create a solution using a flowchart diagram. Users are able to visualise and compare the execution of the solution using the flowchart with the actions of the robot.

All of the tools except PlanAni allow users to code or create a solution within the tools, thus catering for the kinesthetic learning preference. In PlanAni, users can only run built-in examples to understand how the code is executed. None of the tools cater for the aural learning preference. Jeliot, Ville and PlanAni include functionality to ask users questions regarding the code or provide explanations when the code is executed. This may assist users that prefer the read/write learning preference. All of the tools address the visual learning preference by using visualisation when building the code solution or during code execution.

All of the tools can be used to assist users to apply their understanding of simple exercises to more complex exercises. None of the tools explicitly scaffold the learning. In all the tools, the example exercises provided should include simple as well as complex examples of different programming concepts to assist user understanding.

The error handling item is *greyed* out for Scratch and Alice (Table 3). These tools do not require error handling or error messages to be displayed to users. The use of the drag-and-drop interface ensures that users can only use the correct statements and syntax. Error handling and compiler messages are also not applicable for PlanAni as built-in examples are used which cannot be edited by the user.

Table 3. Evaluation of programming assistance tools using selection criteria: Programming skills and knowledge.

✓ = tool meets the criteria ● = tool can be adapted to meet the criteria	RoboMind	BlueJ	Greenfoot	Scratch	RobotProg	B#	Jeliot	Ville	PlanAni	Alice3D	JGrasp
Programming skills & knowledge:											
Code comprehension					✓	✓	✓	✓	✓		
Promotes problem solving using strategies									✓		
Code execution	✓				✓	✓	✓	✓	✓		
Syntax knowledge	●	✓	✓			✓	✓	✓	✓		✓
Knowledge of programming principles & concepts	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Teaching/Learning approach:											
Constructivist (promote self-study)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Feedback to guide solution creation				✓		✓				✓	
Learning Preferences:											
Visual	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Aural											
Read/Write							✓	✓	✓		
Kinesthetic (user actually codes)	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
Other:											
Simple & Complex examples (scaffolding)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Error handling	✓	✓	✓	▨		✓	✓	✓	▨	▨	✓
Simple error messages/suggestions to correct errors	✓			▨		✓			▨	▨	
Visualisation/Animation	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Can Adapt?	✓					✓		✓	✓		

Only B# and RoboMind use simple error messages that try to inform users of syntax errors in the code using language and terms that are easier for novice programmers to understand. BlueJ, Greenfoot, jGrasp and Jeliot – use the standard Java compiler. The error messages are the same messages that expert programmers would receive in professional programming environments such as Netbeans or Eclipse. All of the tools that have been evaluated are freely available for use and can be downloaded, without charge, from their respective websites. The source-code of RoboMind and B# are available for modification and adaption. Ville can be set up to convert code examples into other programming languages that are not included with the initial installation. In PlanAni, although the examples are built-in, it is possible to extend these examples using a special file format. A programmer with understanding of programming concepts and principles and who is able to work with these different tools will be able to adapt these tools to cater for more programming concepts and/or different programming languages.

6. CONCLUSION

Teaching novice programmers to program requires an understanding of the difficulties of learning to program. This research study has discussed the difficulties novice programmers face when learning to program and highlighted reasons for some of these difficulties. Figure 1 and Section 2 explain the knowledge and skills required to program successfully.

Novice programmers can be assisted by the programming environments or tool in which they learn to program, especially if the programming tool is specifically designed to assist novice programmers. Before selecting a tool, however, it is important to identify what knowledge and skills a novice programmer is trying to develop. Table 2 and Table 3 (Section 5) indicate that programming assistance tools do not meet all the criteria that have been identified to assist novice programmers. It is recommended that skills and knowledge that are most important for the novice to develop (for example, problem solving, understanding of code execution, or syntax knowledge) should first be identified in order to guide the tool selection.

One should also be aware of how difficult it may be to use a tool if no explicit instruction or assistance will be provided before providing or recommending a programming assistance tool to novice programmers. A brief document describing the main interface of the tool together with sources where additional help can be found is recommended. Including simple and complex example exercises with the tool may also assist users to understand how different programming concepts can be implemented.

The selection criteria (Table 1) presented in this paper has been used in a research study to select tools for IT learners in South African secondary schools. The selection criteria will be evaluated and changed, where necessary, based on the findings of the research.

7. ACKNOWLEDGMENTS

We would like to thank the schools, teachers and learners who participated in this research study for their valuable contributions.

8. REFERENCES

[1] Al-Imamy, S. Alizadeh, J. and Nour, M.A. 2006. On the Development of a Programming Teaching Tool: The Effect of Teaching by Templates on the Learning Process. *Journal of Information Technology Education*, Vol. 5, 271-283.

[2] Baldwin, K. and Kuljis, J. 2000. Visualisation Techniques for Learning and Teaching Programming. In *Journal of Computing and Information Technology - CIT 8*, Vol. 4, 285-291.

[3] Bednarik, R. and Tukiainen, M. 2006. An eye-tracking methodology for characterizing program comprehension processes. In *Proceedings of the 2006 symposium on Eye tracking research & applications (ETRA '06)*. ACM, New York, NY, USA, 125-132.

[4] Bednarik, R., Moreno, A. and Myller, N. 2005. Jeliot 3, an Extensible Tool for Program Visualisation. In *Proceedings of the Koli Calling 2005: 5th Annual Finnish / Baltic Sea Conference on Computer Science Education.*

[5] Ben-Ari, M., Levy, R. and Uronen, P.A. 2000. An Extended Experiment with Jeliot 2000. In *Proc. Of the Program Visualisation Workshop,* Porvoo, Finland.

[6] Byckling, P. and Sajaniemi, J. 2006. Roles of Variables and Programming Skills Improvement. *SIGCSE Bulletin*, Vol. 38(1), March 2006, 413-417.

[7] Cooper, S., Dann, W. and Pausch, R. 2003. Teaching Objects-first in Introductory Computer Science. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education (SIGCSE '03)*. ACM, New York, NY, USA, 191-195.

[8] De Raadt, M. 2008. *Teaching Programming strategies explicitly to Novice Programmers*. Doctoral dissertation. University of Southern Queensland.

[9] Department of Education. 2008. *National Curriculum Statement. Grades 10-12 (General)*. Learning Programme Guidelines. Information Technology.

[10] Fleming, N. and Baume, D. 2006. Learning Styles Again: VARKing up the right tree! Educational Developments, SEDA Ltd., Issue 7.4, 4-7, November 2006.

[11] Garner, S. 2007. A program design tool to help novices learn programming. In *ICT: Providing choices for learners and learning. Proceedings Ascilite Singapore 2007*, 321-324.

[12] Gayo-Avello, D. and Fernández-Cuervo, H. 2003. Online Self-Assessment as a Learning Method. In *Proceedings of the 3rd IEEE International Conference on Advanced Learning Technologies*, 2003. Published 9-11 July 2003, 254-255, ISBN: 0-7695-1967-9.

[13] Greyling, J.H., Cilliers, C.B. and Calitz, A.P. 2006. B#: The Development and Assessment of an Iconic Programming Tool for Novice Programmers. In *7th International Conference on Information Technology Based Higher Education and Training (ITHET'06)*, 367-375.

[14] Haataja, A., Suhonen, J., Sutinen, E. and Torvinen, S. 2001. High School Students Learning Computer Science Over the Web. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning*. Wake Forest University. Vol. 3(2), October 2001.

[15] Henriksen, P. and Kölling, M. 2004. Greenfoot: Combining object visualisation with interaction. In *Companion to the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA)*, pp. 73-82, Vancouver, BC, CANADA, November 2004.

[16] Hu, C. 2008. Just Say ‘A Class Defines a Data Type’. In Communications of the ACM, Vol. 51(3), 19-21.

[17] jGRASP 2009. Overview of jGRASP and the Tutorials. DOI= http://www.jgrasp.org/tutorials187/00_Overview.pdf.

[18] Kölling, M. 1999. The Problem of Teaching Object-Oriented Programming, Part 1: Languages. *Journal of Object-Oriented Programming*, Vol. 11(8), 8-15.

[19] Kölling, M. and Rosenberg, J. 2001. Guidelines for Teaching Object Orientation with Java. In *Proceedings of the 6th conference on Information Technology in Computer Science Education (ITiCSE 2001)*, Canterbury, 2001.

[20] Kölling, M. and Rosenberg, J. 2001. BlueJ - The Hitch-Hikers Guide to Object Orientation.

[21] Koorsse, M., Calitz, A.P. and Cilliers, C.B. (2010). Programming in SA Secondary Schools: The Inside Story. SACLA, 2010.

[22] Koorsse, M. Cilliers, C.B. and Calitz, A.P. (2010). Motivation and Learning Preferences of Information Technology Students in South African Secondary Schools. SAICSIT, 2010.

[23] Lahtinen, E., Ala-Mutka, K., and Järvinen, H. 2005. A study of the difficulties of novice programmers. In *Proceedings of the 10th Annual SIGCSE Conference on innovation and Technology in Computer Science Education* (Caparica, Portugal, June 27 - 29, 2005). ITiCSE '05. ACM, New York, NY, 14-18.

[24] Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B., and Thomas, L. 2004. A Multi-National Study of Reading and Tracing Skills in Novice Programmers. In *Working Group Reports from ITiCSE on innovation and Technology in Computer Science Education*, Leeds, United Kingdom, June 28-30, 2004. ITiCSE-WGR '04, 119-150.

[25] Lister, R., Simon, B., Thompson, E., Whalley, J. L., and Prasad, C. 2006. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Bologna, Italy,

June 26 - 28, 2006). ITICSE '06. ACM, New York, NY, 118-122.

[26] Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., and Paterson, J. 2007. A survey of literature on the teaching of introductory programming. In *Working Group Reports on ITiCSE on innovation and Technology in Computer Science Education* (Dundee, Scotland, December 01 - 01, 2007). J. Carter and J. Amillo, Eds. ITiCSE-WGR '07, 204-223.

[27] Pendergast, M.O. 2005. Teaching Introductory Programming to IS Students: Java Problems and Pitfalls. In *Journal of Information Technology Education*, Vol. 5, 491-515.

[28] Rajala, T., Laakso, M., Kaila, E. and Salakoski, T. 2007. VILLE – A Language-Independent Program Visualisation Tool. *Seventh Baltic Sea Conference on Computing Education Research (Koli Calling 2007),* Finland.

[29] Resnick, M., Malone, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. and Kafai, Y. 2009. Scratch: Programming for All. In *Communications of the ACM*, Vol. 52(11), 60-67.

[30] Robins, A., Rountree, J. and Rountree, N. 2003. Learning and Teaching Programming: A Review and Discussion. In *Computer Science Education*, Vol. 13(2), 137–172.

[31] Rongas, T., Kaarna, A. and Kalviainen, H. 2004. Advanced Learning Technologies. In *Proceedings of IEEE International Conference on Advanced Learning Technologies, ICALT'04,* 678–680.

[32] Shuhidan, S., Hamilton, M. and D'Souza, D. 2009. A Taxonomic Study of Novice Programming Summative Assessment. In *Eleventh Australasian Computing Education Conference (ACE2009),* (Wellington, New Zealand, January 2009). Conferences in Research and Practice in Information Technology (CRPIT), Vol. 95. M. Hamilton & T. Clear, Eds.

[33] Utting, I., Cooper, S., Kölling, M., Maloney, J. and Resnick, M. 2010. Alice, Greenfoot, and Scratch - A Discussion. *ACM Transactions on Computing Education*, Vol. 10(4), Article 17, Pub. Date: November 2010.

[34] Vickers, P. 2009. How to Think Like a Programmer. Cengage Learning EMEA. ISBN: 978-1-84480-903-5.