



INSTRUCTIONAL TECHNOLOGY FOR THE TEACHING OF NOVICE PROGRAMMERS AT A UNIVERSITY OF TECHNOLOGY

**Dissertation submitted in fulfilment
of the requirements for the degree of
Master of Information Technology**

RESEARCHER

**Mr. Godfrey Rudolph
185026494**

Department of Information Technology
Faculty of Informatics and Design
Cape Peninsula University of Technology

**Supervisor:
Professor Retha de la Harpe**

**Co-Supervisor:
Dr Ernest Pineteh**

Department of Information Technology
Faculty of Informatics and Design
Cape Peninsula University of Technology

DECLARATION

I declare that this dissertation is a presentation of my original research work and that it has not been submitted for a similar degree at any other university. The research was conducted under the supervision of Professor Retha de la Harpe and Dr Ernest Pineteh at the Cape Peninsula University of Technology, Cape Town campus. It is submitted in fulfilment of the requirements for the degree of Master in Information Technology (MTech IT) in the Department of Information Technology, Faculty of Informatics and Design.

GODFREY RUDOLPH

June 2015

ACKNOWLEDGEMENTS

This dissertation is the result of the collective efforts of many people who, knowingly or unknowingly, had an influence on my life and contributed to the completion of my dissertation.

I would like to express my sincere and immense gratitude to:

- My late father and late mother, Frederick and Martha for always believing in me and supporting me from the inception of my studies.
- My two children Kendyl & Dean, and Lynette for making me believe that I have the ability to soar from strength to strength. Without your unwavering support and continuous words of encouragement, I doubt I would have been able to achieve what I did.
- My biological brothers and sisters, and their respective families for their encouragement and support.
- My promoters, Professor Retha de la Harpe, Dr Ernest Pineteh and research mentors Professor Bennet Alexander & Izak van Zyl, for their invaluable guidance, immense encouragement, dedication and support.
- My spiritual leader, Priest Welkom, spiritual sister Rochelle, colleagues and friends, who are too numerous to mention, for your prayers, encouragement, assistance, understanding and patience.
- My colleague and friend, Alvino Moses and mentor Moegamat Alexander, for the invaluable insight into this dissertation.

ABSTRACT

Learning computer programming can be fun, challenging and improve problem solving which is a useful ability in general. A teaching-learning environment with a strong emphasis on problem solving promotes social behaviour and discloses the personal benefits that individuals working in almost any Information Technology position can get from programming knowledge. This research project is looking at the challenges experienced by novice programmers and the negative effect it has on the student and the university. This study will address the knowledge and skills needs of programming students and the challenges for students and educators to evolve from traditional to technology-supported teaching and learning.

Computer programming is a cognitively challenging subject and good instructional strategies are important in providing the student with optimal learner support. Novice programmers often struggle to understand how a computer executes a program, which impacts negatively on the delivery of the subject and throughput rates. The majority of first year Information Technology students at Cape Peninsula University of Technology are novice programmers and lack strong logic and reasoning as well as other Information Technology skills that can facilitate their interpretation and application of key concepts in programming.

These challenges and negative impact on the academic development of programming students have therefore forced the researcher to investigate innovative teaching strategies and/or instructional technologies that can facilitate novice programmers in learning the basic programming concepts. The purpose of this on-going study is to enhance the traditional method of teaching and the understanding of the problems experienced by novice programmers. This study attempts to respond to the question of what the tentative design principles of instructional technology are that can be used to facilitate novice programmers' understanding of programming concepts.

A mix methodology was considered but at the end a qualitative approach was employed. Multiple sources of data gathering, which include participant observations,

video recording, a questionnaire, and document analysis, were used as research instruments.

The findings, relative to providing a basis for finding a mechanism to help our first year students to cope with the abstract concepts of programming, reflected the literature review. Other key findings included:

- Students have little or no prior computer or programming experience
- Student population is diverse in terms of computer skills and programming knowledge
- Visualization will help reduce the difficulties in writing programs

The overall outcomes of this study suggest that:

- Good programming examples that include games should be used
- Students must be given the opportunity to be more active in their learning.
- Computerized assistants should be provided for novice programmers
- A visualization tool similar to Scratch should be considered
- A basic background in Mathematics is recommended

Table of Contents

DECLARATION	2
ACKNOWLEDGEMENTS	3
ABSTRACT	4
LIST OF FIGURES.....	11
LIST OF TABLES.....	12
DEFINITION OF TERMS	13
LIST OF ABBREVIATIONS	14
CHAPTER 1 INTRODUCTION	15
1.1 INTRODUCTION AND BACKGROUND	15
1.2 PROBLEM STATEMENT	16
1.3 RESEARCH QUESTIONS	16
1.3.1 SUB-QUESTIONS	17
1.4 AIMS AND OBJECTIVES.....	17
1.5 DELINEATION OF THE RESEARCH	17
1.6 RESEARCH APPROACH	18
1.7 OUTCOMES AND OUTPUT.....	18
1.8 SIGNIFICANCE OF STUDY	19
1.9 ETHICAL CONSIDERATIONS.....	19
CHAPTER 2 LITERATURE REVIEW	21
2.1 INTRODUCTION	21
2.2 LEARNING AND TEACHING PROGRAMMING	22
2.2.1 TEACHING STRATEGIES.....	23
2.2.2 RESOURCES AND TOOLS TO AID WITH LEARNING AND TEACHING.....	24
2.2.2.1 PROGRAM VISUALIZATION.....	28
2.2.2.2 PROGRAM / ALGORITHM ANIMATIONS.....	33

2.2.2.3 INTERACTIVE ANIMATIONS AND GAMES PROGRAMMING	34
2.3 DIFFICULTIES OF LEARNING PROGRAMMING	37
2.4 FIRST PROGRAMMING LANGUAGE	41
2.5 NOVICES VS EXPERT PROGRAMMERS	42
2.5.1 ATTRIBUTES OF A GOOD PROGRAMMER	42
2.5.2 CHARACTERISTICS OF NOVICE PROGRAMMERS.....	43
2.6 HOW NOVICES LEARN TO PROGRAM	45
2.7 SCAFFOLDING	49
CHAPTER 3 RESEARCH METHODOLOGY	52
3.1 INTRODUCTION	52
3.2 RESEARCH APPROACH	52
3.2.1 DESIGN RESEARCH	55
3.2.2 RESEARCH SETTING	55
3.2.3 RESEARCH METHODS	56
3.3 RESEARCH METHODOLOGY FOR THIS STUDY.....	56
3.4 SELECTION CRITERIA AND PROCESS	60
3.4.1 PROFILING	61
3.5 RESEARCH INSTRUMENTS FOR DATA COLLECTION.....	62
3.5.1 WORKSHOPS	62
3.5.2 PARTICIPANT OBSERVATION	63
3.5.3 VIDEO	64
3.5.4 QUESTIONNAIRE	64
3.5.4.1 QUESTIONNAIRE ITEMS.....	65
3.6 DATA ANALYSIS.....	65
3.7 INTERPRETATION	67

3.8 ROLE OF THE RESEARCHER	68
3.9 CHAPTER SUMMARY	68
CHAPTER 4 DATA ANALYSIS AND FINDINGS	69
4.1 INTRODUCTION	69
4.2 BACKGROUND	69
4.3 WORKSHOPS DESIGN	71
4.3.1 WORKSHOP OBJECTIVES	72
4.3.2 AGENDA.....	72
4.3.3 PARTICIPATION PROCESS	72
4.4 DEMOGRAPHICS OF PARTICIPANTS	74
4.5 ACADEMIC PERFORMANCE.....	75
4.6 WORKSHOPS	78
4.6.1 WORKSHOP 1	78
4.6.1.1 TARGET GROUP	78
4.6.1.2 RESPONSES	78
4.6.1.3 KEY THEMES	83
4.6.2 WORKSHOP 2	84
4.6.2.1 TARGET GROUP	84
4.6.2.2 RESPONSES	85
4.6.2.3 KEY THEMES	89
4.6.3 WORKSHOP 3	90
4.6.3.1 TARGET GROUP	90
4.6.3.2 RESPONSES	91
4.6.3.3 KEY THEMES	94
4.7 CHAPTER SUMMARY	95

CHAPTER 5 DISCUSSIONS OF FINDINGS.....	96
5.1 INTRODUCTION	96
5.2 COMPARISON OF PARTICIPANTS RESPONSES	96
5.2.1 LEARNING OF PROGRAMMING CONCEPTS	98
5.2.2 VISIBILITY, NAVIGATION AND INTERFACE.....	100
5.2.3 ELIMINATION OF COMPILATION STEP	100
5.2.4 HELP SCREENS AND DEFAULT PARAMETERS	101
5.2.5 VISUAL FEEDBACK.....	101
5.2.6 RUNNING OF INCORRECT PROGRAMS	102
5.2.7 VARIABLES AS CONCRETE OBJECTS.....	102
5.2.8 IDEAS TO HELP FIRST YEARS	102
5.2.9 SCRATCH AS AN INSTRUCTIONAL TOOL.....	104
5.3. PROBLEM SOLVING SKILLS.....	104
5.4 MOTIVATIONAL FACTORS.....	105
5.5 ATTITUDE	105
5.6 DIVERSE STUDENT BACKGROUNDS	107
5.7 PRIOR KNOWLEDGE	108
5.8 TEACHING APPROACH.....	109
5.9 KEY FINDINGS FROM THE STUDY	109
5.10 CHAPTER SUMMARY	114
CHAPTER 6 CONCLUSIONS AND RECOMMENDATIONS.....	115
6.1 INTRODUCTION	115
6.2 SUMMARY OF CHAPTERS.....	115
6.3 SUMMARY OF FINDINGS.....	117
6.4 RESEARCHED QUESTIONS REVISED	118

6.5 VALIDITY OF RESEARCH	119
6.6 RECOMMENDATIONS	120
6.7 FURTHER RESEARCH.....	122
6.8 CONCLUSION	123
REFERENCES	125
BIBLIOGRAPHY OF ADDITIONAL REFERENCES.....	134
APPENDICES.....	139
Appendix A: Invitation Letter to Participate	139
Appendix B: Clock Project Worksheet	140
Appendix C: Questionnaire.....	147

LIST OF FIGURES

- Figure 2.1 Conceptual framework of concepts
- Figure 3.1 Design research methodology
- Figure 3.2 Framework of the research study
- Figure 3.3 Structure of a thematic network
- Figure 4.1 Student enrolments by population at CPUT (2012)
- Figure 4.2 Ethnic profile of participants
- Figure 4.3 Gender profile of participants
- Figure 4.4 Screenshots of Scratch user-interface
- Figure 5.1 Screenshot of Visual Studio integrated development environment
- Figure 5.2 Screenshot of a C++ runtime environment

LIST OF TABLES

Table 3.1	Four stages of case study methodology
Table 3.2	Admission rating for matriculants (2008) onwards
Table 3.3	Admission rating for matriculants before (2008)
Table 4.1	First year performance and acceptance
Table 4.2	Summary of Workshop 1 results
Table 4.3	Summary of Workshop 2 results
Table 4.4	Summary of Workshop 3 results
Table 5.1	Comparison of participant responses
Table 6.1	Summary of workshops findings
Table 6.2	Research questions revisited

DEFINITION OF TERMS

Novice – a beginner; one who is not very familiar or experienced in a particular subject

Programmer - a person who designs, writes and tests computer programs

Instructional technology - the process of using technologies such as multimedia, computers, audio-visual aids, interactive media and teleconferencing as tools to improve teaching and learning.

Web 2.0 a second generation of World Wide Web which focuses on the ability for people to collaborate and share information online.

E-learning –the network-enabled transfer of skills and knowledge

Virtual –it distinguishes something that is merely conceptual from something that has physical reality.

Constructivism –a psychological theory of knowledge which argues that humans generate knowledge and meaning from their experiences.

Mobile device - a pocket-sized technological device tool such as cell phone, laptop and palmtop

Mobile learning - Any sort of learning that happens when the learner is not at a fixed, predetermined location by making use of mobile devices.

Social software – a type of software which allows people to communicate and collaborate while using the application

Social presence – the ability of participants to project themselves socially and emotionally, through a medium of communication.

Podcasting –Podcasting allows subscribers to use a set of feeds to view syndicated Web content. With podcasting however, you have a set of subscriptions that are checked regularly for updates and instead of reading the feeds on your computer screen, you listen to the new content on your iPod or any other audio device. It is similar to RSS

LIST OF ABBREVIATIONS

CPUT Cape Peninsula University of Technology

UKZN University of KwaZulu-Natal

DS1 Development Software 1

DIT Department of Information Technology

FID Faculty of Informatics and Design

SA South Africa (n)

SMS Short Message Services

HE Higher Education

UNISA University of South Africa

LMS Learner Management System

NDIPIT National Diploma Information Technology

DoE Department of Education

UoT University of Technology

DS2 Development Software 2

SDLC Software Development Life Cycle

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION AND BACKGROUND

Computer programming is a cognitively challenging subject and good instructional strategies are important in providing the student with optimal learner support (Chan Mow, 2008). Novice programmers often struggle to understand how a computer executes a program (Guo, 2006). This impacts negatively on the delivery of the subject and on throughput rates because lecturers spend valuable time dealing with final year students who are not able to undertake some of the simplest programming tasks (Jenkins and Davy, 2000).

Research shows that only 38% of first year computer programming students can write a simple program for calculating the average of a set of numbers. Interestingly, this is one of the tasks that a student with one semester of programming knowledge is expected to handle adequately (Clear et al., 2008). Students' inability to handle some of the basic programming tasks confirms recurring claims that programming courses are difficult, and often have the highest dropout rates. Moreover, the high dropout rate reinforces the perception that, it takes ten years of experience to develop a novice into an expert programmer (Guo, 2006). These challenges have forced programming lecturers to innovate teaching strategies and/or instructional technologies that can facilitate increased understanding of programming concepts.

The Cape Peninsula University of Technology (CPUT) currently uses Learner Management System (LMS) software also known as Blackboard to create and host courses including programming on the Internet. Courses supported by this software can be operated independently as online courses or as supplementary to traditional classroom courses. In general this system has impacted positively on teaching and learning, motivating the institution to offer an effective solution beyond traditional course management (Reeves et al., 2002). To this end, face-to-face interaction combined with computer mediated instruction can provide realistic practical opportunities for students

and teachers to make learning independent, useful and ever growing. The above examples illustrate how digital technology can enhance teaching and learning (Patalong, 2003) as well as the impact of newer instructional technology solutions such as social media on the delivery of Information Technology (IT) courses at CPUT.

Although the first year programming subject Development Software 1 (DS1) is also hosted on Blackboard, it is not clear whether the software has enhanced the delivery of key programming concepts to novice programmers at CPUT. This is perhaps because the technology does not cater for the teaching and learning needs of individual subjects. To address this concern, exploring alternative strategies for teaching novice programmers at CPUT need to be considered.

CPUT has a strong focus on career education to prepare students for the proper application of skills in the workplace. The Department of Information Technology (DIT) is one of the academic departments in the Faculty Informatics and Design (FID) at CPUT. The DIT offers certificates, diplomas and Bachelor, Masters and Doctor of Technology Degrees in IT according to the needs of the respective student population. The IT course prepares students for professional careers opportunities in Business Applications, Communications Networks, Software Development and Multimedia Technology.

1.2 PROBLEM STATEMENT

The majority of first year IT students at CPUT are novice programmers and find it difficult to interpret and apply the key programming concepts.

1.3 RESEARCH QUESTIONS

- What are the tentative design principles of appropriate instructional technology that can be used to facilitate novice programmers' understanding of programming concepts?

1.3.1 SUB-QUESTIONS

- What are the challenges of technology-supported teaching of programming concepts?
- What are the contextual aspects that need to be considered for the design of an instructional technology intervention?
- How do novice programmers interact with instructional technologies during the programming process?
- How can instructional technology be used to engage students in learning and acquisition of programming skills?

1.4 AIMS AND OBJECTIVES

The aim of this study is to explore the aspects that influence the novice programmers' ability to master the programming concepts. The following objectives are aligned to the research problem and questions:

- To assess the skills and knowledge that a novice programmer requires in order to master programming concepts.
- To investigate how instructional technology can be used to address the skills gap of students learning programming concepts.
- To identify existing technology solutions and how they have contributed to the teaching of programming.
- To identify other possible technology solutions that can be used to teach programming concepts to novice programmers.

1.5 DELINEATION OF THE RESEARCH

The result of the research was aimed at a focused group as opposed to generalization. The primary group was the first year DS1 students of CPUT. This study does not consider the educational aspects of teaching programming concepts and the use of instructional technology per se. The focus is only on the role of instructional technology as used by the lecturer as a teaching strategy to explain the programming concepts to the students. Mobility of students and the use of instructional technology outside the classroom is also not considered in this study.

1.6 RESEARCH APPROACH

The majority of the first year IT students at CPUT struggle to interpret and apply the basic abstract concepts of programming. According to Plomp (2009), design research is the study of educational interventions that will show the way to solutions for complex problems in educational practice in a real-world setting. The researcher therefore found it appropriate to engage in an educational design research methodology to consider the development of an intervention and to explore alternative methods to teach programming at CPUT. It was decided to follow an exploratory towards an intervention design research approach that will provide insights and contributions to enhance the teaching and learning of abstract programming concepts to novice programmers at CPUT.

1.7 OUTCOMES AND OUTPUT

Design research has two outcomes: design principles which represent the knowledge that was generated during the design process, and the different designs or models produced during the design process. In this study the principles that need to be considered for the design of an appropriate instructional technology tool as an intervention will be produced in the form of recommendations. The outcome of this study will be improved understanding of the challenges experienced by novice programmers when learning programming concepts, and how they respond to an example instructional technology.

The output of this study is:

- guidelines for the use of technology to enhance teaching and learning
- findings that suggest how students interact with technology as part of their learning process
- research papers and a thesis.

1.8 SIGNIFICANCE OF STUDY

The outcome provides a basis for recommendations to support the development of learning materials and approaches for basic programming courses. The research project also has the potential to complement the ongoing vision of CPUT in preparing students to be critical thinkers and knowledgeable professionals through technology education and innovation.

The use of technology in teaching and learning will enhance group collaboration for both students and instructors. Mobile learning will bring new technology into the classroom. It will be a useful add-on tool for students regardless of their different learning style capabilities. Students taking advantage of the learning opportunities offered by mobile devices will effectively support the learning process rather than just being integral to it (Savill-Smith et al., 2006).

1.9 ETHICAL CONSIDERATIONS

Ethics clearance to conduct the study was obtained from the Ethics Committee of CPUT. Permission to conduct the study was also obtained from students who were directly involved in the study. Both verbal and written (Appendix A) consent was obtained prior to the study. The purpose of the study was explained and the expected roles of the participants prior to the research were clarified. To comply with internationally accepted ethical standards, no names of individuals were recorded on research instruments. No individual was linked to a particular completed research instrument, ensuring anonymity. No compensation was paid to any respondent for participation in the study. No respondent or participant was harmed physically, emotionally or otherwise. As with other studies, quality assurance was done with respect to:

- The correctness and completeness of open ended questions;
- All participants understanding the nature and consequences of their participation in the study;
- The quality of data capturing done by encoders; and
- Placing all results in the public domain as soon as available.

The following chapter focusses on the literature review concerning the challenges experienced by novice programmers and the knowledge and skills needs of programming students.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION

Programmers must have comprehensive problem solving skills and be able to think of a suitable solution when presented with a problem. Technologies are available to aid novice programmers in learning to program. This research is an on-going project looking at the challenges experienced by novice programmers and the negative effect it has on the student and the university. This study addresses the knowledge and skills needs of programming students and the challenges for students and educators to evolve from traditional to technology-supported teaching and learning.

This chapter reviews literature relative to skills and knowledge required to master programming concepts and identify existing and other technology solutions to contribute to the teaching of students. The emphasis is on novice programmers that are beginning their programming studies at CPUT. The researcher wants to find out what has been done in his field of study by reviewing existing scholarships and how other scholars have investigated the research problem that the researcher is interested in. The researcher also wants to learn from other scholars: how they have theorized and conceptualized on the issues, what they have found empirically and the instruments they have used and to what effect.

The literature was reviewed to establish the current status of issues relevant to novice programmers' ability to master programming. The outcome of the literature analysis is presented as follows: firstly, the aspects of learning and teaching programming; followed by the identification of difficulties experienced by novice programmer and specifically during the use of their first programming language. This is followed with a discussion about how novice programmers learn programming and the difference between novice and expert programmers. The literature analysis is concluded with suggestions for scaffolding to support the teaching and learning process of programming languages to novice programmers. A proposed conceptual framework

based on the literature reviewed is proposed at the end of the chapter to indicate the concepts and relationships considered in this study.

2.2 LEARNING AND TEACHING PROGRAMMING

Universities offering IT degrees are expected to teach programming languages that are close to those used in the industry. It is therefore rarely possible to select an approach for an introductory programming course based only on pedagogical reasons (Ala-Mutka, 2004). Learning to program involves constructing knowledge in the following four phases of the Software Development Life Cycle (SDLC):

- Phase 1: Analyze the problem
- Phase 2: Design and develop a solution algorithm
- Phase 3: Implement the algorithm
- Phase 4: Test and revise the algorithm

Analyzing the problem means to read and understand what is required. It helps with the initial analyses. In the second phase the problem is broken down into smaller tasks in order to establish a solution outline. The solution outline is then expanded into an algorithm which consists of a set of precise steps that will describe the tasks to be performed. The fourth phase is one of the most important steps in software development, and is often ignored. The objective of this step, as described by Li (1990), is to identify major errors in software and to see if this software does what it is supposed to do and also whether it malfunctions.

All of the software development lifecycle phases need to be mastered by novice programmers and the activities of each phase can be considered as learning activities (Garner, 2003). Students must learn to systematically attack problems before translating them into a working computer program. Three basic problems have been identified when using a text-based language to teach introductory programming (Greyling et al., 2006). Firstly, extensive focus is placed on syntax, forcing students to first get the syntax correct. This leads to the misconception that programming is about writing syntactically correct code. Secondly, limited emphasis is placed on problem solving,

leaving little or no time to focus on other phases of the SDLC. Thirdly, the lack of support for experiencing program execution, resulting in students writing programs that contain correct statements but in the incorrect logical order (Greyling et al., 2006).

2.2.1 TEACHING STRATEGIES

Programming is defined by Saeli et al. (2011) as “a process of writing, testing, troubleshooting, and maintaining the source of code of computer programs”. According to the authors, the level of programming understanding is low even after two years of tuition, due to the difficulty of learning programming skills. However, students will be able to master the learning of programming skills with the support of suitable teaching strategies and tools. Ala-Mutka (2004) states that learning programming is a complex task and advised that teachers should design their teaching approaches carefully. In the traditional approach, teaching the concepts first, is common and is found to be effective. However, different other approaches may also be followed.

The biggest problem with novice programmers is that they do not seem to understand the basic concepts and learning to apply them. Therefore Robins et al. (2003) suggest that teachers focus more on the combination and use of features especially underlying issues of basic program design. For example, building visualized examples of the basic structure and their combinations in different situations to promote students’ understanding of the different programming strategies and help build a mental “library” of different solution schemas for program design.

In this study, the researcher would like to investigate how, by applying different teaching strategies in class, lecturers will meet the learning needs of individual students and recognize the fact that students have different abilities to grasp the programming concepts. Using a lecture method can result in the teachers being active and students being passive. However, students may be kept attentive in class by making use of quizzes. Achieving instructional objectives depends on the teaching method applied. With the aid of visualization, content and concepts can be represented structurally.

According to Greyling et al. (2006), the higher incidence of under-prepared students at South African education institutions, has an actual impact on introductory courses which rely on the use of technological tools as a pedagogical concern. They further argue that novice programmers experience difficulties in problem solving, the use of traditional programming environments and misconceptions concerning programming language constructs. Students are introduced to programming by starting with simple programming statements. This “bottom-up” approach does not give students the opportunity to develop the essential problem-solving skills that are highly important for effective program development. It is therefore suggested that the programming language must be seen as the tool used to implement the solution to a problem. Students should be introduced to the “top-down” approach where the logical processes involved should first be identified in developing a solution.

The above points are relevant to this study and indicate that different students have different learning styles. Hence, educators need to focus on the individual student and be attentive of both their knowledge and preferred learning style. To overcome the barriers of learning to program, good programming examples, including games, should be used for practicing with the aim of enhancing problem solving abilities. Effective support by educators can help restore the confidence of students in their abilities to program. Scaffolding as a teaching method is suggested to allow students to complete a given task whilst providing feedback and assessment. Scaffolding, to support the teaching of programming concepts, will be discussed in Sub section 2.7 in this chapter.

2.2.2 RESOURCES AND TOOLS TO AID WITH LEARNING AND TEACHING

Teaching in an interactive way is likely to motivate students’ participation in classes and the use of dynamic means, showing step-by-step changes in both computer memory and in objects’ state, will benefit students in creating a clearer mental model of program execution (Guo, 2006). The proposed research hopes to explore the introduction of other instructional technologies in the classroom and how it can drive this quality of interactivity at the first year level at CPUT.

According to Van den Berg and Aucamp (2007), the first main digital initiative implemented in South Africa was the MobilED project, an international research and innovation partnership that aims to develop a teaching model in mobile technology to enhance diverse teaching and learning methods. The first phase of this project was piloted at one private and one public school in the Pretoria area in South Africa (SA) using only voice and Short Message Service (SMS) technology. The two schools selected for the pilot project were Cornwall Hill College and Irene Middle School. Cornwall Hill College is a private school and Irene Middle School is a public school comprised mainly of children from previously disadvantaged communities. The three-year pilot project was funded by the government Department of Science and Technology, and Nokia contributed the mobile phones.

The MobilED project was supported by evidence based research outputs to develop sustainable models, processes and practice (Van den Berg and Aucamp, 2007). The project amplified the design of a teaching and learning environment enhanced with mobile technology services. Mobile technologies that support audio playback can therefore be used inside and outside the classroom context to provide user generated audio annotations and information on topics. Since this project was only piloted in schools and not at tertiary level, SA universities such as CPUT can draw on the lessons of the pilot project and introduce a similar model for teaching and learning but at the same time consider the level of study.

Furthermore, Higher Education (HE) promotes lifelong learning and the development of learner skills, which can stimulate creativity and innovation. The development of courses suitable for delivery via the internet, are increasing rapidly at SA schools and universities (Jung, 2005: 94-101). The University of South Africa (UNISA) is one of the ten largest distance education institutions in the world (Wolff, 2002). The university has been the forerunner of digital teaching and learning environments. It has infrastructure with the capacity to integrate online, distance and face-to-face learning. However, it is still unclear how its programming students have benefited from these initiatives

especially in terms of mastering the concepts required by the programming subjects (University of South Africa, n.d.).

Also, the University of KwaZulu-Natal (UKZN) has introduced podcasting as an educational tool on its Westville campus. Lectures podcasting on audio and video files can now be published and accessed by students on their PC's via the internet and their mobile devices. This significant R1 million development for both staff and students was realized through a partnership between UKZN Innovation, UKZN's School of Information Systems (IS&T), the University Research Office and the Core group (University of Kwazulu Natal, n.d.). Again, this model is holistic in that it focuses on the pedagogical concerns of the university rather than the needs of a specific subject like Programming. This research hopes to draw on this landmark projects to conceptualise and test instructional technology which addresses the specific needs of novice programmers at CPUT.

The use of the internet as a tool to aid teaching and learning, allows for dialogue and "near instant" feedback. Despite the challenges faced by instructors on the use of web-based teaching and learning tools, the internet adds value to student learning (Wu, 2005). Greyling and Wentzel (2007) agree that technology assisted learning is not only about conveying content; it is also a medium for promoting social relationships. Social presence forms the foundation for teaching and learning and serves as a building block for a successful learning environment. However, it is difficult to create social presence in large classes.

Available educational technology can serve as an instrument of control to enhance social dimension of online learning (Greyling and Wentzel, 2007). They suggest that students intimidated by face-to-face teaching environments, can be empowered in a virtual world. Social presence is developed through virtual participation of students and by participating in on-line discussions students are more motivated and positive towards their learning. According to Greyling and Wentzel (2007), students become part of growing social relationships enhanced by social presence. This means technology can

be used to encourage online learning and influencing student satisfaction by facilitating online social presence without replacing the lecturer. They further concluded that socialization is an important tool to drive peer learning and an instructional technology for the teaching and learning programming concepts can stimulate interactivity between novice programmers.

The evolving digital age brings about changes in the way instructors structure course layouts. Educators need to give students the opportunity to be more active in their learning. Students should be encouraged to collaborate with other students in forming what they called learning communities. The physical venue, where teaching and learning take place, will become less important. Teachers and students will not be bound to their classrooms. With technology, learning can be transformed into a lifelong experience. Students become actively involved in creating effective learning spaces. They have the opportunity to collaborate and share ideas and thoughts on common issues (Kluge and Riley, 2008).

One of the objectives of virtual learning environments, according to Piccoli et al. (2001), is to promote the interaction between students and teachers. In virtual learning spaces, the internet overcomes the issue of distance to provide limitless knowledge and information. Long distance learning is characterised by the mode of education delivery regardless of the physical classroom. Welsh et al. (2003) define e-learning as the use of computer network technology over an intranet or through the internet to deliver information and instruction. In other words it is a modality of education where the teacher is established to the physical and temporary distance and is mediated by technology. Learning Management Systems (LMS) support communication, interaction and availability of content (Reeves et al., 2002). Students and teachers are allowed to communicate through forums that are available on the internet or intranet. Communication through these virtual spaces promotes a spirit of loyalty amongst students (Dos Reis & Martins, 2008).

As indicated above, social interactivity and peer learning are essential for the development of students' logic and critical skills and this researcher hopes to use this body of knowledge as the conceptual framework of this project. One way to assist novice programmers is to provide computerized assistants created specifically for them. One class assistant is software visualization tools. Software visualization uses interactive computer graphics, graphic design and animations to enhance the interface between computer programmers and their programs (Price et al., 1983). Several low-level program visualization tools have been designed explicitly for novice programmers and have enthusiastically been embraced by students. However, there is little empirical evidence regarding their efficacy (Smith and Webb, 2000). This is confirmed by Sorva et al. (2013) who found that program visualisation systems for beginners are often short-lived research prototypes but that research to date is inclusive regarding the use of visualisation tools to the learner engagement of novice programmers.

Innovative technologies can be integrated into the classroom to enhance teaching and learning. Instructors will be able to take advantage of technology to enrich the educational experience of their students. The basic concepts that underline all programming languages include sequence, iteration and selection.

2.2.2.1 PROGRAM VISUALIZATION

Visualizations have been used for a long time in computer science education because they could contribute towards to improved understanding and be useful for learning abstract and complex concepts in this field. A group working on improving the Educational Impact of Algorithm Visualization studied the use of visualizations in computer science education with three surveys and an extensive literature review and the results of the work have been published in Naps et al. (2002). According to the surveys, all respondents were confident that visualizations help students' understanding and learning of concepts (Ala-Mutka, 2004). For respondents, basic programming concepts include program structure, flow control, conditional testing, variables, expressions and looping constructs.

The National Center for Women and Information Technology study (NCWIT), according to Utting et al. (2013), discovered that Scratch, an interactive animation and games programming setting, “uses hands-on, active learning; it is visually appealing; it allows users to express their own creativity and to build on their own experiences; it gives immediate, understandable feedback; and it allows users to avoid syntax errors without focusing on minutiae, freeing them to focus on processes and concepts”. Similarly to Scratch, Alice, an innovative 3D programming environment, is very visual and students can do directly map instructions to the result to what they see onscreen. Utting et al. (2013) further acknowledge the importance of program visualization and the need for students to be active as the author, rather than a passive observer of the instructor’s animation.

According to Lattu et al. (2000), visualization offers more transparency to the program and their execution, compared to traditional programming environment. Students are encouraged to approach debugging in a more analytic way. In a learning situation, visualization makes it possible to grasp algorithms without understanding the actual code. Lattu et al. (2000) further state that a fundamental cognitive structure is created for students through visualization, in the orienting phase of the learning.

Tekdal (2013) discovered that students have difficulty in understanding what is happening in memory and what a computer is actually doing during the execution of a program. The author claims that program visualization and animation tools help students to better understand program codes, presenting the execution of program lines using graphical effects.

Ala-Mutka (2004), states that visual learning helps students to gain a clearer understanding of the programming principles. It effectively provides students with an insight into the computer system. Effective visualization could be achieved by using animated PowerPoint slides which include dynamically displayed pictures, graphical diagrams, and line-by-line explanations of what is happening in memory at each stage in the execution of a program fragment. According to Guo (2006), lecturers using these

clearly presented and easy to follow slides, will engage students in active thinking which is often driven by questions from instructors.

According to Smith and Webb (2000), program visualization tools are programming language dependent. They are used to animate low-level features of a program for example source code and change of variable states. These tools should be able to automatically handle all possible programs that can be written in the target language. Smith and Webb (2000) further state that program visualization tools might be a promising aid for novice programmers learning to program. Students benefit by having a greater understanding of the programming concepts and to assimilate the new concepts. The use of a visual programming language will provide students with a visual vocabulary which they can grasp and apply instantaneously (Goosen et al., 2007).

The Tampere University of Technology, Finland, is participating in the international Codewitz project, which is a study for developing visualizations (Ala-Mutka, 2004). The goal of the project is to study computer-aided teaching of programming with the aid of illustration, animation and visualization. According to Ala-Mutka (2004), visualization has been used for a long time in computer science education and is considered beneficial in the understanding and learning of abstract and complex concepts. It is further stated that most approaches concentrate on algorithm animation with less emphasis on visualizing the basic structure of the program and execution.

Naps et al. (2002) state that “visualization technologies, no matter how well it is designed, are of little educational value, unless it engages learners in an active learning activity”. Ala-Mutka (2004) suggests that teachers must always remember that technical tools and visualizations are just learning aids and materials. Teachers need to still thoroughly design an instructional approach to issues on the course and how aiding materials can be incorporated into the education.

Linden and Lederman (2011) identified two main approaches to visualization in support of teaching:

- algorithm animation which is largely programming language independent and its purpose is to show how an algorithm works
- program visualization where the emphasis is on animation source code execution showing changes in the variable states, which is programming language dependent.

Next, a few examples of visualization programs are discussed.

BRADMAN

A “Glass Box Interpreter” called Bradman was used in an experiment conducted by Smith and Web (2000) to test the efficacy of tool in assisting novice programmers learn programming concepts. Tekdal (2013) describes it as a low-level Program Visualization tool. Bradman was developed to assist novice programmers in learning the C programming language. The interpreter makes visible the aspects of the programming process that is normally hidden from the user. The motivating factor for introducing Bradman was to produce an assistant that will provide a useful conceptual model for the students when they assimilate new knowledge about programming. The experiment attempted to evaluate whether access to the “Glass Box Interpreter” will assist the user in developing a better understanding of the execution of a program. According to Tekdal (2013), the results of the experiment by Smith and Web (2000) indicate that students who studied with this tool have a better understanding of some programming concepts than those with no access to it. Bradman was enthusiastically accepted by the students. The result was that students seem to know more about how programs work.

JELIOT3

Moreno et al. (2004) present a program visualization tool, Jeliot3 that was designed and programmed at the University of Joensuu, Finland, to aid novice students learning procedural and object oriented programming. The key feature of Jeliot3 is the fully or semi-automatic visualization of the data and control flows. According to Moreno et al., (2004), the idea of Jeliot3 is to involve students in the construction of their own

programs and at the same time to examine the visual representation of the program execution. Students are engaged with the tool and are learning by doing.

According to Moreno et al. (2004), one of the reasons why Jeliot3 was developed is because visualization of object oriented concepts, for example objects and inheritance, are important and these concepts are not easily grasped by novice programmers. The development of the Jeliot family started when the first system Eliot was developed to help in the production of algorithm animations. Jeliot3 helps to provide clear semantics resulting in students engaging more in the learning process. They further state that teachers and students are able to share graphical and verbal vocabulary which eases the discussion of programming concepts.

Tekdal (2013) reports that another experiment was conducted by Moreno and Joy (2007) using the Jeliot3 animation tool. The study found that, although Jeliot3 animations are difficult for novice students, it helped them to debug their programs and that using the program was easy. According to Tekdal (2013), findings of another qualitative study with Jeliot 3 noted by Sivula (2005) showed that animation programs may increase the motivation of students and helps them to learn programming basics. In additional research on Jeliot3 being used as a learning tool carried out by Hongwarittorn and Krairit (2010), it was discovered that Jeliot3 may help to improve the learning performance of students in Java programming, compared to those who do not use the tool. However, it was also found that the use of the tool did not make an impact on the students' long-term attitudes toward programming.

JIVE

Tekdal (2013) describes JIVE as an interactive execution environment and provides a rich visualization of the execution of Java programs. According to Tekdal (2013) JIVE can be used for attaining a better insight into the behaviour of correct programs and to fix incorrect programs. It is also helpful as an educational tool for teaching object oriented programming in graduate and undergraduate programming courses.

EduVisor

According to Tekdal (2013), EduVisor is a program visualization tool developed by (Moons and Backer, 2009) and is based on the four high-level design principles from theories of learning and the theory of perception. Findings of the experiment by Moons and Backer (2012) show that the students and teachers regard the environment and the way it is used as instrumental to their education which can help with the understanding of programming concepts.

ViLLE

Rajala et al. (2008) developed a dynamic program visualization tool called ViLLE, for teaching programming to novice programmers. The tool can be used both in a teaching environment to demonstrate the dynamic behaviour of program execution and for independent learning over the web. It aims at providing a more abstract view of programming and is language independent. The tool has a built-in syntax editor with which syntaxes of built-in languages can be modified.

Rajala et al. (2008) conclude that that program visualization, in particular the ViLLE tool, enhances students' learning regardless of previous programming experience. In addition, according to the authors, the tool seems to benefit novice learners more than students with some previous experience.

2.2.2.2 PROGRAM / ALGORITHM ANIMATIONS

Price et al. (1983) define algorithm animation as a tool that gives a graphical representation of the algorithm that is needed to implement a program. These tools are to a large extent program language independent. It gives students a visual representation of how the algorithm works. Each animation must be created individually.

Lattu et al. (2000) state that algorithm animation could be one way to help novices understand the logic of algorithms. A program visualization and animation tool according to Tekdal (2013), allows the execution of program lines to be presented using

graphical effects. This tool shows the execution of a program and helps students to better understand program codes.

Next, examples of program / algorithm animations are discussed.

VINCE

According to Ala-Mutka (2004), a web-based tool called VINCE was developed by Rowe and Thorburn (2000), to help students understand the execution of C programs.

VINCE was written completely in Java therefore it is accessible in the form of an applet on a Web page with similar features as that of BRADMAN, including a memory map where the variable content can be easily inspected (Garner, 2003). The tool visualizes the workings of a C program step by step showing the contents of the computer memory and provides an explanation in each step to the users. Studies indicate that VINCE had a positive effect on student learning and is considered a good supplement for an introduction to programming course (Ala-Mutka, 2004; Linden and Lederman, 2011; Tekdal, 2013).

2.2.2.3 INTERACTIVE ANIMATIONS AND GAMES PROGRAMMING

Amory et al. (1999) found that students are motivated to use games as a useful learning tool and that play could influence the development of visualization, experimentation and creativity. “Games could support the development of communities of practice that include reflective activities, interest, understanding and epistemologies” (Amory, 2010: 810). Two examples of interactive animations and games programming are discussed next, namely, ALICE and SCRATCH.

ALICE

Alice is a free, open-source programming language that uses an innovative three dimensional (3D) programming environment to create animations or games. Cooper et al. (2000) define Alice as a three dimensional virtual world that teaches 3D graphics to novice programmers. Developed at the Carnegie-Mellon University, the tool allows you to create virtual worlds with a series of 3D objects that includes for example people, animals and chairs. The Alice interface can be used to place the objects into the virtual

world. With the drag and drop feature, the user can then program a series of parallel or serial actions with the objects in the virtual environment. The sequence of time-based events can be played at any point in time in an animated environment.

According to Daly (2011), the Alice environment makes it easy for students to create animations and/or games by eliminating the frustration and focus on the syntax of the language. With Alice, students can focus on the concepts while creating code, without worrying about semicolons, curly braces, etc. Daly (2011) further states that the software allows students to immerse into a programming rich multimedia environment, enforcing the object oriented concepts needed by students who are considering taking their programming to the next level. Alice, which is programmed in Java, gives students the opportunity to experiment with objects, classes, inheritance, expressions, conditions, loops, variables, arrays, events, recursion, and data structures. Animations and interactive games can be created by dragging and dropping graphical segments of code onto the editing area.

Daly (2011) concludes by supporting the claims that Alice has proved to be enjoyable, promotes positive attitude toward programming, raises motivation and improves retention. In addition, students' confidence levels toward programming concepts are improved by the comfortable programming environment provided by Alice.

Kelleher and Pausch (2007) found that presenting computer programming as a means to the end with storytelling motivates middle school girls (age 11 to 15) to learn programming. According to them storytelling with Alice is based on an existing programming environment called Alice 2.0, which allows novice programmers to create interactive 3D virtual worlds. Programs in Alice 2.0 are constructed by dragging and dropping code elements, eliminating the likelihood of making syntax errors. Users of animation programs in Alice are able to see their mistakes as they occur. Students have the opportunity to gain experience with programming concepts that include looping, conditionals, methods, parameters, variables, arrays, and recursion.

Cooper et al. (2000) found that Alice is a suitable programming language for novice programmers. The executions of animated programs are immediately transparent to the students. The highly visual feedback leads to an understanding of the actual functioning of different programming language constructs.

SCRATCH

Nam et al. (2010) describe Scratch as a multimedia programming setting that is appropriate for novice programmers because it is easy and interesting to learn and facilitates the development of problem solving skills. Problem solving skills is referred to as one of the primary mechanisms needed to solve new problems and the learning of new knowledge by applying existing knowledge (Gijselaers, 1996). Problem solving skills thus affect the way of thinking in our professional and personal lives.

Scratch is defined by Harvey and Monig (2010) as a programming language for children where no keyboard skills are required, because the primitive program elements are available with a graphical drag-and-drop user interface. The object oriented programming concepts are illustrated in the form of multiple animated sprites. These sprites are taken from the Scratch library or imported from any picture file with each sprite having its own script area and its own local variables.

Computer programming requires computer equipment to automate the abstract thinking process in the quest for problem solving, therefore the experience of the computer programming process leads to more reinforced abstract thinking (Nam et al., 2010). Scratch is further described as an educational programming language and teaching-learning tool that piles up graphic building blocks whose commands are all different according to colour and form on objects called sprites. The educational programming language has many benefits, is easy to understand and learn and can make insightful programming possible.

2.3 DIFFICULTIES OF LEARNING PROGRAMMING

For Robins et al. (2003), acquiring and developing knowledge about programming is a highly complex process. It involves a variety of cognitive activities, and mental representations related to program design, program understanding, modifying, debugging and documenting. At the level of computer literacy, it requires construction of conceptual knowledge, and the structuring of basic operations such as loops and conditional statements into schemas and plans. It also requires developing strategies flexible enough to derive benefits from programming aids (programming environment, programming methods).

The central issue evident from these discussions are students' lack of meta-cognitive skills to critically unlock key programming concepts (Ismail et al., 2010). Although this is not uncommon with first year students at most universities in SA, the situation at CPUT is critical because the bulk of its students come from poor schooling background (Scott et al., 2011:446). As stated previously, this gap could have strong implications for the teaching and learning of programming at CPUT and exploring alternative teaching and learning methods can be a useful intervention.

Novice programmers experience a wide range of difficulties and deficits. Programming is generally difficult and has high dropout rates, (Robins et al. 2003). Butler and Morgan (2007) present the results of a survey given to students enrolled in an introductory programming course at Monash University in Melbourne, Australia. Some of their findings indicate that first year introductory programming courses have a relatively high fail rate and that most first year students perceived programming as the most difficult and least interesting subject. Students may then often find themselves in a discipline in which they may not have had any prior knowledge of programming when commencing tertiary education for the first time after completing secondary study. They further state that a basic background in Mathematics and English is normally required to commence a study in many IT courses. Introductory programming courses and computing fundamentals can therefore become a learning challenge to students.

Learning how to program, as stated by Vogts et al. (2010), is complex and includes learning the syntax of a programming language and the use of a program development environment to construct, debug and execute programs. Novice programmers spend extensive time attempting to successfully compile and execute a program resulting in novices thinking that programming is all about getting the syntax right. It is further stated that a need exists for specialized program development environments to be developed while keeping the pedagogical aims in mind.

The findings of a study conducted at the Nelson Mandela Metropolitan University, South Africa by Vogts et al. (2010), indicate that many tertiary institutions make use of professional program development environments in introductory programming courses. They conclude that this is often the result of external pressure to teach programming in a development environment that is popular in industry and that the use of pedagogical program development environments has proven to be successful for novices learning to program.

Another study conducted at two SA tertiary institutions, namely, University of KwaZulu-Natal and Mangosuthu Technikon, by Pillay and Jugoo (2005) reveals the following main causes: learning difficulties and errors made by students in the examination; poor planning and problem solving abilities; a lack of understanding of programming constructs; a lack of knowledge and understanding of the programming knowledge; and insufficient conceptualization of the execution of the program. Other problems identified by the study include the incorrect transfer of knowledge and difficulties with modularization and iteration programming concepts. The knowledge that the students have attained from previous programming examples are incorrectly applied to current problems.

Rogerson and Scott (2010) suggest that a mathematical background with the focus on logic and problem solving is highly important. A positive relationship should therefore exist between mathematical ability and successful student programmers.

With the comparison of novice and expert programmers, Robins et al. (2003: 138) asked the following two questions: “what are the properties of expert programmers?” and “what are the resources and processes involved in creating or understanding programming?” In relation to these questions, Salleh et al. (2013) identify the three main components in programming as: the program, programming tools and the programming language. According to Salleh et al. (2013), programming tools are a key element in programming since they play an important role in programming development and implementation. The software or environment provided by programming tools allow programmers to give instructions, test and then implement the program. The ability and skills needed to use programming tools are considered equally important to skills in syntax and logic. Programming tools are considered as one of the key elements in teaching and learning of programming which relates to issues such as pedagogy, curriculum and programming languages.

According to Winslow (1996), novice programmers are limited to surface and superficially organized knowledge and that they lack the detailed mental models. He further stated that novices fail to apply relevant knowledge and approach programming line by line rather than breaking it up in meaningful structures.

Mayer et al. (1989) and Byrne and Lyons (2001) both state that the personal properties of the students affect their performance. General intelligence and mathematical science abilities seem to be related to success in learning to program. Different student behaviours in confronting a problematic situation can be recognized. Robins et al. (2003) believe that the main source of difficulty is not the syntax and the understanding of concepts but rather the planning of a basic program. Students can explain and understand programming concepts of individual concepts like pointers, but fail to apply it in their programming or combine it into valid programs. Robins et al. (2003), further suggest that another issue that complicates the learning of programming is the distinction between the model of the program as intended and the model of the actual program.

The difficulties of learning programming are categorised by Kanaparan et al. (2013) into two broad categories: demanding cognitive load, and the behavioural traits of the student. According to the authors these difficulties occur irrespective of programming environment or the type of programming language used in an introductory programming course. As stated by the authors, research into the difficulties of learning programming has proposed numerous solutions which include using educational technologies for learning programming, improving the course content and identifying predictors of programming success. It has been reported that these solutions have had insignificant influence on the students. However, a strong relationship has been found between behavioural traits of students and their learning and performance in programming.

Cooper et al. (2000) argue that the real difficulty for novice programmers is that it is required of students to learn why and how a program solves the problem when learning to write, test and debug programs. Many students struggle to visualize the execution steps of the program, making it difficult for them to figure out what went wrong when the program is not working. Cooper et al. (2000) concludes that students have difficulty in applying problem solving techniques by developing algorithms, when learning to program.

According to Yacob and Saman (2012) programming students struggle to master the required competencies and skills, resulting in a high failure rate in introductory programming courses. Yacob and Saman (2012) suggest that students should be motivated and do substantial practice to develop good programming skills.

It is further reported by Yacob and Saman (2012) that some of the reasons stated by students to dislike the programming subject are: programming is a boring subject; they do not understand the lecturer's explanation; the teaching method is not interesting; there are not enough exercises or practices during class lesson; they do not finish course material and then copy from friends without trying to answer questions.

Derus and Ali (2012) identify computing background; level of computing experience; difficulties experienced while learning programming; and factors that lead to poor performance in programming course as the factors that lead to difficulty in students learning programming. Memory related concepts relating to creating a clear mental model of memory movement during program execution were found to be the most difficult topic faced by students. Derus and Ali (2012) further state that one of the factors of students' difficulty in learning programming is their inability to visualize the program state during code execution.

According to Salleh et al. (2013), students need to understand the syntax of a programming language to learn how to develop a program. Frustration and lack of motivation to learn programming are often a result of the complexity of programming and the difficulty to comprehend program logic. These factors lead to high dropout rates in programming courses at most universities.

The ability to compile and coordinate the different components of a program is some of the greatest impediments for novice programmers. It is expected of students to identify and fix problems within their own programs. Goosen et al. (2007) suggest that the programming environment should contain compilers that display easily understandable error messages and offer effective tools to debug errors.

2.4 FIRST PROGRAMMING LANGUAGE

As stated by Goosen et al. (2007), it has been identified in the SA context that the language used should be a general-purpose programming language and not one that has been developed for a certain setting. It is therefore stated by the Department of Education (DoE) as recorded in Ali and Kohun (2005) cited by Goosen et al. (2007), that the purpose of the IT subject serves as a problem solving discipline and a tool for thinking and reasoning with the focus on the development of solutions to problems. Hence, the focal point should rather be on the introduction of general problem solving concepts than on teaching the syntax of a specific programming language. The language used in a first programming course should prepare the students to have a

solid foundation in good programming practices and facilitate the use of the meta-cognitive components of problem solving (Goosen et al., 2007).

An empirical study, with all the role players in the curriculum process of IT in all provinces of SA as participants, was conducted by Goosen et al. (2007) to choose a first programming language. The respondents agreed that the language of choice should provide an instructional environment where skills in higher order thinking and problem solving are developed. According to Goosen et al. (2007) it is of high importance that novice programmers develop a good theoretical understanding of programming in general, in order to equip them with the necessary programming skills to learn future languages and environments.

Rajala et al. (2008) suggest a term called programming language independency paradigm where they argue that learning how the different programming concepts work is more relevant than focussing on the syntactical issues of a specific program language.

According to Pears et al. (2007) after they had conducted, surveys, they found that C, Java and C++ were the most widely used programming languages used both in industry and education. However, there has been much discussion on the suitability of these languages when introducing programming to novices. Pears et al. (2007) argue that, in contrast to languages that have been designed with a specific purpose for education (e.g., Python, Logo, Eiffel, Pascal), languages such as Java, C and C++ have not been designed with this specific purpose in mind.

2.5 NOVICES VS EXPERT PROGRAMMERS

2.5.1 ATTRIBUTES OF A GOOD PROGRAMMER

Wiedenbeck (1985) found that expert programmers are much more accurate and faster than novices when it comes to performing low-level programming tasks for example finding syntax errors and understanding code functionality. Wiedenbeck (1985) and Fix et al. (1993) found that there is a link between programming experience and

characteristics in their mental representations for example the mapping of code to goals and recognition of recurring patterns. Agno-Balabat and Rojo (2013) also found that to become an expert in programming involves a lot of practice and requires the aptitude and capabilities to understand the execution of a computer program in order to form a valid mental presentation of the problem to be solved by the program.

2.5.2 CHARACTERISTICS OF NOVICE PROGRAMMERS

Novice programmers are defined by Ala-Mutka (2004) as programmers that lack the knowledge and skills of programming experts. Novices often fail to apply the knowledge they have obtained correctly. It is further stated that Novice programmers lack the knowledge and skills to easily grasp and apply programming concepts. Their skills and knowledge are limited to surface knowledge of programs and they generally approach programming “line by line” rather than at the level of bigger program structures. According to Goosen et al. (2007) the needs, knowledge and abilities of novice programmers are considerably differently from those of expert programmers. Agno-Balabat and Rojo (2013) therefore proposed program visualization as an educational tool that integrates programming tasks with visualizations of program execution to help novices locate programming errors. The authors conclude that visualization as a helpful tool for novices would assist novices’ practice in writing code and visually tracing it to debug software, making it easier to shift from novice to expert programmer.

Novices spend little time in planning and testing code, and when necessary, try to correct their programs with small local fixes instead of more thoroughly reformulating them. Also, the knowledge of novice programmers tends to be context specific rather than general and they often fail to apply the knowledge they have obtained correctly. In fact, an average student does not usually make much progress in an introductory programming course (Ala-Mutka, 2004). Atachiants et al. (2014) identify testing and debugging as the two very complex areas for novice programmers and reported that some researchers found programming tools that support source-level debugging with data visualization to be more effective. The authors agree that planning is common amongst novice programmers and that the absence of good planning might result in more bugs. The incremental running and iteratively testing of code while new code is

being written, have been found to be an effective debugging strategy for both novice and expert programmers

According to Smith and Webb (2000), novice programmers have difficulties in developing, comprehending and debugging of a computer program. Each individual student therefore brings into the learning experience a unique blend of knowledge, beliefs and fears. Students use their own metaphors to try and make sense of their learning. Furthermore, novices lack the necessary analytical skills resulting in the learning of programming to be highly complex for them. Experiences allow problem representation to be conceptualized. They further state that the meaning and importance students attach to original experiences and action taken by them is influenced by the different styles of learning by students. The pre-existing knowledge can then prevent students to see the programming problem as a problem. On the behavioural differences between expert and novice programmers, the findings of Loh and Sheng (2013) indicate that there exists a trend in novices to follow rules “blindly” when solving problems as they still need to acquire the context in which those rules operate. They will learn to apply the right rules with the right conditions as they develop in proficiency to solve the problems. However, expert programmers have a tendency to solve problems based on their instinct and ignore or sometimes even freely break the rules.

A group of novices learning to program will typically contain a huge range of different backgrounds, abilities and levels of motivation and it results typically in a huge range of unsuccessful to successful outcomes (Robins et al., 2003). They further found from a survey (covering background, intended major, expected workload, etc.) of students in an introductory programming paper that the most reliable predictor of success was the grade that the student is expected to receive. They indicated that students in general have a reasonable accurate sense of how they would do in the first two weeks of their course. Campbell (2013) also identified motivation to be the major factor affecting novice programming performance and that it is positively associated with student grade. Intrinsic motivation was found to be higher for students with some experience. However,

in relation to self-efficacy in particular, it has been reported that no evidence was found that prior programming experience affected success.

In this study, the researcher wants to investigate whether students can develop their skills and willingness to learn the programming concepts without emphasising ethnic and educational backgrounds.

2.6 HOW NOVICES LEARN TO PROGRAM

Soloway and Spohrer (1989), outline deficits in novices' understanding of various specific programming language constructs such as variables, loops, arrays and recursion., note shortcomings in their planning and testing of code, explore more general issues relating to the use of program plans, show how prior knowledge can be a source of errors, and more. Novices are very local and concrete in their comprehension of programs (Robins et al., 2003). Novice programmers spend very little time planning and testing code but rather tend to attempt small "local" fixes rather than significantly reformulating the programs, (Linn and Dalbey, 1989). Novices have a poor grasp of the basic sequential nature of program execution. Winslow (1996) concludes that novice programmers know the syntax and semantics of individual statements but do not know how to combine these features into valid programs. Novice programmers have trouble translating a solution by hand into an equivalent computer program.

Dehnadi (2009) agrees that novices find it difficult to understand the syntax and underlying semantics of a programming language with no comprehension on the capabilities of the computer. The author further states that since novices are at the beginning of their mental model development, they lack the mechanical understanding and therefore build a poor mental model which does not satisfactorily meet their learning requirements. In the case of recursion and iteration in programming, students with poor mental models of the mechanical process result in them adopting poor learning strategies.

Despite some evidence that a well-designed programming environment can assist students learning to program, McIver (2002) argues that there have been few, if any,

direct evaluations on whether the choice or design of programming development environment has a real impact on learning.

Utting et al. (2013) describe today's students as belonging to the Net generation, who are active experiential learners dependent on technology for accessing information and interacting with others. Compared to previous generations, these students may not willingly engage with instructional resources, such as text books. According to Utting et al. (2013), the expectation of "always on" access to the internet as they code is likely to influence the strategies which students adopt in attempting to solve programming problems.

According to Sorva et al. (2013), instead of seeing particular programming concepts (e.g., objects, recursion) as active components of a dynamic process that occurs at runtime, novices see these concepts merely as pieces of code. Sorva et al. (2013) further states that learning to program is sometimes perceived primarily by novices as learning to write code in a particular language, rather than learning to design behaviours to be executed by computers. The way novices' reason about programs and the way they practice programming is transformed by their learning to relate program code to the dynamics of program execution (Sorva 2010).

Yacob and Saman (2012) argue that, although confidence is not in itself a reliable predictor of success, it plays a significant role in the successful outcome of students learning to program. Motivation was found as one of the major reasons for students to drop out IT courses. It is further stated by Yacob and Saman (2012) that students will learn to know programming if they are interested or motivated. Teachers must be able to tune into the motivation of the class, further encouraging students to attach some sort of value to learn.

According to Derus and Ali (2012), the majority of students agreed that, apart from having discussions with their lecturers and their peers the practical or laboratory activities could help them to learn fundamental programming more effectively. Because

of its dynamic concept, learning to program needs to involve practical activities and intensive training. Through active learning strategies in laboratory activities, the students' understanding of difficult terminology in programming as well as stimulating their interest in the field of programming will be addressed.

It is further argued by Derus and Ali (2012) that, in order to succeed, novices need a set of learning strategies to help them to cope with their process of programming. The absence of the ability to visualize the program state during code execution is one of the factors that lead to novices experiencing difficulty in learning to program. Linden and Lederman (2011) suggest that students who play an active role in constructing knowledge will have both greater retention and greater learning enjoyment than students involved in passive pedagogical teaching approaches.

The different kinds of characteristic behaviour are evident when observing novices in the process of writing programs. Perkins et al. (1989) distinguish between two main types: "stoppers" and "movers". When confronted with a problem, stoppers simply stop, abandoning all hope of solving the problem on their own. Students' attitudes to mistakes or errors are important. Those who are frustrated or have a negative emotional reaction to errors are likely to be the stoppers. According to the authors, movers are students who keep trying, experimenting and modifying their code. Movers use feedback about errors effectively and have the potential to solve the current problem and progress.

Rodrigo et al. (2009) indicate that the negative affect and behaviours have a strong influence on how novices learn. When confronted with errors in their programs, students respond by either disengaging from the task by giving up or they will try fixing the bugs by guessing. When students are given a programming problem, one group of students will stop trying to solve the problem when they run into difficulties. This group will then wait for the answer to be given or immediately ask for assistance. The other group however, will keep on trying to solve the problem by using the error messages displayed or start over from the beginning. Robins et al. (2003) indicate two types of novices: an

effective and ineffective novice that is students who learn without excessive effort and those who do not learn without inordinate personal attention.

In a study done by Vogts et al. (2010) at a SA university the following factors such as self-belief and motivation of novices learning to program are important and should not be ignored. It was found that a pedagogical program development environment will have a positive effect on the perceptions of novice programmers learning to program, specifically the feelings of achievement and learning. These positive perceptions could help alleviate some of the difficulties experienced during the learning process. According to Rogerson and Scott (2010), the students' experiences with learning to program are affected by a lack of confidence or apprehension regarding their ability to write a program. It is therefore suggested that programming students must overcome their barriers to learning programming through self-efficacy by believing in themselves and their abilities. Students need to have self-confidence, courage and self-esteem and must be willing to take responsibility for their own learning (Rogerson and Scott, 2010).

Butler and Morgan (2007) suggest that the level of conceptual difficulty in programming as a subject is an important consideration in the delivery of the learning material. Domain issues such as syntax operate at a low level of conceptual difficulty. A concept like loop or decision may be considered as mid-level. Novice programmers may find the leap between understanding the concept to implementing them, very difficult. The biggest challenge therefore does not appear to be the understanding of basic programming concepts but the ability to apply them. In the context of CPUT, DS1 is one of four subjects that new students undertake in their first year of study for the National Diploma Information Technology (NDIPIT). The aim of this core subject is to provide these new students, most of whom have no prior programming experience, with the fundamentals of problem solving, program design and implementation in the context of the C++ programming language. Since this subject is considered introductory, the curriculum spans from low to mid-levels of conceptual complexity.

2.7 SCAFFOLDING

Scaffolding is defined by Van Arsdale (2010) as tasks given to students to carry out under the guidance of a teacher or more skilled peers, in order to achieve knowledge through collaborative talk and shared understanding. Scaffolding can therefore be seen as supportive behaviours by experts to support the development and progress of students. The term scaffolding, which was introduced by Wood et al. (1976), refers to providing temporary help for students to learn successfully in a teaching-learning process. "Scaffolding can be applied to peer interactions when learning a computer program like Scratch" (Van Arsdale, 2010). Resnick, one of the developers of Scratch, believes that traditional education does not teach students to be creative thinkers and problem solvers. Resnick deems creative thinking to be the key to success and satisfaction, both professionally and personally. Scratch allows users to collaboratively create rich media projects which include animations and video games, using their computer and mathematical skills (Van Arsdale, 2010).

Nam et al. (2010) state that programming is difficult and time consuming to learn and that the use of Scratch is the best way to show how to easily approach programming education, together with the supply of scaffolding. One of the goals of Scratch is to foster creative thinking which influences students' motivation, concentration and achievement through constructionist learning.

The following conceptual framework (see Figure 2.1) depicts the concepts and the relationships between them relevant to this study.

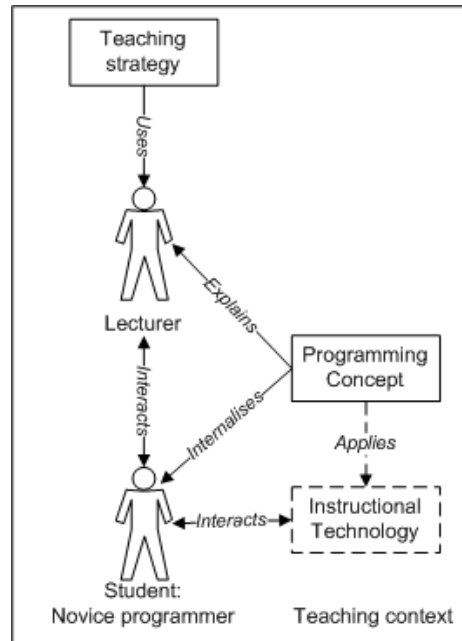


Figure 2.1 Conceptual framework of concepts

The aim of this study is to investigate how the use of instructional technology can be used as a teaching strategy to explain abstract programming concepts to novice programmers. The conceptual framework was derived from the literature to indicate the different concepts and their relationships relevant to this study according to the aim and depicted as Figure 2.1. The main concepts are: the student as novice programmer and lecturer as the humans and then teaching strategy, programming concept and instructional technology as the non-human concepts. Students will interact with technology as part of their learning process to internalise and apply programming concepts. Scratch, as an example of instructional technology, will be introduced as an intervention to improve the understanding and learning of these concepts. Technology will enhance the interaction and collaboration between students and instructors. The conceptual framework will guide the research in exploring alternative teaching methods and support the development of learning materials to teach programming to guide the empirical investigation.

The next chapter focusses on the overall process of collecting data and the methodology that was applied to conduct the research activity.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 INTRODUCTION

Research methodology refers to the overall process of collecting information and data in order to find out the result of a given research problem. It defines how the research activity was conducted, how to proceed and measure progress made. This chapter gives an overview of the methodology that was used and how the research problem was identified.

The research protocol outlines a detailed set of instructions and procedures that was followed in conducting the intended study and the collection of data. Yin (1994) asserted that a research protocol allows the researcher to detail in advance procedures and requirements to be followed during data collection. This has provided direction for research, which helped to improve the reliability of the research findings.

This chapter focuses on how the research problem was investigated. It provides a theoretical overview of the methodology that was used.

3.2 RESEARCH APPROACH

The research approach was dependent on the nature of the research problem which in this case is: the majority of first year IT students at CPUT are novice programmers and lack strong logic and reasoning as well as other IT skills that can facilitate their interpretation and application of key concepts in programming.

This research study deals with a complex problem that is highly contextual. Not only is the educational field complex but this study specifically considers it in an IT domain. Designing and developing software solutions in the IT field require specific skills and the factors that influence the acquiring of the necessary software development skills are also complex. According to Ala-Mutka (2004), programming includes problem solving skills and knowledge of programming languages with their tools. It is further stated that

the common approach to teaching programming skills to software developers is to first teach them the basics of a programming language. Programmers have to deal with abstract concepts as they translate their understanding of the anticipated solution and how to use the programming language into a software program. The syntax of the programming language is usually also new to the novice programmers and often difficult to learn. It is therefore important to first have a good understanding about the problems that novice programmers face when learning the programming concepts and to use a programming language to translate these concepts into programming constructs before any intervention can be designed. Part of this exploration was to use an existing intervention to establish how the novice programmers respond to this example of an instructional technology.

The philosophical assumption for this study was influenced by how the novice programmers socially construct their understanding of the programming concepts. Their translation processes result into programming constructs which are subjective. It is assumed that novice programmers construct their own realities based on their understanding of the programming process that will be influenced by their environments, backgrounds and previous experiences. The research approach was therefore inductive and meanings were derived from the different observations of novice programmers in a practical setting.

There is still too little understanding of the novice programmers' internalization of the programming concepts to already design an intervention. Based on the complexities it was decided to rather focus on the problem and regards this study as a step towards an intervention. The actual intervention will be designed in further research based on the findings. This study therefore focused more on the novice programmers' interactions with an example instructional technology tool during the programming process as they try to make sense of how they should use the programming concepts to develop a software solution for a given problem. The emphasis was on the researcher "standing back" and allowing the novice programmers' voices to be heard and their actions to be observed.

Although the research is envisaged as two parts, it was decided to follow a design research methodology where the first two phases form part of this study. Design research has the following main phases: 1) problem identification; 2) identification of tentative products and design principles; prototyping and assessment of the preliminary products and theories (Plomp, 2009). The outcome after phases one and two is tentative products and theories and after the third phase, the problem resolution and advancing theories. The design research methodology is illustrated in Figure 3.1. This study only focused on phases one and two. As part of the first phase, the author conducted an experimental case study under his control. Case study research is a good research strategy for research problems in real-life settings, where “how” questions are typically posed (Rowley, 2002). For example, “*How do novice programmers respond to an instructional technology to use programming concepts?*”

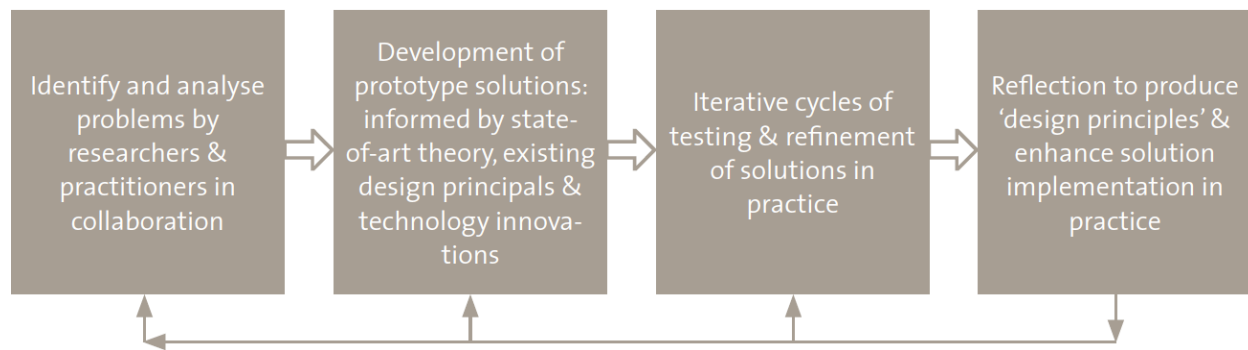


Figure 3.1 Design research methodology (Plomp, 2009:14)

During the second phase, a focused literature review was done to establish the current status of the research and to develop the theoretical conceptual framework. The analyzing part was done by using a specific instructional technology tool to establish the novice programmers’ responses to using this tool. Analyzing of the practical context part was done with three workshops where the novice programmers were working with the example instructional technology tool in a practical setting.

Insights gained from the empirical data collected in this study may be used to enhance teaching and learning of abstract programming concepts at CPUT and other institutions

of higher learning. The key focus of this research approach was to enhance the understanding of challenges in teaching programming concepts and contributing to the body of knowledge on the problems experienced by novice programmers. In this study, design research was found as the suitable approach

3.2.1 DESIGN RESEARCH

The researcher found it appropriate to engage in design research as a suitable approach. Design research is summarized by Plomp (2009:13) as “the systematic study of designing, developing and evaluating educational interventions as solutions for complex problems in educational practice, which also aims at advancing our knowledge about the characteristics of these interventions and the processes of designing and developing them”.

By doing educational design research the researcher aimed to provide insights and contributions for generating mechanisms to address the problem of grasping and understanding abstract programming concepts. Design research as a research approach or strategy facilitates the development of an intervention to explore alternative methods for teaching and learning programming at CPUT. According to Plomp (2009: 31) design research is conducted in real world settings because it addresses complex problems in educational practice. This research therefore aimed at designing an intervention in a real world setting using the successful DS1 students of the previous year, current first year and repeating DS1 students as the participants.

3.2.2 RESEARCH SETTING

The study was conducted at CPUT using the first year level DS1 students as a pilot group because the researcher has unfettered access to this cohort of students. An in-depth investigation was done on all the first year level DS1 students taught by the researcher at CPUT. The researcher used purposive sampling which is popular in qualitative research (Patton, 1990), as opposed to random sampling. Because the emphasis was to be on quality rather than quantity, the objective was not to maximize numbers but to become “saturated” with information on the topic (Padgett as cited in Bowen, 2005: 217).

3.2.3 RESEARCH METHODS

The qualitative and quantitative methods were used to collect empirical data for this research study. Although a mixed method approach was not specifically used in this research project, the combination of quantitative and qualitative empirical data provides to some extent the possibility of triangulation. The integrative methodological approach by combining quantitative and qualitative methods employed by the same study is referred to as triangulation (Bryman, 2006).

Kelle (1995) as cited in Fielding and Schreier (2001) distinguishes three meanings or models of triangulation:

- triangulation as the mutual validation of results obtained on the basis of different methods (the validity model)
- triangulation as a means toward obtaining a larger, more complete picture of the phenomenon under study (the complementarily model)and
- triangulation in its original trigonometrically sense, indicating that a combination of methods is necessary in order to gain any (not necessarily a fuller) picture of the relevant phenomenon at all (the trigonometry model).

The triangulation model used for this study is to obtain a larger, more complete picture of how novice programmers use programming concepts with the use of a software tool for a specific problem.

3.3 RESEARCH METHODOLOGY FOR THIS STUDY

Only once a better understanding is gained about the challenges that novice programmers experience in practice will it be possible to design an appropriate intervention. Ultimately the aim will be to design an intervention that will assist novice programmers to master the programming concepts but falls outside the scope of this study.

The study was conducted at a single institution – CPUT - and it focused on a single issue, namely the learning of programming concepts with the use of an instructional technology - for novice programmers. A qualitative mode of enquiry ensured an in-depth

discussion to questions given to the subjects. The questions addressed the knowledge and skills needs of programming learners and the challenges for students and educators to evolve from traditional to technology-supported teaching and learning. Although a design research strategy is followed, the stages of case study research, recommended by Yin (1994), were used as a broad outline for conducting his study. The following four stages are given in table 3.1:

Table 3.1 Four stages of case study methodology (Yin, 1994)

Design the case
Conduct the case
Analyze the case evidence, and
Develop the conclusions, recommendations and implications

The research was an exploratory qualitative pilot study of instructional technology as a tool for enhancing the traditional method of teaching structured programming and design techniques. The focus group for the pilot study was students at the institution, both successful and repeating, who had just completed their first year. The study population for the actual research included all the current first year DS1 students of CPUT. The purpose of the exploratory qualitative approach was to enhance the understanding of the problems experienced by novice programmers and to generate mechanisms to address these problems. The insights gained from the pilot study will be used to design and implement a technological solution to enhance teaching and learning of programming concepts. Study participants were included or excluded based on their willingness to participate.

The research was conducted as follows:

- Conduct a thorough literature review to establish the concepts relevant for the study.
- Interact with students who have just completed their first year (both successful and repeating students) and invite them to suggest possible technology solutions that they think could have helped them to master the programming concepts. This is done in the form of workshops
- Analyse the findings of the literature and ideas generated by students
- Analyse and interpret the results.

Yin (1994) suggests that the case study investigator must be able to operate as a senior investigator during the data collection and needs to know the following aspects:

- Reason for conducting the study
- Type of evidence being sought
- Variations that might be expected

The behaviours and interactions of the research subjects were carefully observed and respondents were encouraged to give credible responses which were converted into useful qualitative data. A qualitative approach advocated for more diversity in response and capacity to adapt to new developments or issues during the research process.

Figure 3.2 provides a diagrammatic representation of the research approach.

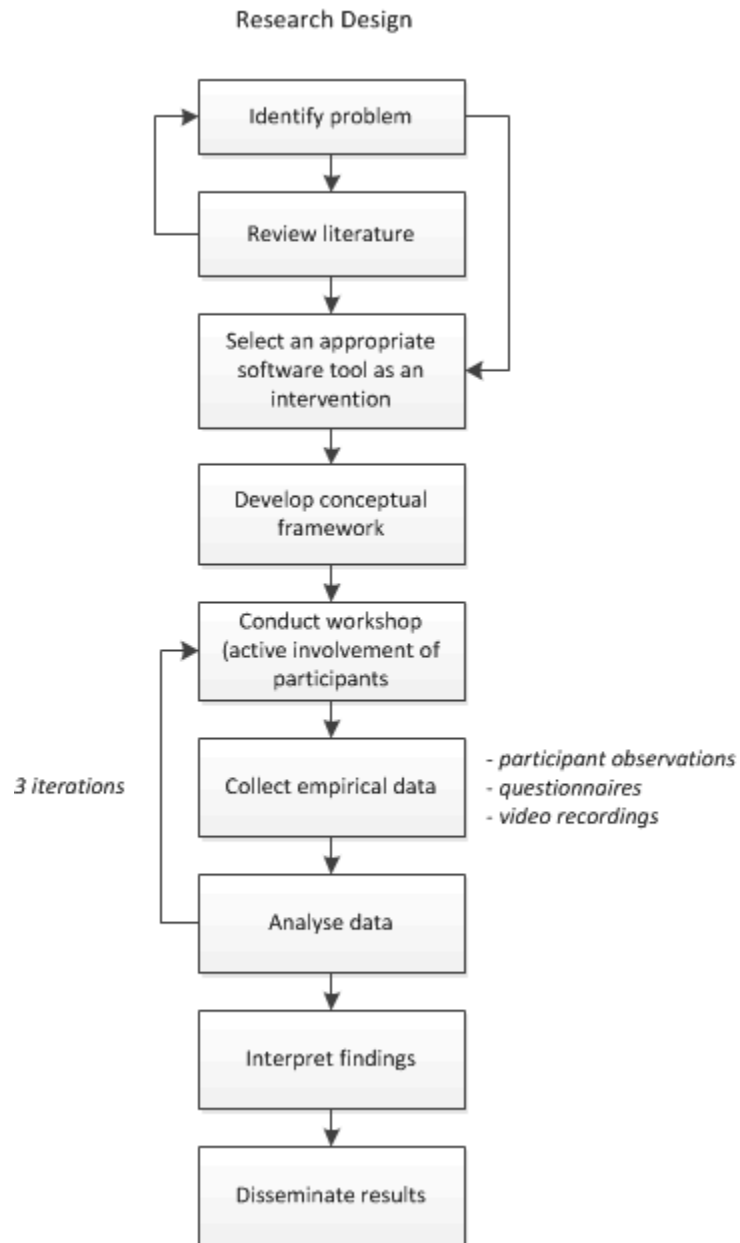


Figure 3.2 Framework of the research study

3.4 SELECTION CRITERIA AND PROCESS

An internal admission rating process is conducted by the DIT to determine student acceptance of the programme for a NDIPIT, a three year higher education qualification. For applicants who matriculated from 2008 onwards, points scored will equal the best five scores for their final year of secondary education subjects of which one of the subjects must include the Mathematics or Mathematics Literacy rating. To be considered for admission to the NDIPIT, an applicant must obtain a minimum score of 20 with Mathematics or a score of 22 with Mathematics Literacy. In addition to the institutions admission requirements, all applicants for the NDIPIT must obtain the following minimum rating: Home (first) Language 4, Second Language 3, Mathematics 3 and Mathematics Literacy 5. The rating codes are given in Table 3.2:

Table 3.2 Admission rating for matriculants from (2008) onwards

2008 – 2011 Matriculants		
Rating Code	Rating	Marks %
7	Outstanding Achievement	80 – 100
6	Meritorious Achievement	70 – 79
5	Substantial Achievement	60 – 69
4	Adequate Achievement	50 – 59
3	Moderate Achievement	40 – 49
2	Elementary Achievement	30 – 39
1	Not Achieved	0 – 29

For applicants who matriculated before 2008, points scored will equal the best five scores. To be considered for admission to the NDIPIT, an applicant must obtain a minimum score of 25.

Table 3.3 Admission rating for matriculants before (2008)

Matriculated Before 2008					
Mathematics		English			
HG	SG	1 st	2 nd	Other	
Symbol		Matric HG		Matric SG	
A		8		6	
B		7		5	
C		6		4	
D		5		3	
E		4		2	
F		3		1	

DS1 is one of two major subjects in NDIPIT and requires a minimum final mark of 50% to pass. Hence, should a student fail this subject, it will have to be repeated in the part-time programme of the following year. However, if students fail both their major subjects, they may be excluded from returning to the NDIPIT the following year.

3.4.1 PROFILING

Participants were randomly selected and were contacted in advance in writing (Appendix A) to inform them about the research project and to obtain their permission to contact them on their mobile telephone numbers. It was planned that each target group per workshop would contain at least thirty DS1 students, although this was not possible in the eventual practice. The potential respondents for these focus groups were purposefully selected according to a matrix to include a variety of students representing different groups. Participants were selected according to gender and ethnic groups. There was also an attempt to include students with varying marks, e.g. to include students with good, average and low marks. Study participants were included depending on their willingness to participate; therefore a list with possible substitutes was also compiled. The focus groups were organised outside formal class times, usually during academic registration, university vacation periods or over lunch breaks.

Participants were selected from a list of students who had successfully completed the DS1 subject the previous year, unsuccessful students who were repeating the subject and new students. Successful and repeating students were randomly selected based on their final mark scored in the previous year. New students were selected based on their points scored during the selection process for admission into the IT programme.

3.5 RESEARCH INSTRUMENTS FOR DATA COLLECTION

The data collection process was aligned with the principles of social constructivism, which views learning as a social construction of knowledge from shared meanings. The data collection took in consideration various aspects of participants involved in the process. This included their characteristic novice behaviour and attitude towards the use of technology. To develop trust and to comply with internationally accepted ethical standards, the researcher offered anonymity by not linking any name of individuals to a completed research instrument.

Multiple sources of data gathering, which includes participant observations, video recording, a questionnaire, and document analysis, were used as research instruments. The use of observation and document analysis as complement data gathering sources is widely accepted as a method of enhancing validity (Kirk & Miller, 1986 cited in Parry 1998:96). The researcher used video as a way of representing insights based on his observation.

3.5.1 WORKSHOPS

A total of three workshops were conducted, each one following the exact same format. The target groups for the workshops were successful DS1 students of the previous year (2011), the current first years (2012) and the repeaters (2012) who failed DS1 in 2011 but satisfied the policy for exclusion from study at CPUT. All three workshops were an interactive and participatory exercise.

At the start of the workshops the objectives and processes were discussed with the participants. Scratch, an innovative and well-researched tool that offers a visual programming environment was introduced. Participants were given a task to design a

new clock using the Scratch tool. After completion of the clock project, each participant was given the task to design their own game or animation, allowing them to engage in exploratory self-directed learning. At the end of the workshop each participant answered a self-completed questionnaire of their own experiences and how the tool helped them to better understand the abstract concepts in programming. These responses were then converted into useful qualitative data.

During the workshops, the researcher acted as a “participant observer” and he played both roles: as a participant (a lecturer performing his teaching duties) and as an observer witnessing learning taking place during the process. As a ‘participant’ the researcher was not doing what those being observed did, but interacted with them, to varying degrees, while they used Scratch to do the given tasks. The data collection workshops were well planned and constructed and included the DS1 students of the IT Department. Consequently, the instruments that were used resulted in a high degree of reliability, validity and findings of the research. Also the use of field notes and memos helped the researcher to determine if the validity of the findings had been affected in any way.

The observer paid attention to key issues such as:

- Familiarization with the use of instructional technology tools
- The level of digital divide that exists amongst students from cross-cultural backgrounds.
- Student access to a PC and the internet, and
- Instructors integrating technology into their teaching

3.5.2 PARTICIPANT OBSERVATION

Field-notes and video recordings were used as a data collection method based on participating and observing. Laurier (2010) states that participant observation involves spending time working with people or communities in order to understand them and further suggests that participant observers should keep written field-notes or video notes of their research. Participant observation should not be regarded as an approach which can be effectively used in isolation from other research procedures Jackson

(1983). The aim of the researcher therefore was to stay as close as possible to the observable facts being studied and therefore tried to be a part of the things being observed.

3.5.3 VIDEO

Video recording was used to capture the live user experiences and participant activities that would take place in the actual workshops. The researcher did not spend too much time on pondering whether participants would be acting natural with the presence of the video camera because this was an alternative data collection method in a traditional lecturing setup. The video material was analysed for its relevance for the design task. This media gave the researcher the opportunity to witness the nonverbal expression of feelings, collaboration and discussions of participants. According to (Ylirisku and Buur, 2007), videos can be used for data collection, interpretation and evaluation. Video is used as a sense-making tool which focuses the attention onto specific aspects of the data without losing the emphatic qualities of the data.

3.5.4 QUESTIONNAIRE

At the end of the workshop participants were asked to provide feedback on an open-ended self-completed questionnaire from their own experiences. Participants were reminded that responses should not be about the Scratch tool but the ability of the tool to help them to better understand the abstract programming concepts. These responses were then converted into useful qualitative data. A questionnaire (Appendix C) was used as a data gathering instrument to collect responses from the participants. The questionnaire was aimed at a focus group, using the successful DS1 students of the previous year, current first year and repeating DS1 students of CPUT, as the study participants. After the workshops, participants were asked to carefully think of an alternative idea that could be more relevant to our teaching and learning which the researcher could explore.

In the first part of the questionnaire, participants had to explain their experiences of learning and understanding programming concepts at the beginning of their first year. The second part of the questionnaire was about their experiences with the tool (Scratch) which includes: navigation, visibility, ease of use and single-window user interface or

multi-purpose design; elimination of the compilation step; help screens, default parameters and illuminating demonstrations for commands; visual feedback during code execution and troubleshooting; ability of an incorrect program still running by eliminating syntax and runtime errors; and variables as concrete or visible objects.

Participants had to give their personal opinions on how these functionalities of the tool (Scratch) would influence or assist new students with the learning of programming concepts. In the final part of the questionnaire, participants had to suggest any possible ideas that could help new first year DS1 students to master programming concepts. They also had to give their opinion about the suitability of Scratch as an instructional tool to assist novice programmers with the learning of new programming concepts, keeping in mind the diversity of students at CPUT.

3.5.4.1 QUESTIONNAIRE ITEMS

The questions on the evaluation form supported the objective of the session and were encouraging them to rather elaborate than giving brief answers.

The instrument consisted of four sections, namely:

- own experience of learning and understanding programming concepts
- experiences with the Scratch tool
- possible ideas that may help DS1 students to master programming concepts
- suitability of Scratch as an instructional tool to assist new programmers

3.6 DATA ANALYSIS

The analysis of data was carried out considering both the qualitative and quantitative data collected. Data was organized in tables and sorted by respondent, question and time returned. The data was coded to then identify the different themes based on the codes as part of the data analysis. The themes identified during the analysis of the data were presented in different tables. Codes were created for the occurrence of fundamental but previously uncovered issues and connections. Data was reduced into smaller groups by means of coding to gain an understanding of the enquiring issues.

Coding allowed the researcher to see the relationship between categories and patterns. Codes were applied and reapplied to the qualitative data by grouping and regrouping data in order to consolidate meaning and explanation. A process called codifying was carried out by the researcher whereby the codes were applied and reapplied to the qualitative data, grouping and regrouping the data in order to consolidate meaning and explanation. The identified themes, based on the analysed empirical data, were compared to the themes identified in the literature.

The analysis of questionnaire responses and observations was based on an inductive approach geared to identifying patterns in the data by means of thematic codes. This process was carried out in a systematic way that resulted in credible answers to the research questions and objectives embedded within the study. Bernard (2006 as cited by Jameson, 2008) states that “analysis is the search for patterns in data and for ideas that help explain why those patterns are there in the first place”.

Thematic networks are described by (Attride-Stirling, 2001) as “a way of organizing a thematic analysis of qualitative data. Thematic analyses seek to unearth the themes salient in a text at different levels, and thematic networks aim to facilitate the structuring and depiction of these themes. Thematic networks systematize the extraction of: (i) lowest-order premises evident in the text (Basic Themes); (ii) categories of basic themes grouped together to summarize more abstract principles (Organizing Themes); and (iii) super-ordinate themes encapsulating the principal metaphors in the text as a whole (Global Themes). These are then represented as web-like maps depicting the salient themes at each of the three levels, and illustrating the relationships between them”. See the structure of a thematic network in Figure 3.3

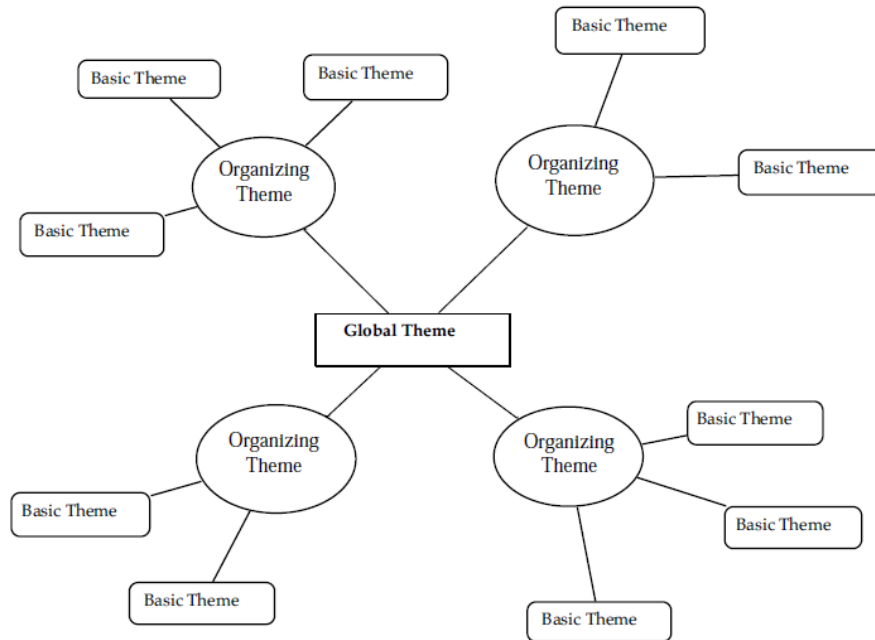


Figure 3.3 Structure of a thematic network. (Source: Attride-Stirling, 2001)

Content analysis was applied to analyse the questionnaire and target group in the relation to the research questions and sub-questions. The analysis of questionnaire responses and observations was based on an inductive approach geared to identifying patterns in the data by means of thematic codes. This process was carried out in a systematic way that would result in credible answers to the research questions and objectives embedded within the study. The final themes apparent from the data were then presented as a pattern.

3.7 INTERPRETATION

The video recordings were fully studied and the observations together with the field notes and questionnaire responses formed the raw data for further analysis. The process of data collection and data analysis was continuous and iterative. Content analysis was conducted to look for categories, patterns and trends with the use of thematic data analysis. The findings derived from the analysed data were used to attach meaning to the findings as part of the interpretation process.

3.8 ROLE OF THE RESEARCHER

The researcher played the role of participant observer to observe the learning taking place during the process and to interact with the participants. According to Laurier (2010) participant observation, which has no predetermined formal steps to doing it, involves spending time working with people or communities in order to understand them. Hence, the method based on participating and observing was found appropriate since the researcher had free access to the cohort of DS1 students at CPUT and as a qualitative researcher, felt very personally involved in every step of the research process. The researcher participated in the workshops both as researcher and lecturer. The students already knew him as lecturer and his role in the workshop was therefore perceived as something already familiar to them.

With respect to anonymity, Fink (2000) argues that respondents in a qualitative study will not be anonymous to the researcher as they will be in a quantitative study, therefore the researcher will feel obliged to protect the data that was collected to strengthen the researchers' loyalty towards the respondents. However, no names of respondents were recorded on research instruments that were used for data collection by the researcher. Before, during and after the workshops the researcher took field-notes as a supplement to video recordings, as methods of data collection which allowed the researcher to stay as close to the observable facts being studied as possible. Notes were written by hand and some were later typed and stored as text files. Note-taking helped the researcher recall sufficient details of the workshops which were described and analysed.

3.9 CHAPTER SUMMARY

This chapter provided an overview of the methodology employed for this study. It presented a detailed description of the research design, the research instruments for data collection and the selection criteria and process used to randomly select the participants. It also detailed the data analysis process and how the empirical data was interpreted.

The following chapter focuses on the results obtained after analysis of the data.

CHAPTER 4

DATA ANALYSIS AND FINDINGS

4.1 INTRODUCTION

This chapter presents the analysis of the data gathered to achieve the objectives of this study. Additionally, the analysis and findings are only based on the items in this chapter. The analysis of the statements will be presented in Chapter 5.

Furthermore, this chapter will present how data, gathered from three workshops including observations of the researcher, was collected and processed in response to the problems identified in Chapter One. The workshops were conducted with current first year as well as successful and repeating DS1 students of the previous year. The expected outcome of the workshops was to provide a basis for recommendations to support the developing of learning materials and approaches for basic programming courses and find a mechanism to help our first year students to cope with the abstract concepts of programming. The analysis of the qualitative data proceeded into the findings of the research.

4.2 BACKGROUND

CPUT is one of four universities but the only University of Technology (UoT) in the Western Cape province of SA with a current enrolment of over 33,000 students (Cape Peninsula University of Technology Active Web, n.d.). The current student population is very diverse in population group (Figure 4.1). Consequently, diversity in student population has resulted in diversity in terms of educational standard and computer expertise. Four population groups (African, Coloured, Indian and White) were represented in the sample population. African students had the highest percentage of enrolment for 2012 (56%), followed by Coloured (29%), White (14%) and Indian (1%). The need to profile students in population groups is a requirement from the national government for political reasons. In this study no distinction is made between the different population groups.

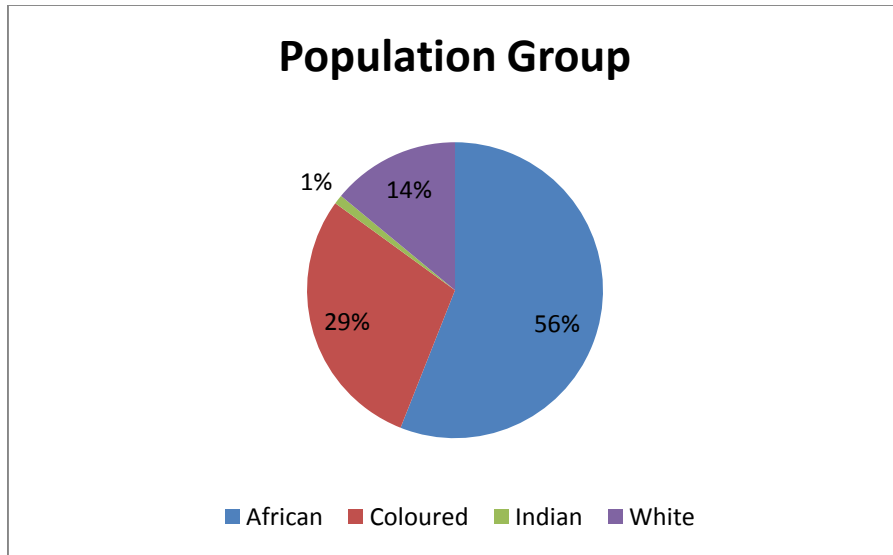


Figure 4.1 Student Enrolment by population at CPUT (2012)

CPUT consists of six faculties: Applied Science, Business, Education and Social Sciences, Engineering, Health and Wellness Sciences, and Informatics and Design. The IT Department, one of twelve departments within the Faculty of Informatics and Design, offers a course in IT with programme options that present career opportunities in Business Applications, Communications Networks and Software Development. Software Development students are prepared for careers in computer programming, systems analysis and design, and database administration. DS1 is the first of three parts of the Development Software pillar to cover the three years of the national diploma in IT. The other major subject is Information Systems. Business Applications students are prepared for careers in IT business solutions with less emphasis on programming and Communication Network students are prepared for careers in network development and administration and systems administration.

In the context of CPUT, the greater part of the first-time entry student population comes with mediocre or inadequate problem solving skills. This situation has strong implications for the teaching and learning of programming at the university. Most of the students have little or no prior programming experience and are therefore at risk for not succeeding in their first year programming subject. Factors contributing to this situation include a lack of prior computer experience, limited and poor mathematics preparation,

poor self-efficacy and the adjustment from secondary to university study. This impacts negatively on the delivery of the introductory programming subject and throughput rates of the university. In addition to the institution's admission requirements, all applicants for the National Diploma Information Technology (NDIPIT) at CPUT must have obtained in matric a minimum of moderate achievement (40 – 49%) in Mathematics or a minimum substantial achievement (60 – 69%) in Math Literacy.

4.3 WORKSHOPS DESIGN

Each workshop had the same objectives, agenda, participation process, materials and supplies. These were discussed first after which each workshop was discussed as they were conducted in practice.

All three workshops followed the exact format. Participants were given the task to create a new project called Clock and experiment with importing backgrounds that are not part of the standard set that comes with Scratch and also experimented with editing the size of Sprites (objects). A worksheet (Appendix B) with detailed instructions on how to create the Clock project including visual screenshots of the different phases of development and the image of a clock was made available on the desktops of all computers in the lab. The Clock project was a relatively easy challenge but was useful for participants who were new to Scratch, to encourage them in representing their knowledge and ideas after seeing the ability of the tool.

After completion of the Clock project, participants were instructed to create their own game project with Scratch and were encouraged to look at existing projects in the Scripts Library of the tool to see how those games were coded. Participants were presented scaffolded projects to design a game or create an animation which allowed them to engage in exploratory, self-directed learning. It was expected of participants to include in their project a conditional statement, variables, coordination and synchronization, sequencing and iteration (looping) programming concepts and skills supported in Scratch. These concepts are also covered in the first year of the IT Diploma. The idea of the exercise was for participants to use their own experiences and to establish how this tool could help them to work out the solution and consider

Scratch as an aid to help them better understand the programming concepts. Participants were required to make a presentation in order to demonstrate their work in front of their peers and the workshop facilitator. Each participant duly completed questionnaires at the end of the workshops, based from their own experiences.

4.3.1 WORKSHOP OBJECTIVES

- Identify some of the challenges faced by novice programmers.
- Identify possible skills gaps of students learning programming concepts
- Explore how students respond to a technology solution
- Observe students' use of instructional technology in practice
- Obtain students' feedback on the use of Scratch as an example of instructional technology
- Obtain feedback and observations from different groups to establish whether there are any differences

4.3.2 AGENDA

1. Workshop objectives and process (15 min)
2. Introduction of Scratch tool (30 min)
3. Work session creating a new clock project using Scratch (15 min)
4. Creating a new game using Scratch (90 min)
5. Demonstration of actual game (30 min)
6. Completion of questionnaires (30 min)

4.3.3 PARTICIPATION PROCESS

Participants were given the option of working in pairs or as individuals with one presenter facilitating the entire workshop. The workshop sessions were interactive with each participant given the task to design a new clock project using the Scratch tool. After completion of the Clock project, participants were instructed to create their own game project.

4.3.3.1 MATERIALS AND SUPPLIES

An innovative learning tool called Scratch was used in the three workshops because it is a suitable programming language that offers a visual programming environment, allowing users to create interactive, media rich projects and is freely available at

<http://scratch.mit.edu>. Programming in Scratch is done by snapping together colourful command blocks representing statements, expressions and control structures, to control 2-D graphical objects called sprites moving on a background called the stage. Since Scratch encapsulates state (variables) and behaviour (scripts) although with neither class nor inheritance, it is an object-based language but not an object-oriented one.

The researcher decided to use Scratch as an example of an instructional technology given that it was available and has already been used for similar purposes as it was designed to contribute to the understanding of basic programming concepts such as event handling, sequential, and conditional statement. A participant was supplied with an electronic copy of detailed hand-outs, including a worksheet (Appendix B) that was used in the workshops. The worksheet contained detailed instructions on how to create the Clock project including visual screenshots of the different phases of development and the image of a clock. Each participant was supplied with a personal computer with Scratch installed on each of them.

4.3.3.2 ACTUAL WORKSHOP

The practical part of the workshop was an exercise with challenges to be solved.

The Clock project was to encourage participants in representing their knowledge and ideas after seeing the ability of the tool. Participants were also given the task to design a game or create an animation which allowed them to engage in exploratory, self-directed learning. The new projects had to include programming concepts such as conditional statement, variables, coordination and synchronization, sequencing and iteration (looping). Participants had to use their own experiences and establish how the Scratch tool could help them finding a solution to the problem presented. Each participant had to demonstrate their new game or animation.

Participants were asked to place themselves in the role of the “teacher” for new students, throughout the workshop. The workshop was video recorded by a senior student of the Film and Video department in the FID of CPUT. During the workshop, the researcher was acting as a “participant observer” and played both roles: as a participant (a lecturer performing his teaching duties) and as an observer witnessing nonverbal

expression of feelings, collaboration and discussions, learning taking place during the process, paying attention to key issues such as the use of the Scratch software tool. By participating and observing, the researcher made use of field-notes and video recordings as methods of data collection.

For the purposes of clarity, the findings (observed patterns) will be presented as they occurred in separate workshops.

4.4 DEMOGRAPHICS OF PARTICIPANTS

Biographical data of participants was required in order to determine the demographic profile of the participant groups only to ensure a balanced group of participants. Demographic details, including gender and ethnic groups, were obtained for 50 students from the CPUT database. Four population groups (African, Coloured, Indian and White) were represented in the sample population. African students had the highest percentage of participation in this exploratory study (72%), followed by Coloured (20%), White (6%) and Indian (2%).

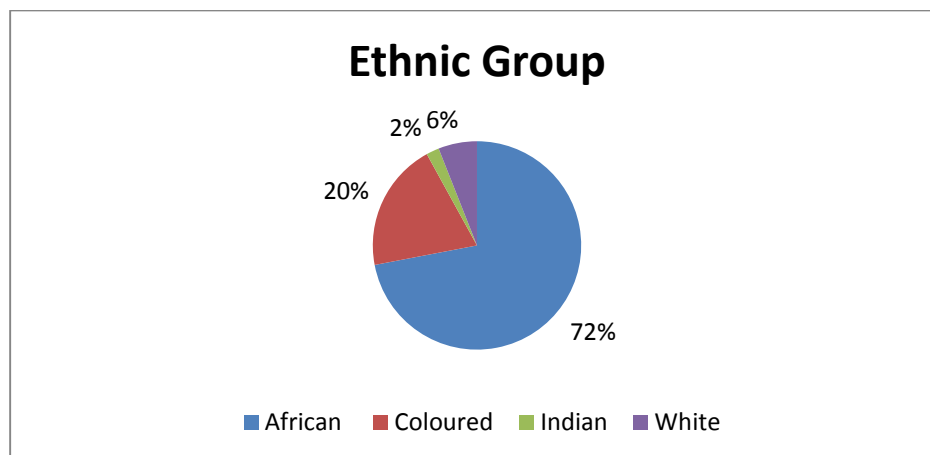


Figure 4.2 Ethnic Profile of Participants

A total of 50 students participated in this exploratory study: 19 in workshop 1 (15 male and 4 female), 15 in workshop 2 (10 male and 5 female), and 16 in workshop 3 (9 male

and 7 female). In all three workshops, the male participants (68%) were found to be dominant in total over the female participants (32%). This is in line with the typical cohort of IT students where there are more male than female students.

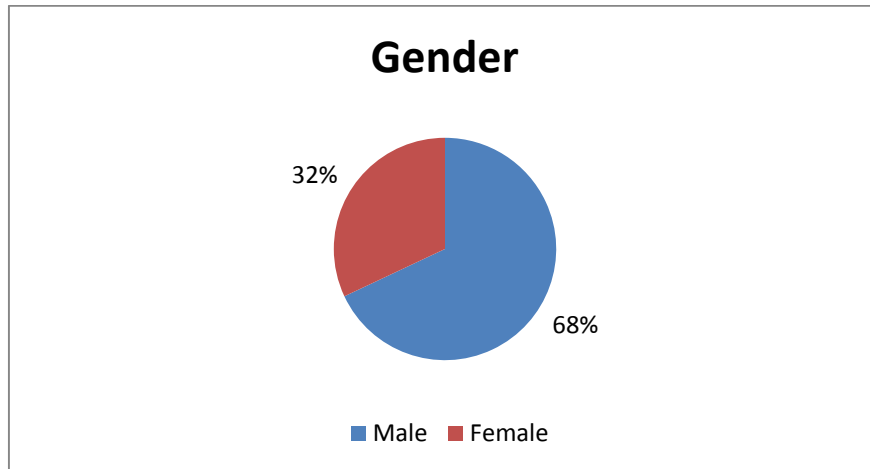


Figure 4.3: Gender Profile of Participants

The current first year IT student population at CPUT is very diverse which has resulted in diversities in terms of educational standard and computer skills background. The reason for the diversity is that they are from different social, cultural, geographical and language backgrounds. For most of the students English is their second or even third language and yet this is the official instructional language. It is recognised that these different backgrounds could contribute to the problems experienced by novice programmers but these were not specifically considered in this study. The majority of the participants in workshop 2 and workshop 3 had little or no prior experience of programming on their first day of arriving at CPUT.

4.5 ACADEMIC PERFORMANCE

DS1, a programming subject, is one of two major subjects in the first year of the NDIPIT. Four subjects in the first year form a common first year for further specialization during the second and third years of the qualification. This subject provides a basis for all the other programming subjects in further years. This subject is a pre-requisite for not only Development Software 2 (DS2) but also for Technical Programming in the second

year of the diploma. Basic principles and fundamentals of good programming are taught in this subject with a focus on programming proficiency. This will lay the foundation for the following two years of studies and further. DS1 is taught on a weekly basis which consists of six periods (45 minutes per period), namely two theory periods, two lab or practical periods, and two tutorial periods. During the practical periods students are expected to implement the work that was covered in the theory classes. There are approximately a total of 280 students in the first year who are divided into 8 groups with between 30 and 35 students per group. The group size is deliberately small to provide a better opportunity for the lecturer to engage in a more hands-on manner.

The content of the subject aim to teach students to:

- i. solve problems using structured design techniques
- ii. design program solutions using modern design techniques
- iii. develop a solution algorithm using pseudocode
- iv. use a general, modern program language to implement the solution algorithm

Problem solving, program design, C++ syntax and C++ implementation is covered in the first year subject.

This subject has three theory assessments, two practical assessments, and various class tests and class exercises. Each of the theory assessments consists of a written test that takes place during the official departmental test week towards the end of every term. One practical assessment is a practical test that will be written in the beginning of the fourth term – the students do this in a lab working on a specific problem that they must solve using a computer. The other practical assessment is a group project done in October, the final month of the same academic year, wherein groups of two or three students will design and program a small software system.

This software system is demonstrated to the lecturer and part of the assessment is to establish whether the system works for a set of test data. Throughout the year, at least once a month, small class tests or practical programs will be given to students and

evaluated in practice. DS1 is a continuous assessment subject; therefore every mark obtained throughout the year will count towards the final mark. A minimum mark of 50% is required to pass this subject. Below is a summary of the average marks for the different groups. These marks were extracted from the system and the averages were calculated for the specific group of participants.

Table 4.1: First year performance and acceptance

	Performance and Acceptance			
	Workshop 1 (successful)	Workshop 2 (new)	Workshop 3 (repeating)	
Average Acceptance Score		25 points (2012)		
Average Performance Marks				
Semester 1	65 % (2011)	42 % (2012)	35 % (2011)	39 % (2012)
Semester 2	89 % (2011)	62 % (2012)	37 % (2011)	75 % (2012)



Figure 4.4 Screenshot of the Scratch user interface

4.6 WORKSHOPS

4.6.1 WORKSHOP 1

4.6.1.1 TARGET GROUP

A group of sixteen second year IT students participated in workshop 1. Participants, randomly selected based on their final marks, had successfully completed their first year and were at the beginning of their second year of their NDIPIT, when the workshop was conducted. One can therefore assume that they had mastered the programming concepts.

4.6.1.2 RESPONSES

The researcher lectured some of the respondents in their first year. Since this was the first workshop a senior researcher, a social anthropologist in the Information Technology department, helped to facilitate the pilot workshop.

4.6.1.2.1 OBSERVATIONS

The researcher conducted a number of participant observations during the pilot workshop. After a brief tutorial on the Scratch tool, participants were seen to be inspired by showing eagerness to start exploring with the tool. They immediately started to experiment with the backgrounds, pictures and command blocks. The presence of the camera appeared not to be distracting to the participants for the reason that they stayed focused and relaxed. All participants in this workshop engaged passionately and enthusiastically with their practical exercise.

Only four of the participants used pen and paper to design the algorithm solution for the problem while the rest of participants jumped straight into creating their projects on the Scratch tool. It was observed that participants actively engaged, shared and collaborated effectively in groups. Participants appeared to apply their prior programming knowledge from their first year and easily and comfortably used the Scratch software.

It was clear to the researcher, based on the observations, that participants had the ability to grasp programming concepts immediately and with ease. Most of the

participants had no experience with Scratch. However, they were comfortable in experimenting with the different effects after an hour of being exposed to the tool. It was noticeable in the new games and animations of participants in the pilot workshop, that participants had innovative and creative thinking skills. Participants appeared eager to demonstrate their new projects.

4.6.1.2.2 VIDEO

During the interviews with participants, video recorded, learning interests amongst participants were observed. One of the participants responded that they were analysing coding instructions of existing game programs in order to apply similar programming styles. Peer support was evident amongst participants.

Students in workshop 1 were asked to demonstrate their working programs. Participants stated that, although they were struggling with certain programming concepts, they managed to apply their prior knowledge and worked independently to complete their projects. Participants appeared very confident while busy working on their tasks.

4.6.1.2.3 QUESTIONNAIRE

Some of the participants of workshop 1 indicated that they were already well prepared at the beginning of their first year. They further stated that they understood programming concepts, due to having Java as a subject at school.

“I have done programming at high school so when starting my first year it was more or less revision” (Respondent 1.3).

Participants were of the opinion that the best way to learn and understand programming concepts in order to construct a working program, was by regular practising.

“Without practice it is difficult to construct a working program” (Respondent 1.2).

Participants stated that the amount of detail covered by the lecturer, together with the many class exercise handouts and sufficient computer laboratory equipment, contributed constructively to their learning process.

“Dedicating most of our lessons in theory were far more depressing” (Respondent 1.7). “Learning and understanding took lots of hard work and dedication” (Respondent 1.9).

Some participants indicated that they have found it hard to adapt to the single-user interface of the Scratch tool compared to the interface of Visual Studio that they were used to. However, the majority seem to find the visibility and graphics interesting and the interface easy to navigate with.

“It has an easy-to-navigate user interface” (Respondent 1.5).

The tool offers too little control to the user and it would have been a lot better if there were more options available (Respondent 1.6). However, participants found the navigation and visibility of the tool to be simple, user-friendly and straightforward.

“The navigation and visibility of the tool has much more simplicity and is easy on the eye” (Respondent 1.4). “It teaches basic programming structures in a very user friendly way” (Respondent 1.6).

According to participants the elimination of the compilation step might appear to be very valuable. However, the students will need to learn how to compile eventually (Respondent 1.9).

It was therefore strongly suggested by participants that the compilation step not be eliminated as this will negatively influence the ability of the student to learn the concepts and that students will not know where mistakes were made. Students will be unable to determine whether their program is working or not.

“It will stop you from seeing where your problem is” (Respondent 1.3).

Respondent 1.10 argues that with the elimination of the compilation step, novice programmers will gain more confidence. Respondent 1.6, 1.7 and 1.8 agree that confidence levels would greatly increase if the student sees her/his program working the first time with no bugs or vague error messages.

“It will eliminate stress on the student, knowing that you will not get stuck the whole day with the coding of your program.”

In terms of learning the programming concepts, participants state that help screens, default parameters and illuminating demonstrating for commands is useful to the novice programmer, making it easier for them to understand the programming concepts.

“You learn easier with a visual concept of what you are learning” (Respondent 1.10).

In contrast, some participants argue that help screens and default parameters are not useful because the students will not understand what they are doing.

“Help screens should guide you in the right direction and not give you the correct answer or solution to the problem” (Respondent 1.1).

Visual feedback during code or script executing and troubleshooting was found by participants to be very useful as you can see the progress you have made in your program.

“For a novice it is helpful to see each step of your code executed” (Respondent 1.3).

“With visual feedback it is very easy when you know where the problem is or what you are doing right” (Respondent 1.8).

“With debugging, visual feedback is useful to pinpoint all mistakes and to know exactly what was done wrong” (Respondent 1.2).

According to participants, the ability of an incorrect program to still run by eliminating syntax or runtime errors will have a negative impact on the development of students and their understanding of the programming language.

“Students would learn to ignore syntax rules and will not learn proper coding ways and habits which will take them longer to becoming an expert programmer” (Respondent 1.9). “It will negatively influence their understanding of syntax

because Scratch does all the coding for you. Users do not actually write any code so they don't learn that part of programming" (Respondent 1.2)

Respondent 1.6 states that users will be unaware of their errors and that logic errors need to be pointed out to users so that they can learn from their mistakes.

"Programmers will keep making the same mistakes" (Respondent 1.1).

Having variables as concrete or visible objects can help the student have an idea of what their program is doing (Respondent 1.1). Some students have a better grasp with visual feed, therefore it is quite useful as programming concepts are very abstract and understanding how variables are passed in a program can sometimes be a problem (Respondent 1.3). It was also pointed out by participants that variables as visible objects make it easier to understand the background work of a program.

"Learning would be faster and easier in understanding the concepts of variables" (Respondent 1.8).

In order to master programming concepts, it was suggested by participants that first year students enrolling for DS1 should have a passion for the subject. Students must be able to conceptualize problems in a new, innovative way and need to think out of the box (Respondent 1.5).

"Reading and practicing is essential" (Respondent 1.1).

"It will be a good idea to have video clips which students can watch repeatedly" (Respondent 1.3).

It was further suggested by participants that Java instead of C++, should be taught as an introductory programming language for students to master programming concepts in their first year of computer studies. Respondents 1.7 and 1.9 recommended a visual tool like Scratch for novice programmers or Alice, which in their opinion is similar to Scratch but easier to use.

According to participants Scratch as an instructional tool to assist new programmers will help students better understand the programming concepts. However, the visual tool may be challenging for students with no prior access to computers.

“Scratch allows students to learn new programming concepts and is a great confidence builder. However, the fact that they are not writing or learning any code, might negatively affect their learning” (Respondent 1.10).

“For first time programmers, Scratch requires a lot of explanation” (Respondent 1.8). “Scratch includes a fun and educational side at the same time” (Respondent 1.5).

4.6.1.3 KEY THEMES

The following key themes emerged from participant responses in the pilot workshop:

- prior programming experience will enhance the preparedness of students
- software development and programming languages pre-acquaintance are useful
- practice is crucial to understand and apply programming concepts
- interface adaptation to single-user interface of Scratch is challenging.

Visualization is important for understanding and learning abstract and complex programming concepts. Knowing how to compile a program makes you a better programmer and helps to identify syntax errors. Syntax or run-time errors should never be ignored as it will hinder students in their development and understanding of the programming language. Students should have a strong passion for programming in order to solve challenging technical problems.

Variables as visible objects will help students understand the abstract concepts of programming Java should be considered the language of choice to teach students how to program. Similarly, a visual programming language like Scratch or Alice that uses an innovative 3D programming environment to create animations and games was suggested as the tool to assist novices.

Table 4.2: Summary of Workshop 1 results

		Workshop 1
Demographics	Males	79%
	Females	21%
	Indian	5%
	White	16%
	African	37%
	Coloured	42%
Academic Performance		
		Average Marks
	Semester 1	65% (2011)
	Semester 2	89% (2011)
Key Themes		
		<ul style="list-style-type: none"> • Prior programming experience will enhance the preparedness of students • Software development and programming languages pre-acquaintance are useful • Practice is crucial to understand and apply programming concepts • Interface adaption to single-user interface of Scratch is challenging

4.6.2 WORKSHOP 2

The second workshop followed the exact same format as that of the pilot workshop conducted with 2nd year DS2 students. However, during the second workshop, the researcher was acting as a “participant observer” and fulfilled the role of workshop facilitator.

4.6.2.1 TARGET GROUP

Sixteen first year DS1 students participated in the second workshop. Participants were first entry students, randomly selected based on the points they scored during the admission rating process conducted internally by the DIT

4.6.2.2 RESPONSES

4.6.2.2.1 OBSERVATIONS

Participants appeared excited and enthusiastic from the beginning of the second workshop and showed great eagerness to learn independently. All participants were consulted about the video recording of the workshop and indicated that they were not distracted in any way by the presence of the video camera. Participants appeared comfortable in following and grasping the concepts of the Scratch tool presented in the introduction section of the workshop and demonstration of the program conducted by the facilitator. Although all the participants were unfamiliar with Scratch, some of them were able to assist their peers who needed assistance.

It appears that students wanted to find solutions to the problem presented to them, on their own first, rather than depending on each other. Participants experimented with the importing of backgrounds that were not part of the standard set that comes with Scratch and also with editing the sizes of Sprites. Students appeared relaxed while engaging in the given task. The majority of participants successfully completed the clock project in the time allocated and appeared eager to assist their peers who had difficulties with the task.

Participants were given the choice to form groups but preferred to work on their own. Only two of the sixteen participants made changes on existing games in the Scripts library whilst the rest of the students designed their own games and animations. During the design stage, two of the participants asked for help with the usage of conditional statements in Scratch. After some assistance from the workshop facilitator, the students appeared to grasp and apply the concepts. Throughout the design and development of their new games and animations, participants appeared focused, quiet and had minimum interaction or collaboration with each other. No visible signs of nervousness or frustration by participants were observed.

Three participants volunteered to demonstrate their games and animations and entertained their peers with their creative designs of animations and a quiz game. The

presentations were observed to have more innovation than that of the participants in the first workshop, bringing a different creative element to the workshop. A question and answer session was allowed for the three volunteers to answer questions related to their designs. Participants were fascinated by the “prompt and accept” feature in the quiz game. Despite their challenges experienced with applying the timing and synchronization concepts using the Scratch tool, participants appeared excited and confident in the demonstration of their creative work.

4.6.2.2.2 VIDEO

Participants in workshop 2 exposed signs of uncertainty but at the same time excitement about the given tasks to create either a new game or make changes to existing programs. Despite the observed uncertainties, participants displayed eagerness to work on their projects and appeared very participative. Participants articulated dedication in that they were willing to demonstrate to the group their completed tasks. During the demonstrations, the eagerness to experiment and the creativity of participants could be observed.

It was perceived that animation programming was of great interest to the participants. One of the participants who demonstrated his project called “The Animated Comic Book” stated that he managed to complete the task despite his struggle with certain concepts. Participants expressed interest in the completed projects of their peers by asking questions on how certain tasks were accomplished.

4.6.2.2.3 QUESTIONNAIRE

Participants stated that the understanding of programming concepts became easier with practice.

“I have learned that programming needs time, practice and dedication in order to learn and understand it” (Respondent 2.12).

According to Respondent 2.2, being keen to know and allowing you to think logical is what helps to understand the programming concepts at the beginning of the first year of study.

“I have learned that in programming you need to be more creative” (Respondent 2.10, 2.15)

It was also stated by participants that one needs to be keen to learn and should have logical thinking skills. (Respondent 2.13) argues that “programming has many rules and it is hard to know all of those rules in time. Programming styles of lecturers and that in the text books are different and confuses students”.

“If you never had access to a computer, it is hard to understand how it works” (Respondent 2.13)

Responses from participants reveal that their experience about the visibility, ease of use and single-window interface of Scratch, is that the tool is fun, interesting and user friendly.

“Single-window interface with four main panes, allocates everything just a click away and simplifies programming for me” (Respondent 2.2).

(Respondent 2.9) experiences the functionality of the tool to be easy to use and very interesting when creating new games.

“Scratch is a more fun, more interactive, less complex approach to programming” (Respondent 2.5).

However, Respondent 2.10 argues that “it is not that simple to use Scratch especially if you are not familiar with it”.

Respondent 2.13 states that students will spend less time coding with the elimination of the compilation step. However, other participants argue that students will not be able to identify their mistakes or syntax errors, which may cause confusion and negatively influence the ability of the student to learn programming concepts. As a result, students will be struggling with programming in their second year of the course (Respondent 2.7).

“The negative part about it is that students won’t have to understand programming languages anymore” (Respondent 2.5)

“Programming will be made easy and students will lack creativity and design skills” (Respondent 2.9)

The opinion of participants with regards to help screens default parameters and illuminating demonstrations for commands is that it will be helpful for novices to better learn and understand the programming concepts. According to Respondents 2.7 and 2.8, it helps a lot if you are not familiar with the program and is also useful when creating games. However, (Respondent 2.12) argues that “help screens are not always helpful as they may seem. In order to create a successful program, programmers need to know exactly what they are doing”.

Participants state that with visual feedback during code execution and troubleshooting, students will know immediately where mistakes were made and that it will aid in the fixing of program errors.

“With visual feedback you can see where your program is incorrect” (Respondent 2.3).

According to (Respondent 2.6), the ability of the program to still run even if it is incorrect, “will make a student not know if the program is correct or not. The student will always repeat the same mistake”. It is also stated that eliminating syntax or runtime errors will confuse students, limit their programming skills and problem-solving ability and therefore negatively influence their learning of programming concepts.

“The outcome of programs will not meet its required specifications” (Respondent 2.8).

“Learning of concepts will be negatively influenced as students will not be able to understand the problem and how to solve it” (Respondent 2.13)

Participants stated that having variables as concrete or visible objects helped students to use them correctly and could reduce the difficulty of writing programs.

It is recommended by participants that lecturers do more practical work in class to help first year DS1 students master the programming concepts. It is further suggested that students must practice more every day and “apply the mind-set of a programmer” (Respondent 2.2).

“Students must know their programming languages” (Respondent 2.5).

“Students must familiarize themselves with more than one programming language” (Respondent 2.14)

A suggestion was made by participants that a visual and fun tool similar to Scratch be installed in all computer labs at the University. This will greatly assist students in mastering the programming concepts.

On the question of the suitability of Scratch as an instructional tool to assist new programmers, Respondent 2.8 stated that “Scratch is a very useful and informative tool that can assist and improve first year programming development”.

(Respondent 2.14) argues that “Scratch only allows you to do limited things compared to other advance software tools”.

4.6.2.3 KEY THEMES

The following key themes emerged from the responses in Workshop two:

- practice is essential to learn and understand concepts
- compilation step not to be eliminated
- help screens and default parameters very helpful
- visual feedback during execution very helpful
- incorrect program should not run
- variables should be presented as concrete or visual objects

Table 4.3: Summary of Workshop 2 results

		Workshop 2
Demographics	Males	67%
	Females	33%
	Indian	-
	White	-
	African	93%
	Coloured	7%
Academic Performance		
		Average Marks
	Semester 1	42% (2012)
	Semester 2	62% (2012)
Key Themes		
		<ul style="list-style-type: none"> • Practice is essential to learn and understand concepts • Compilation step not to be eliminated • Help screens and default parameters very helpful • Visual feedback during execution very helpful • Incorrect program should not run • Variables should be presented as concrete or visual objects

4.6.3 WORKSHOP 3

The third workshop followed the exact format as that of workshops one and two. As with the second workshop, the researcher was acting as a “participant observer” and fulfilled the role of workshop facilitator.

4.6.3.1 TARGET GROUP

Participants in the third workshops were students who failed DS1 the previous year and are repeating the subject. The repeaters were randomly selected based on their final mark for DS1 in their previous year of study. Names of randomly selected participants were published on the student notice boards.

4.6.3.2 RESPONSES

4.6.3.2.1 OBSERVATIONS

A number of participant observations were perceived by the researcher in the third and final workshop. The twenty-one participants appeared to be very energetic, excited and conversational on arrival at the workshop venue. Participants were given the option of working in groups of two, but they preferred to work individually. The video recording of the workshop was not in any way indicated as a distraction by the participants. During the Scratch introduction section, participants were supporting their peers who were falling behind. This was due to the fact that students were exploring Scratch instead of paying full attention or stayed focused on the introductory presentation. The facilitator also had to provide individual support and attention in some cases.

Participants appeared to be impatient in completing their given tasks without careful planning and thought. Hence, they decided to make changes to existing games and animations instead of creating their own. It was observed that participants in this workshop became more confident and started to enjoy the programming environment of Scratch tool as the workshop progressed. Even though participants in the third workshop were repeating the DS1 subject, they appeared to be unable to coordinate prior knowledge in grasping and applying concepts. None of the participants in this workshop were willing to demonstrate their projects.

4.6.3.2.2 VIDEO

During the interview video recorded in workshop 3, one of the participants indicated that the Scratch tool is fun, easy and simple to use. The participant expressed satisfaction for the fact that he did not have to do a lot of coding. A second participant that was interviewed expressed his view on the Scratch tool to be interesting as it does not acquire a lot of thinking on his part. Participants were of the opinion that Scratch tool would have helped them during their first year of study. Participants in workshop 3 acknowledge the Scratch tool as easy to use and that the tool just needed some focus to appreciate it completely.

It was further suggested by participants that the Scratch tool should be introduced to students during their first year of study because it would assist them with basic programming. It was observed that participants collaboratively engaged with their peers and appeared very enthusiastic and excited about the Scratch tool. One of the participants stated that the tool is a great aid during the design phase of programming.

4.6.3.2.3 QUESTIONNAIRE

Participants of the third workshop agree with participants of the first and second workshops that learning and understanding programming concepts requires a lot of practice.

“Programming was hard in the beginning but become more exciting with practice” (Respondent 3.5).

“Programming is something that needs to be practised regularly” (Respondent 3.6).

Responses from participants show that the lack of prior programming knowledge before studying at CPUT made it difficult to understand the programming concepts, despite the fact that they are repeating the DS1 subject.

“I had to adapt to something that was foreign to me” (Respondent 3.1).

“It was difficult because I did not have any programming experience” (Respondent 3.3).

“For a person who never did programming it was very confusing” (Respondent 3.6).

“Understanding the concepts of the programming language was very difficult” (Respondent 3.9)

However, Respondent 3.14 stated: “I am repeating programming for a second time and now I understand it”.

According to participants the navigation, visibility, ease of use and the single-window interface makes the tool easy to use and programming interesting. Respondent 3.12 states that “it is easy to develop your own game”.

“There is no coding involved, making it easier to accomplish a task” (Respondent 3.6).

“Navigation and visibility is not that bad as you can see clearly what the object on screen looks like” (Respondent 3.12).

Respondent 3.11 argues that “everything is found on one screen but I wish that the window that we use to design our programs can be enlarged”

The elimination of the compilation step was perceived by participants as making life easy for the student and leads to time efficiency. However it may also result in apathetic programmers, reluctant to develop their problem solving skills and enhance their ability of learning programming concepts.

Respondents 3.8, 3.12, 3.15, 3.10 are of the opinion that help screens, default parameters and illuminating demonstrations for commands will improve the programming skills of novices, help create interest in programming, and make it easy to learn programming concepts.

“They give me an interest to learn from my mistakes and get more knowledge”
(Respondent 3.11)

Participants state that visual feedback during execution and troubleshooting makes it easy to see where mistakes were made. According to Respondents 3.13, 3.15 it improves their ability to code more accurately and also hone their programming skills.

“I can see right away when I have made a mistake” (Respondent 3.7).

“It is very useful because it helps you to be able to see your mistakes and also be able to fix them” (Respondent 3.8).

Visual feedback is useful as it teaches you to learn from your mistakes and make corrections rapidly. It also allows you to see if the program is executing logically according to the desired specifications (Respondents 3.2, 3.12).

Participants are of the opinion that the ability of the program to run even if it is not correct will create confusion and encourage students to ignore the coding rules. The elimination of syntax or runtime errors will have a negative impact on the learning of

programming concepts and cause students not to be familiar with the language and rules of programming (Respondents 3.3, 3.7).

“It might create the false impression that the program has no errors”
(Respondents 3.13, 3.11)

It was suggested by participants that students should practice programming on a daily basis and consider using the Scratch tool at university and also at home to master the programming concepts. Students should consider watching or listening to programming tutorials on a video-sharing website like YouTube (Respondent 3.13). Students will master the programming concepts through hard work and must continuously try to improve on their thinking capability (Respondent 3.3). It was further suggested by Respondent 3.15 that students should consider working together collaboratively and also form study groups amongst each other.

Scratch is viewed by participants as an instructional tool to assist novice programmers learning new programming concepts, since it will be fun, helps students understand programming concepts better and also strengthen their programming skills. Respondent 3.1 affirms that it is easy to adapt to the Scratch tool.

“Scratch helps a lot to improve the knowledge and understanding of programming” (Respondent 3.15).

4.6.3.3 KEY THEMES

The following key themes emerged from the responses in the third and final workshop.

- programming becomes easier with practice
- prior programming knowledge essential
- elimination of compilation step very helpful
- help screens and default parameters improve programming skills
- visual feedback assist with troubleshooting
- incorrect program still running creates confusion

Table 4.4: Summary of Workshop 3 results

		Workshop 3	
Demographics	Males	56%	
	Females	44%	
	Indian	-	
	White	-	
	African	94%	
	Coloured	6%	
Academic Performance			
		Average Marks	
	Semester 1	35% (2011)	39% (2012)
	Semester 2	37% (2011)	75% (2012)
Key Themes		<ul style="list-style-type: none"> • Programming becomes easier with practice • Prior programming knowledge essential • Elimination of compilation step very helpful • Help screens and default parameters improve programming skills • Visual feedback assist with troubleshooting • Incorrect program still running creates confusion 	

4.7 CHAPTER SUMMARY

This chapter presented the responses of the research instruments. The findings of the items suggest that in the three workshops the respondents had different views. The analysis indicates that the successful group in the first workshop was well prepared for programming at the beginning of their first year.

The next chapter will discuss and interpret the findings and link it to the reviewed literature.

CHAPTER 5 DISCUSSIONS OF FINDINGS

5.1 INTRODUCTION

Chapter four represented the analysis of data gathered to achieve the objectives of this exploratory study. Chapter five draws the conclusion on these findings which includes the analysis on these findings. This chapter therefore discusses the findings relative to the aims and objectives of this study and literature.

5.2 COMPARISON OF PARTICIPANTS RESPONSES

Table 5.1 reflects the comparison that was drawn from participant responses in the three different workshops.

There were a total of 50 research objects in this study. The participants were undergraduate IT students at CPUT. From the exploratory study it was observed by the researcher that a tool similar to Scratch would help students improve their programming learning performance. All of the participants in this study did not have equal backgrounds in computer and programming skills when they entered the university for the first time.

Table 5.1 Comparison of participant responses

	Workshop 1 (Successful)	Workshop 2 (New)	Workshop 3 (Repeating)
Learning and understanding programming concepts at beginning of 1st year	More or less revision of last year; practice is essential	Practice and dedication is important; creative skills are needed; prior access to computers is essential	Programming requires regular programming; lack of prior programming knowledge makes programming difficult; very confusing and difficult for novice programmers
Experience with tool (Scratch), visibility, navigation & interface	Graphics interesting and interface easy to navigate; easy and straightforward; user-friendly	Interface is fun, interesting and user-friendly; less complex approach to programming	Navigation, visibility and interface easy to use; no coding makes programming easy
Elimination of compilation step	Negative influence on ability to learn; will boost confidence of novice programmers	Less time spent on coding; unable to identify mistakes or syntax errors; may result in lack of creativity and design skills	Makes life easy; may result in lazy programmers
Help screens and default parameters	Useful to novice programmers; makes understanding of programming concepts easier	Helpful for novice programmers to understand programming concepts; useful when creating games	Improves programming skills of novice programming; allows users to learn from mistakes and get more knowledge
Visual feedback during code/script execution	Helps with debugging and identifying mistakes	Helps to identify mistakes and errors	Helps to identify mistakes; helps develop programming skills; allows user to learn from mistakes
Program still runs even if not correct	Negative influence on understanding of programming language; may result in students ignoring syntax rules; students unable to identify mistakes	User will not know if program is correct; may limit programming skills and problem solving ability	Creates confusion and may encourage user not to know rules of programming rules; negative impact on the learning of concepts; can be very misleading
Variables as concrete (visible) objects	Useful; helps understand background workings of program; helps understand function of variables	Helps using variables more correctly; makes programming easier	No response
Ideas to help 1st year Development Software students	Passion for subject is essential; problem solving skills, reading and practice is essential; Java or Alice as first year programming language	More practical work daily; must think like a programmer; must learn more than one programming language	Practice daily and using Scratch tool; hard work and improving on the ability to think; should form study groups
Suitability of Scratch as instructional tool	Helps better understand programming concepts; maybe challenging to first year with no prior computer knowledge; no coding might negatively influence learning	Tool is very useful and informative; tool has very limited functionality	Helps understand programming concepts better and strengthen programming skills; improves knowledge and understanding of programming

Questionnaires were distributed to all participants for them to fill in before they left the workshop venues; therefore the response rate was 100%. Out of the fifty students who answered the questionnaires, 32% were female, 68% were male. Ages of the participants ranged from nineteen to twenty eight years old. Workshop one was facilitated a qualified Social Scientist and Anthropologist. Workshops two and three were facilitated by the researcher, who presented the workshop outline, program explanation and demonstration of the Scratch tool to participants.

5.2.1 LEARNING OF PROGRAMMING CONCEPTS

The responses from participants in the three workshops were similar in that the understanding and learning of programming concepts became easier with practice. Participants in workshop one indicated that they were well prepared in their first year as they already had Java as a subject at school. “I have done programming at high school so when starting my first year it was more or less revision” (Respondent 1.3). This is different to responses from participants in workshop three that found it difficult to understand the programming concepts due to a lack of prior programming knowledge, despite the fact that they were repeating the DS1 subject. “It was difficult because I did not have any programming experience” (Respondent 3.3).

From the empirical data it appears that having prior programming knowledge is essential for the learning and understanding of abstract concepts. This is similar to Greyling et al. (2006) and Derus and Ali (2012) stating that the under-preparedness of students has an impact on introductory courses that rely on technological tools as a pedagogical concern. A lack in computing background is therefore identified as one of the factors that leads to the difficulty in students learning programming. However, in relation to self-efficacy, Campbell (2013) differs by reporting that no evidence was found that prior programming experience affected success.

From the literature analysis it is evident that there is a great concern that novices spend too little time in the planning of a basic program. This is similar to the findings of Linn and Dalbey’s (1989) and Dehnadi (2009) that novices fail to apply syntaxes and basic concepts in their programs. As a result, novices lack the mechanical understanding and

therefore build a poor mental model causing them to adopt poor learning strategies. With relation to understanding the difficulties that students experience while learning to program, Jenkins (2002) came to the conclusion that there was nothing inherently difficult in a programming subject and that some students simply had no aptitude. Programming can be an enjoyable and creative activity when students are allowed to work on assignments that inspire them. Kanaparan et al. (2013) have found that there exists a strong relationship between the behavioural traits of students and their learning and performance in programming.

In the context of CPUT the researcher has observed that students might understand the syntaxes and concepts but fail to apply them in solving a problem. In most cases the students will then confuse their syntax handling difficulties with problem solving difficulties. Some of the students will become frustrated, believe that they cannot do programming and then transfer to another course or discipline of study. Ala-Mutka's (2004) discussion on the characteristics of a novice programmer reiterated some of the programming challenges at CPUT discussed in the foregoing literature review.

“Admission and retention rates to computer science university courses are falling and enrolment is male dominated. There is a need to both foster the development of computational thinking in young students and to motivate them to study computing subjects by improving the perception of computing, especially for girls” (Romero et al., 2007). The authors suggest that there are three issues that conspire to make computing concepts difficult to learn:

- i. context: problems and scenarios to which concepts are applied are often not very motivating
- ii. abstraction: some of the concepts are presented in a too abstract fashion
- iii. great attention to detail is needed to make something appealing work

5.2.2 VISIBILITY, NAVIGATION AND INTERFACE

Some participants in workshop one was of the opinion that it was difficult to adapt to single-user interface of the Scratch tool compared to the interface of Visual Studio. This is different to responses of participants in workshop three that found single-window interface easy to use. However, the responses of the majority of participants in workshop one is similar to that of workshop two and three who refer to the visibility, navigation and functionality of the Scratch tool to be fun, interesting and easy to use. It therefore seems that Scratch is a suitable instructional technology to use when teaching programming concepts and that the students adapt well to it. Scratch appears to be very accessible as a tool that enables the user to do many creative things very quickly.

This is similar to Price et al. (1983), stating that interactive graphics and animations of software visualizations will enhance the interface between the programming tool and the programmer. The interaction with Scratch seems to be a positive experience for the students regardless of their level of experience. This is similar to Nam et al. (2010) and Utting et al. (2013) who describe Scratch as an appropriate tool for novice programmers as it is easy and interesting to learn and aids in the development of problem solving skills. Scratch as an interactive animation and games programming setting, promotes active learning and is visually appealing.

5.2.3 ELIMINATION OF COMPILATION STEP

Responses from participants in the three workshops indicated that the elimination of the compilation step might appear to be helpful and that students will spend less time with their coding. However, from the empirical data it is recommended that the compilation step not be eliminated as it will negatively affect the student's ability to learn the programming concepts. With the result, students will be unable to identify their mistakes or syntax errors which may slow down the development of their problem solving skills. On the other hand, the responses of Respondents 1.6, 1.7, 1.8 and 1.10 are different in that they believe that the outcome of eliminating the compilation step will enhance the confidence level of students when they witness their program executing at the outset without any bugs or vague error messages.

It seems that it is essential for students to learn how to compile a program. A program needs to be tested, debugged and corrected. Eliminating the compilation step will limit the student's development of problem solving skills. This is similar to Salleh et al. (2013) stating that one of the greatest impediments for the novice programmers is the ability to compile and coordinate the different components of a program. Understanding the syntax of a programming language will teach them how to develop a program.

5.2.4 HELP SCREENS AND DEFAULT PARAMETERS

The responses from respondents in the three workshops are similar in that the help screens, default parameters and illuminating demonstrations will be helpful for students to learn and better understand the programming concepts. As a result, the programming skills of novice programmers will be enhanced. It may also assist students with understanding a program that is unfamiliar to them and even be helpful with games programming. In contrast, some of the responses from respondents in workshop one indicate that help screens and default parameters not useful as students will have no understanding on what they are doing. "Help screens should guide you in the right direction and not give you the correct answer or solution to the problem" (Respondent 1.1).

5.2.5 VISUAL FEEDBACK

The responses on visual feedback during code or script executing and troubleshooting are all similar from all three workshops as it points out immediately where mistakes were made. This will aid in the fixing of programming errors and allow students to the progress that was made in their programs. Students are given the opportunity to learn from their mistakes and are able to observe whether the program is executing logically according to the desired specifications. It therefore seems that visual feedback is very useful because students are able to see their progress and are able to better understand the abstract and complex concepts in programming. Using a visualization tool similar to Scratch enhances the students' comprehension of program execution. With the aid of the graphical form presented by the tool, users will be able to identify logical errors in their programs. It is similar to (Ala-Mutka, 2004) and Utting et al. (2013), stating visualization helps students to understand the learning of concepts. Program

visualization allows the student to be active as the author, rather than the passive observer of the instructor's animation

5.2.6 RUNNING OF INCORRECT PROGRAMS

Responses from all three workshops on the ability of incorrect program to still run despite the syntax or runtime errors are all similar in that it will have a negative influence on the development of student's understanding of the programming language. This will create confusion amongst students and may encourage students to ignore the rules of programming. With the result, it will limit the enhancement of their programming and problem solving skills. Students will be unaware of the logical errors in their code and will be unable to learn from their mistakes. From the empirical data it seems that incorrect programs should not run as students will not be able to troubleshoot and fix errors and will repeatedly make the same mistakes. This is different from Utting et al. (2013), stating that avoiding syntax errors with an instructional tool like Scratch, will free the user to focus on processes and concepts rather than focussing on details.

5.2.7 VARIABLES AS CONCRETE OBJECTS

For the reason that programming concepts are abstract in nature, respondents in workshop one indicated that it will be helpful having variables as concrete or visible objects. Variables as visible objects will enhance the student's comprehension on how variables are passed in a program, making it easier to understand the background work of a program. This is similar to responses in workshop two, indicating that variables as visible or concrete objects will assist students in using them correctly which will reduce the difficulty of writing programs. It appears that students will have a better grasp on understanding the workings of variables that are visible. This will result in accelerating their learning and understanding of abstract concepts.

5.2.8 IDEAS TO HELP FIRST YEARS

The empirical data from responses in workshops one and two recommends that a visual tool similar to Scratch or Alice be introduced to first year programming. This will greatly assist students in mastering the programming concepts. This is similar to Cooper et al. (2000) and Daly (2011) who found Alice to be a suitable programming language for novice programmers. The Alice environment eliminates the frustration and the focus on

the language syntax, and has proved to be enjoyable, promoting a positive attitude towards programming and improves the retention of programming students. As an introductory programming language for first year students of computing studies at universities, responses from participants in workshop one suggest that a language such as Java be introduced to novices as a first year programming language. This is different to Jenkins (2002) who states that the purpose of an introductory to programming course is to teach students how to program and not to teach them a particular programming language, for example Java. Respondents in workshops two and three are of the opinion that students should practise programming on a daily basis and be able to apply the mind-set of a programmer. Responses from participants suggested that more practical lab sessions be made available in the first year programming course.

From the participant responses it became clear that learning programming is a practice-based process. Respondents in workshop three were of the opinion that the forming of study groups will aid first year programming students in mastering the abstract concepts in programming. It seems that students should be encouraged to form learning communities by working in collaboration with their peers. The sharing of ideas and thoughts on programming concepts may result in making learning a lifelong experience.

The empirical data further suggests that a visual programming language will greatly contribute in novice programmers grasping the abstract programming concepts. This is similar to Lattu et al. (2000) and Agno-Balabat and Rojo (2013) proposing program visualization as an educational tool. Visualization as a helpful tool to novice programmers offers more transparency to the program and their execution. It will enable the possibility for students to grasp the logic of algorithms without understanding the actual code. Regular practise is also identified as being vital to enhance the programming and problem solving skills of students. This is similar to Yacob and Saman (2012) and Agno-Balabat and Rojo (2013) who suggest that students do a lot of practise in order to accelerate their development of good programming skills. This will help novice programmers to become experts in programming.

5.2.9 SCRATCH AS AN INSTRUCTIONAL TOOL

Responses from participants in all three workshops are similar in that Scratch as an instructional tool will assist novice programmers to better understand the programming concepts. The informative and fun nature of the tool will improve first year programming development and strengthen their program writing skills. Respondents in workshop three affirm that it is easy to adapt to the Scratch, which is dissimilar to responses in workshop one indicating that the visual tool may be challenging for students with prior access to computers. “For first time programmers, Scratch requires a lot of explanation” (Respondent 1.8).

It seems that Scratch is interactive and informative and it will enhance the problem solving skills of novice programmers. The tool allows students to be actively involved in the construction of their creative designs. Students with no prior knowledge of computers will be able to easily familiarise them with the tool. This is similar to Tangney et al. (2010), Nam et al. (2010) and Agno-Balabat and Rojo (2013), describing the Scratch tool as a highly potential first language for novice programmers and that it facilitates the development of problem solving skills. Using a visualization tool like Scratch will enhance the students’ understanding of program execution. It would also assist novice programmers in writing code and visually tracing it during troubleshooting. The Scratch tool introduces users to the use of objects (sprites) in programming and helped students to better understand looping conditions.

The following aspects came out strong in the literature analysis although were not specifically identified by the thematic analysis.

5.3. PROBLEM SOLVING SKILLS

An experienced programmer draws on many skills and much experience. Some of the required skills include problem solving ability and some mathematics underlying the process. “Research literature and practical experience of subject experts indicate that teaching programming to novices has proven challenging for both student and lecturer”

(Costello et al. ,2007). Problem solving skills are therefore fundamental in solving new problems.

It has been found that in order for programmers to develop competency, they need to have good problem solving skills and knowledge of a programming language. However, many first-time entry students come to university with mediocre or inadequate problem solving skills. According to Costello et al. (2007), research supports the fact that novices lack the meta-cognitive skills needed to become life-long learners and proficient programmers.

5.4 MOTIVATIONAL FACTORS

Students approach computing degrees with a variety of motivations. Some may have a genuine interest in the subject (intrinsic motivation). This is similar to Campbell (2013), stating that motivation is the major factor affecting the performance of novice programmers. Intrinsic motivation was found to be high for students with some experience. Students who struggle in programming see their degree as a means towards a lucrative career (extrinsic motivation). A third form of motivation is when students try to please their parents or family (social motivation).

The reason for many students to study computing is the possibility of gainful employment after their graduation. However, personal interest in the subject is on the other hand an important motivating factor (Tangney et al., 2010). The structure of engagement and a student-directed setting over an extended unstructured period, accessible after school, provides a strong context for changes in the perception of computing.

5.5 ATTITUDE

In the context of CPUT, first year DS1 students at the beginning of the 2011 academic year, feared that they would not be able to master the programming subject. A group of five students in the classroom expressed their concern that they had never been exposed to computers before arriving at university. They therefore requested that the

researcher, who was also their lecturer at that time, provide them with all the support necessary to successfully master the first year programming subject.

The negative attitude towards the programming subject was in the mind of students before studies even commenced. Students indicated that some of their peers who were repeating DS1 informed that the subject is very difficult to master. This is in line with Jenkins (2002) who states that programming acquires the reputation of being difficult because of the view that is passed on to new students by their predecessors. As a result students approach a course with an expectation that it will be difficult and are therefore not motivated to succeed.

IT lecturers at CPUT are currently teaching C++ as an introductory programming language to first year IT students. Visual Studio, an integrated development environment (IDE) from Microsoft (Figure 5.1 and Figure 5.2) is used to develop console applications.

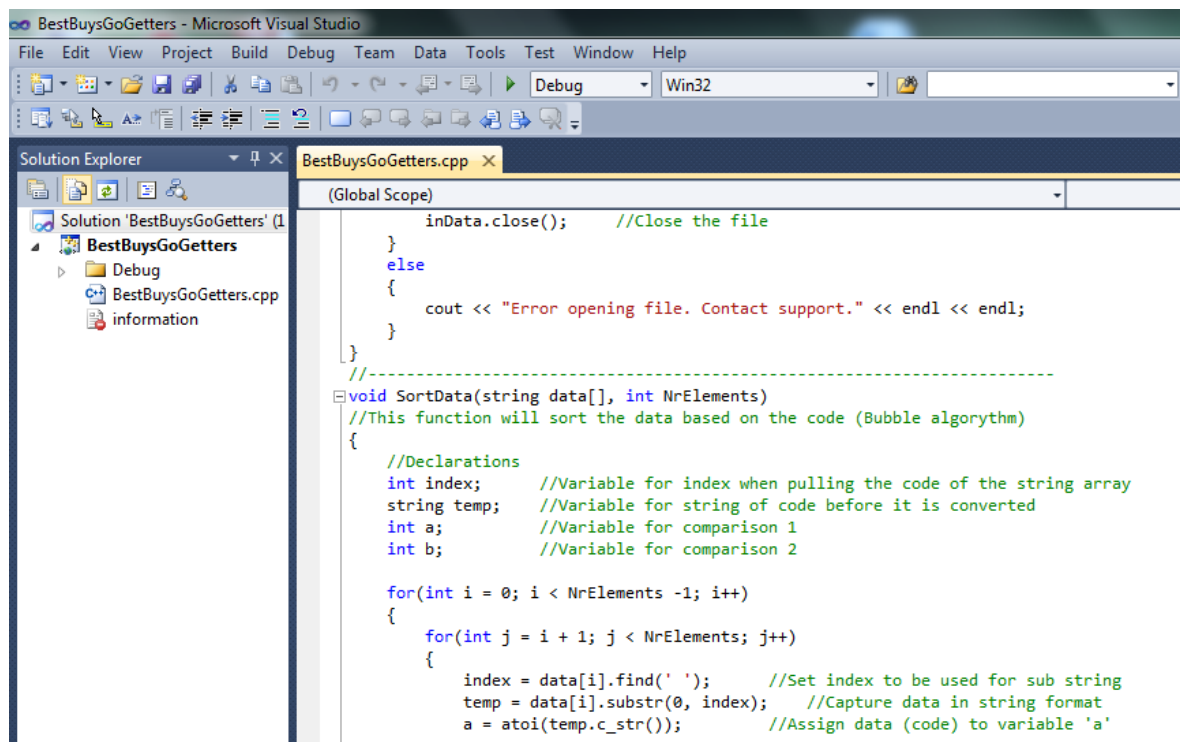
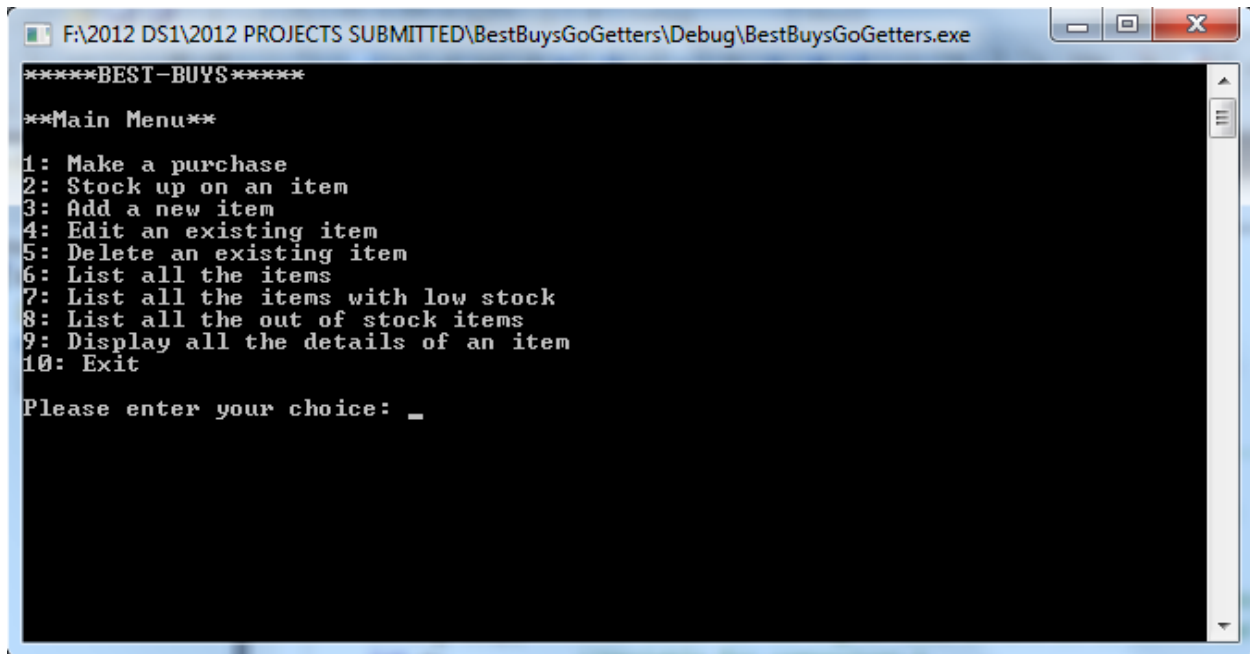


Figure 5.1 Screenshot of Visual Studio integrated development environment



```
*****BEST-BUYS*****  
  
**Main Menu**  
1: Make a purchase  
2: Stock up on an item  
3: Add a new item  
4: Edit an existing item  
5: Delete an existing item  
6: List all the items  
7: List all the items with low stock  
8: List all the out of stock items  
9: Display all the details of an item  
10: Exit  
  
Please enter your choice: _
```

Figure 5.2 Screenshot of C++ runtime environment

5.6 DIVERSE STUDENT BACKGROUNDS

Some students may find it difficult to adapt to a life of study at university. First time entry students at institutions of higher learning must learn how to take ownership of their own education. They may need the necessary guidance and support already at secondary school to help develop them into self-motivated and self-sufficient problem solvers.

In their discussion of teaching delivery issues and problems (Carter & Boyle, 2002) conclude that it is crucial that the expectations of first time entry students be aligned to that of a university at an early stage. Students arrive at university from a diverse range of backgrounds, with different expectations regarding their higher education. The situation at CPUT is critical because the bulk of its students come from poor and disadvantaged backgrounds which have strong implications for the teaching and learning of programming.

For most students university experience is new irrespective of prior experience. “Student expectations upon arrival at university and their level of preparedness for formal and logical reasoning cannot be ignored; also their ability to connect the practical skill of the subject to the abstract theory behind it must be developed” (Carter & Boyle, 2002).

Four different population groups (African, Coloured, Indian and White) were represented in the sample population for this study. Student participants were from different social, cultural, geographical and language backgrounds with the male participants (68%) dominant in total over female participants (32%). The diversity was evident in terms of educational standard and computer background. However, the differences were not specifically considered in this study and did not influence the findings in any way.

5.7 PRIOR KNOWLEDGE

There should be connection between subjects at secondary school and those offered to students in their first year of study at university level. During an educational research project, Moskal et al., (2000), observed in classrooms that students who have little or no prior programming experience are at risk for not succeeding in their first year programming course. It was also discovered that the majority of students leave computer science at the end of their first year. Factors contributing to this situation include a lack of prior computer experience, limited and poor mathematics preparation, poor self-efficacy and the new university environment.

“Students with no prior programming experience are disadvantaged in successfully completing their computer science degree” (Moskal et al., 2000). This result is correlated to the fact that universities are shifting from imperative programming languages similar to C and Pascal for novice programmers, to object-oriented languages like C++ and Java. Students now have to learn additional concepts which include class, object, information hiding, inheritance and polymorphism.

In the context of CPUT, the prior knowledge of programming and previous backgrounds in computers enables students to better perform low-level programming tasks, understanding code functionality and locating syntax errors faster. However, it was also evident that students with prior programming from school were lacking certain programming principles and concepts which include the struct data type, arrays and I/O operations on external data files.

5.8 TEACHING APPROACH

It is suggested by Tangney et al., (2010) that an innovative pedagogical approach needs to be explored for teaching programming to students in secondary schools. In appropriate scaffolding settings, pupils can be facilitated to learn where much of the learning is student initiated with peers and mentors being the sources of help.

5.9 KEY FINDINGS FROM THE STUDY

Responses to the research questions are provided, namely:

What are the tentative design principles of appropriate instructional technology that can be used to facilitate novice programmers' understanding of programming concepts?

- What are the challenges of technology-supported teaching of programming concepts?
- What are the contextual aspects that need to be considered for the design of an instructional technology intervention?
- How do novice programmers interact with instructional technologies during the programming process?
- How can instructional technology be used to engage students in learning and acquisition of programming skills?

5.9.1 What are the challenges of technology-supported teaching of programming concepts?

The biggest challenge for novice programmers is that they fail to understand the basic concepts in programming and then learning to apply them. The literature of Greyling et

al. (2006) revealed that the high incidence of unprepared students at higher education institutions in SA has an impact on introductory programming courses that rely on technological tools as a pedagogic concern. Acquiring and developing knowledge and skills about programming is a highly complex process which involves a variety of cognitive activities, and mental representations related to program design, program understanding, modifying and debugging.

Novices lack the necessary analytical skills resulting in the learning of programming to be highly complex for them. The situation at CPUT is critical because of students' lack of meta-cognitive skills to unlock key programming concepts. In the context of CPUT, students are at risk of not succeeding in their first year programming subject. This is due to little or no prior computer or programming experience, limited and poor mathematics preparation, poor self-efficacy and the adjustment from secondary school to university study.

Furthermore, the literature of Winslow (1996) suggests that the main source of difficulty is not the syntax and understanding of concepts but rather the planning of a basic program. Students are able to explain individual concepts, know the syntax and semantics of individual statements, but fail to apply it in their program or combine it into valid programs in solving a problem. In most cases students will confuse their syntax handling difficulties with problem solving difficulties which result in them getting frustrated and making them believe that they cannot do programming. In the context of CPUT, the researcher has noticed that when first year programming students are given a programming problem, they immediately try and write the language source code on the computer. Novice programmers spend little time in planning and testing code and often fail to apply the knowledge they have obtained correctly. The biggest challenge therefore does not appear the understanding of basic programming concepts but the ability to apply them correctly and the ability to organize structure and plan their thoughts.

5.9.2 What are the contextual aspects that need to be considered for the design of an instructional technology intervention?

The bulk of first-time entry students at CPUT come with mediocre or inadequate problem solving skills from poor schooling and disadvantaged backgrounds. This gap has had strong implications for the teaching and learning of programming at CPUT and throughput rates of the university. As mentioned in the problem statement, the majority of first-year IT students at CPUT are novice programmers and lack strong logic and reasoning skills that can facilitate their interpretation and applications of key programming concepts.

Currently, the student population at CPUT is very diverse, resulting in diversity in terms of computer skills and programming knowledge. The categories represented in the sample student population are African, Coloured, White and Indian. Students of the African demographic group had the highest percentage of enrolment for 2012 followed by Coloured, White and Indian. Overall, male participants succeeded in percentage, over the female participants in all three workshops.

As an IT lecturer at CPUT, the researcher has concluded that some first year students may find it difficult to adapt to a life of study at university. For most students university experience is new irrespective of prior experience. The reviewed literature emphasized the importance that the expectations of first time entry students be aligned to that of university at an early stage. Students arrive at university from a diverse range of backgrounds with different expectations regarding their higher education. Furthermore, the literature was suggestive that student expectations on arrival at university and their level of preparedness for logical reasoning should not be ignored; and that their ability to connect the practical skill of programming to the abstract theory behind it must be developed.

5.9.3 How do novice programmers interact with instructional technologies during the programming process?

The literature review revealed that students will be able to master the learning of programming skills with the support of suitable teaching strategies and tools. It was further highlighted by the literature that one way to assist novice programmers is to provide computerized assistants, for example software visualization tools that use interactive computer graphics, graphic design and animations to enhance the interface between computer programmers and their programs. The reviewed literature suggested that innovative technologies can be integrated into the classroom to enhance teaching and learning. Instructors will be able to take advantage of technology to enrich the educational experience of their students.

A well-researched and innovative learning tool called Scratch, that offers a visual programming environment, was used in this study. Scratch was used as an example of instructional technology as it was designed to contribute to the understanding of basic programming concepts. The findings of this study found the visibility and graphics of the tool to be interesting and the interface easy to navigate with. Furthermore, the ease of use and single-window interface makes the tool easy to use and programming interesting.

The study findings strongly suggested that the compilation step not be eliminated as it will negatively influence the ability of the student to learn the concepts and to identify their mistakes and syntax errors. In terms of help screens, default parameters, and illuminating demonstrations for commands, the findings have shown that it will be helpful for novices to better learn and understand the programming concepts. Moreover, it will improve the programming skills of students and help create an interest in the programming.

The reviewed literature emphasized the importance of the use of visualizations in computer science education because they are important for understanding and learning abstract and complex concepts in this field. The literature further states that program

visualization tools might be a promising aid for novice programmers learning to program. Students benefit by having a greater understanding of the programming concepts and to assimilate the new concepts. The use of a visual programming language will provide students with a visual vocabulary which they can grasp and apply instantaneously. The findings of this study found visual feedback during code or script executing and troubleshooting to be very useful because the student will be able to see the progress made in the program. Visual feedback allows the student to see if a program is executing logically according to the desired specifications. Students will know immediately where mistakes were made, allowing them to learn from their mistakes and make corrections rapidly.

According to the findings of this study, the ability of an incorrect program to still run by eliminating syntax or runtime errors will have a negative impact on the development of students. It will limit their programming skills and problem-solving ability and encourage students to ignore the coding rules of the programming language. The findings of this study also suggested that having variables as concrete or visible objects is quite useful. Programming concepts are mostly abstract and understanding how variables are passed in a program can sometimes be a problem. Visual feed will help students to use variables correctly and also reduce the difficulty of writing programs.

5.9.4 How can instructional technology be used to engage students in learning and acquisition of programming skills?

The researcher found it appropriate to engage in an educational design research methodology to facilitate the development of an intervention to engage students in learning and acquisition of programming skills. Design research as a research strategy facilitates the development of an intervention to explore alternative pedagogies for teaching and learning programming at CPUT. Insights and contributions were provided by the approach to enhance the teaching and learning of abstract programming concepts to novice programmers at CPUT. The recommendations produced by this study highlight the principles that need to be considered for the design of an appropriate instructional technology tool as an intervention.

Designing and developing software solutions in the IT field require specific skills. It is important to understand the problems that novice programmers face when learning programming concepts, before an intervention can be designed. Therefore the researcher in his exploratory study used an existing intervention to establish how novice programmers respond to this example of instructional technology. This study presented to the researcher an improved understanding of the challenges experienced by novice programmers when learning programming concepts. The educational design research approach applied by the researcher, provided insights to aid in generating mechanisms in addressing the problem of grasping and understanding abstract programming concepts

5.10 CHAPTER SUMMARY

This chapter discussed the findings of the study relative to the literature review and the objectives of the study. The discussion highlighted areas where the findings aligned with related literature and where they did not. Areas that required future research were identified.

The next chapter will discuss the overall conclusions of the study and provide recommendations and potential areas for future research.

CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS

6.1 INTRODUCTION

This chapter concludes the study, highlights limitations, and makes recommendations for potential future research. The conclusions for this study are drawn from the overall response statements as presented in Chapter five.

6.2 SUMMARY OF CHAPTERS

The dissertation is structured as follows:

Chapter One: Introduction

This chapter outlines the research problem and sets out the objectives of the research.

Chapter Two: Literature review

In this chapter the literature is reviewed relating to skills and knowledge required to master programming concepts and identifies existing and other technology solutions to contribute to the teaching of students. The aim of the reviewed literature is to reveal what has been done in this field of study by reviewing existing scholarship. This chapter also explores how other scholars have theorized and conceptualized on the issues, what they have found empirically and the instruments that they used and to what effect.

Chapter Three: Research methodology

This chapter discusses the tools and methods used in gathering meaningful data and provide an overview on the methodology of the study. Given the literature review presented, the focus is on how the objectives of the study will be achieved and what research methods are best suited for the study. A detailed description of research design and questionnaire design is presented and why it was necessary to employ sampling selection and techniques to respond to the research questions. The research

protocol outlines a detail set of instructions and procedures that will be followed in conducting the study and the collection of data.

Chapter Four: Data analysis and findings

This chapter outlines the analysis of data that was gathered including observations of the researcher. The focus is on the results obtained after the analysis of the data in response to the problems identified in Chapter One. The responses from research instruments are presented. The analysis of the qualitative data proceeded into the findings of the research.

Chapter Five: Discussions of findings

Findings relative to the objectives of the study and related literature are discussed and interpreted and linked to the reviewed literature. The discussions highlight the areas where the findings are aligned to the related literature and where it is not. Chapter Five draws the conclusion on these findings which includes the analysis on these findings.

Chapter Six: Conclusions and recommendations

The conclusions and recommendations are drawn based upon data analysed linking it to objectives of the subject under investigation.

6.3 SUMMARY OF FINDINGS

A summary of findings relative to the aims and objectives of this study is found in Table 6.1.

Table 6.1 Summary of Workshop findings (copied from Chapter 4)

		Workshop 1	Workshop 2	Workshop 3	
Demographics	Males	79%	67%	56%	
	Females	21%	33%	44%	
	Indian	5%	-	-	
	White	16%	-	-	
	African	37%	93%	94%	
	Coloured	42%	7%	6%	
Academic Performance		Average Marks			
	Semester 1	65% (2011)	42% (2012)	35% (2011)	39% (2012)
	Semester 2	89% (2011)	62% (2012)	37% (2011)	75% (2012)
Key Themes		<ul style="list-style-type: none"> • Prior programming experience will enhance the preparedness of students • Software development and programming languages pre-acquaintance are useful • Practice is crucial to understand and apply programming concepts • Interface adaption to single-user interface of Scratch is challenging 	<ul style="list-style-type: none"> • Practice is essential to learn and understand concepts • Compilation step not to be eliminated • Help screens and default parameters very helpful • Visual feedback during execution very helpful • Incorrect program should not run • Variables should be presented as concrete or visual objects 	<ul style="list-style-type: none"> • Programming becomes easier with practice • Prior programming knowledge essential • Elimination of compilation step very helpful • Help screens and default parameters improve programming skills • Visual feedback assist with troubleshooting • Incorrect program still running creates confusion 	

Student demographic details, including gender and ethnicity were extracted from the CPUT student database. The diverse student population group consisting of Indian, White, African and Coloured are represented in the sample population with African student having the highest percentage (72%) of participation. In total, male participants (68%) were found to be dominant in this exploratory study.

Majority of the workshop one participants had prior programming experience or Java as a subject at high school level, compared to participants of workshops two and three with little or no prior knowledge. Hence, the noticeable difference in academic performance of the programming subject DS1, in their first year of study at university. Academic performance of participants in workshop three had significantly increased in semester two of 2012 (75%) compared to their semester two results of 2011 (37%), after repeating DS1 for a second time.

The researcher has identified certain key themes that emerged from the responses of participants in the three workshops. Having prior programming knowledge is essential as it will enhance the preparedness of students in their first year of study at university. Practice is crucial and will improve the ability of the student to better understand and apply programming concepts. Visualization was identified as being helpful as it advances the learning of abstract and complex concepts and assists with troubleshooting.

6.4 RESEARCHED QUESTIONS REVISED

One research question and four sub-questions motivated and guided the exploratory study. The study responded to the research questions listed in Table 6.2 which defined the whole research process and gave guidance to the arguments and inquiries of the researcher.

Table 6.2 Research questions revisited

Research Question		
What are the tentative design principles of appropriate instructional technology that can be used to facilitate novice programmers' understanding of programming concepts?		
Research Sub-Questions	Objectives	Findings
What are the challenges of technology-supported teaching of programming concepts?	Identify the challenges experienced by novice programmers to learn the programming concepts	Novice programmers fail to apply syntaxes and basic concepts in solving problems; spend too little time planning
What are the contextual aspects that need to be considered for the design of an instructional technology intervention?	Profile the novice programmer for a specific context; methods	Students from diverse backgrounds and different expectations; expectations of first years to be aligned to that of university
How do novice programmers interact with instructional technologies during the programming process?	Establish the novice programmers' responses to the instructional technology during the programming process	Instructional technology experienced as to be very accessible and enabling creative things; visualization enhances student comprehension of program execution
How can instructional technology be used to engage students in learning and acquisition of programming skills?	Develop the tentative design principles of an instructional technology intervention	Visual programming tool allows students to be more active in learning; students actively involved in construction of their creative designs

6.5 VALIDITY OF RESEARCH

Yin (1994) suggests that a high quality case study is characterized by rigorous thinking, sufficient presentation of evidence to reach appropriate conclusions and careful considerations of alternative explanation of evidence. It is further suggested that multiple sources of evidence will ensure constant validity. The data collection workshops were well planned and constructed and included DS1 students of the IT

Department. Instruments that were used resulted in a high degree of reliability, validity and findings of the research. The use of field notes and memos helped the researcher to determine if the validity of the findings has been affected in any way.

6.6 RECOMMENDATIONS

The primary goal of the study was to investigate how instructional technology can be used to teach programming to novices at a UoT. This study found that students must learn to understand the full SDLC in order to systematically attack problems before even translating their solution into a working program. In order to master the learning of programming skills, suitable teaching strategies and tools must be available to support the learning process. Learning programming can be a complex task; therefore educators have to carefully design their teaching approaches. Good programming examples that include games should be used to practice programming with the aim of enhancing problem solving abilities. Research has found that students are motivated using games as a useful learning tool. Play could influence the development of visualization, experimentation and creativity.

The effective support provided to students by educators will help restore the confidence of students in their abilities to program. Educators also need to give students the opportunity to be more active in their learning. Students should be encouraged to collaborate with other students and form learning communities. The sharing of ideas and thoughts will aid first year programming students in mastering the abstract concepts of programming and may result in making learning a lifelong experience. Novice programmers can also be assisted by providing computerized assistants created for them. The engagement of students in active thinking is often driven by questions from instructors. From the exploratory study it was observed that a tool similar to Scratch should be considered to improve the programming learning performance of students. This coincides with reviewed literature that using a visualized tool in the classroom is appropriate for novice programmers and will facilitate the development of problem solving skills.

The reviewed literature suggests that educators should focus more on the combination and use of features underlying the issues of basic program design. An instructional approach must be thoroughly designed to issues on the programming course, and how aiding materials can be introduced into education. Lecturers must recognize the fact that students have individual learning needs and that different teaching strategies need to be investigated and applied in the classroom. In the traditional lecture method educators are active while students remain passive. Students may be kept attentive by making use of quizzes. Innovative technologies can be integrated into the classroom to enhance the teaching and learning process. Instructors will be able to take advantage of technology to enrich the educational experience of the students. The expectations of first time entry programming students should be aligned to that of the university at an early stage. Hence, their level of preparedness for formal and logical reasoning cannot be ignored

The aid of visualization will allow programming content and concepts to be represented structurally. Software visualizations use interactive computer graphics, graphic design and animations that will enhance the interface between computer programmers and the program. Visualization technology will be of little educational value unless it engages students in active learning activities. Reviewed literature states that program visualization tools might be a promising aid for novices learning to program. Students will enjoy the benefit of having a greater understanding of programming concepts and to better assimilate new concepts. A visual programming language will provide students with a visual vocabulary which they can grasp and apply instantaneously.

A programming language must be seen as a tool used to implement a solution. The need exists for specialized program development environments to be developed. The language of choice for the first programming language should provide an instructional environment for the student to develop skills in higher order thinking and problem solving. Access to technology should be given to students at a level that will benefit them. Students will be able to develop their skills and willingness to participate irrespective of their ethnic and educational background. Research has found that a

pedagogical program development environment will have a positive effect on the perceptions of novice programmers learning to program. Positive perceptions, including feelings of achievement and learning, could alleviate some difficulties experienced during the learning process.

A basic background in mathematics and the English language is recommended before students commence on a study in IT. Reviewed literature states that a mathematics background puts the focus on logic and problem solving. A positive relationship should therefore exist between mathematical ability and successful student programmers. It is further stated in the reviewed literature that the general intelligence and mathematics science abilities seems to be related to success when it comes to learning to program. Prior programming experience is also highly important for the reason that the higher incidence of under prepared students rely on the use of technological tools a pedagogical concern. A connection should exist between subjects at secondary school and those offered in their first year of study at a university. The reviewed literature suggests that an innovative pedagogical approach needs to be explored to teach programming at high school level. It is further recommended that a technology solution be designed and implemented to enhance the teaching and learning of programming.

6.7 FURTHER RESEARCH

The insights gained from this research study will be used to design and implement a technological solution to enhance teaching and learning of programming concepts. A thorough literature review will be conducted to establish the concepts relevant for further research. The findings of this research will be used to develop a concept solution that can be used to teach programming concepts to novice programmers.

The new programming concept will be taught to the new first year IT students and their comprehension of the model will be assessed and analysed. Students will be introduced to the concept technology solution and their comprehension of the programming concepts will be observed after having used the solution. Feedback will be obtained from first year IT lecturers and students. The concept solution will be refined and

reintroduced to lecturers and students after which the results will be analysed and interpreted. The final results and proposed technology solution will be presented.

6.8 CONCLUSION

In summary there are many factors that contribute to the challenges experienced by novice programmers which include the lack of prior computer or programming knowledge, poor mathematics and problem solving skills, and the difficulty to adapt to university life. The biggest challenge is not merely in the understanding of the basic programming concepts but in the ability to apply it when constructing a complete working program.

The outcomes of the study indicate that unprepared students have a negative impact on the delivery of programming courses offered at university. Many first-time entry students come to the university with mediocre or inadequate problem solving skills. Students with little or no prior programming experience and poor mathematics preparation are at risk of failing their first year of programming. The reviewed literature concludes that students with no prior programming experience are disadvantaged in successfully completing their IT qualification. It was established that novice programmers lack analytical skills whereas experienced programmers draw on their skills which include problem solving ability and mathematics. Problem solving skills were therefore found to be fundamental in solving new problems.

The study also revealed that students might not have difficulty in understanding the syntaxes of the programming language but merely fail to perform proper planning. Due to little time spent on planning, students jump straight into coding and therefore fail to apply their knowledge. Prior knowledge of programming and previous background in computers enables students to better perform low-level programming tasks. Access to computers is essential in learning programming concepts in the first year of study.

The study also established that suitable strategies and tools are needed to master abstract concepts. Innovative technologies in the classroom will enhance the teaching

and learning of programming. Novice programmers show more interest in programming when they are using a tool with a visual environment that offers user-friendly graphics and visibility. With visual feedback students have a better understanding and are able to see how a program executes logically. It was further established that visualization enhances the students' comprehension of program execution. Interactive graphics and animations of software visualizations enhance the interface between the tool and the programmer. From the study it was observed that the use of a self-explanatory aid similar to Scratch, supported by a strong visual environment, is sufficient to introduce programming to those with no previous programming experience. However, the tool does not provide the opportunity to experience errors such as mismatched braces and missing semicolons.

In conclusion, it emerged from the exploratory study that prior programming knowledge and access to computers is essential for novice programmers to learn and understand programming concepts. Further research is necessary to design and implement a technological solution to enhance the teaching and learning of abstract programming concepts.

As a researcher it was observed that students with prior programming experience had the ability to grasp programming concepts immediately and with ease. This was despite the fact that they had no experience with the instructional tool Scratch. As discovered in the literacy review and findings, prior programming experience will enhance the preparedness of students in their first year of IT studies at a university. An interactive visualization software tool similar to Scratch should be considered as it will enhance the learning performance and comprehension of abstract programming concepts by students.

REFERENCES

- Agno-Balabat, A. C. G., & Rojo, J. N. N. 2013. A Program Visualization Approach in Developing an Interactive Simulation of Java Programs for Novice Programmers. *Mindanao Journal of Science and Technology*, 10(1).
- Ala-Mutka, K. 2004. Problems in learning and teaching programming-a literature study for developing visualizations in the Codewitz-Minerva project. http://www.cs.tut.fi/~edge/literature_study.pdf. [Accessed on June 23, 2014]
- Amory, A. 2010. Learning to play games or playing games to learn? A health education case study with Soweto teenagers. *Australasian Journal of Educational Technology*, 26(6):810-829.
- Amory, A., Naicker, K., Vincent, J. & Adams, C. 1999. The use of computer games as an educational tool: 1. Identification of appropriate game types and game elements. *British Journal of Educational Technology*, 30:311-322.
- Atachiants, R., Gregg, D., Jarvis, K., & Doherty, G. 2014, April. Design considerations for parallel performance tools. In *Proceedings of the 32nd annual ACM conference on Human factors in computing systems* (pp. 2501-2510). ACM.
- Attride-Stirling, J. 2001. Thematic networks: an analytic tool for qualitative research. *Qualitative research*, 1(3):385-405.
- Bowen, G. A. 2005. Preparing a qualitative research-based dissertation: Lessons learned. *The Qualitative Report*, 10(2):208-222. Retrieved from <http://www.nova.edu/ssss/QR/QR10-2/bowen.pdf> [Accessed on August 21, 2013]
- Bryman, A. 2006. Integrating quantitative and qualitative research: how is it done? *Qualitative research*, 6(1):97-113.
- Butler, M., & Morgan, M. 2007. Learning challenges faced by novice programming students studying high level and low feedback concepts. In *Proceedings of Ascilite* (pp. 99-107). Singapore, 2-5 December 2007.
- Byrne, P. & Lyons, G. 2001. The effect of student attributes on success in programming. *Proceedings of the 6th annual conference on Innovation and technology in computer science education*. Canterbury, Kent, United Kingdom, June 25-27, 2001
- Campbell, V. 2013. A model for systematically investigating relationships between variables that affect the performance of novice programmers.
- Cape Peninsula University of Technology. 2013. IATUL Conference 2013. <http://active.cput.ac.za/IATUL2013/>. Cape Town, South Africa, 14-18 April 2013

Carter, J., & Boyle, R. 2002. Teaching delivery issues: Lessons from computer science. *Journal of Information Technology Education*, 1(2):65-90.

Chan Mow I. T. 2008 Issues and difficulties in teaching Novice computer programming. Proceedings of CISSE 2007. International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering. University of Bridgeport. December 3-12, 2007

Clear, T., Edwards, J., Lister, R., Simon, B., Thompson, E. & Whalley, J. 2008. The teaching of novice computer programmers: bringing the scholarly-research approach to Australia. Proceedings of the Tenth Australasian Computing Education Conference (ACE 2008). Wollongong, Australia, Jan 22-25, 2008

Cooper, S., Dann, W., & Pausch, R. 2000. Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges*, 15(5):107-116).

Costelloe, E., Sherry, E., & Magee, P. 2007. Determining Areas of Weakness in Introductory Programming as a Foundation for Reusable Learning Objects. *Electronic Journal of e-Learning*, 5(1):21-30.

Daly, T. 2011. Minimizing to maximize: an initial attempt at teaching introductory programming using Alice. *Journal of Computing Sciences in Colleges*. 26(5):23-30

Dehnadi, S. 2009. *A cognitive study of learning to program in introductory programming courses* (Doctoral dissertation, Middlesex University).

Derus, S. R. M., & Ali, A. Z. M. 2012. Difficulties in learning Programming: Views of students, 74-78. In *1st International Conference on Current Issues in Education (ICCIE2012)*. Yogyakarta, Indonesia, September 15 – 16, 2012.

Dos Reis F.L. & Martins, A. E. 2008. E-Learning Methodology: The debate forums.

Fielding, N. & Schreier, M. 2001. Introduction: On the Compatibility between Qualitative and Quantitative Research Methods. Forum Qualitative Sozialforschung / Forum: *Qualitative Social Research*, 2(1), Art. 4.

Fink, A. S. 2000. The role of the researcher in the qualitative research process. A potential barrier to archiving qualitative data. In Forum Qualitative Sozialforschung/Forum: *Qualitative Social Research*, 1(3).

Fix, V., Wiedenbeck, S., & Scholtz, J. 1993. Mental representations of programs by novices and experts. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems* (pp. 74-79). ACM.

Garner, S. 2003. Learning Resources and Tools to Aid Novices Learn Programming. *Informing Science & Information Technology Education Joint Conference (INSITE)*. Pori, Finland, 24-27 June 2003.

Gijselaers, W. H. 1996. Connecting problem-based practices with educational theory. In *Bringing Problem-based Learning to Higher Education: Theory and Practice*. R.J. Menges and M.D. Svinicki (Eds), 68:13-21.

Goosen, L., Mentz, E., & Nieuwoudt, H. 2007. Choosing the “Best” Programming Language?!. *Computer Science and IT Education Conference (CSITEd 2007)*. November 16-18, 2007, Mauritius. Informing Science Press:269-282.

Greyling F.C &Wentzel A. 2007. Humanising education through technology: creating social presence in large classes. *South African Journal of Higher Education*, 22(4):654-667.

Greyling, J. H., Cilliers, C. B., & Calitz, A. P. 2006. B#: The Development and Assessment of an Iconic Programming Tool for Novice Programmers. In *Information Technology Based Higher Education and Training, 2006. ITHET'06. 7th International Conference on* (pp. 367-375). IEEE. July 10-13, 2006. Ultimo, New South Wales.

Guo, H. 2006. Visual Material and Learning Aids in Teaching Object Oriented Programming, In *Proceedings of the 7th Annual ICS HE Academy Conference*, Dublin, Ireland. August 29-31, 2006:64-69.

Harvey, B. & Mönig, J. 2010. Bringing “No ceiling” to scratch: Can one language serve kids and computer scientists. *Proceedings of Constructionism 2010 Conference*. pp. 1-10. Paris, France: Constructionism, August 16 – 21, 2010.

Hongwarittorn, N. & Krairit, D. 2010. Effects of program visualization (Jeliot3) on students' performance and attitudes towards Java programming. Paper presented at the spring 8th International conference on Computing, Communication and Control Technologies, Orlando, Florida, USA. Retrieved from http://www.iiis.org/CDs2010/CD2010IMC/CCCT_2010/PapersPdf/TA750PM.pdf [Accessed on April 23, 2010]

Ismail, M. N., Azilah, N. G. A. H., Naufal, U. M. A. R., & Kelantan, U. T. M. C. 2010. Instructional strategy in the teaching of computer programming: a need assessment analyses. *The Turkish Online Journal of Educational Technology (TOJET)*, 9(2):569–571.

Jackson, P. 1983. *Principles and problems of participant observation*. *Geografiska Annaler. Series B. Human Geography*, 65(1):39-46.

Jameson, T. L. 2008. *Gods and knickknacks: the American adoption of Asian religious items*. (Doctoral Dissertation)

- Jenkins, T. 2002. On the difficulty of learning to program. In Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences 4:53-58, August 2002. Loughborough, United Kingdom.
- Jenkins, T., & Davy, J., 2000. Dealing with diversity in introductory programming, LTSN-ICS 1st Annual Conference, Heriot-Watt University, Edinburgh. August 23-25, 2000.
- Jung, I. 2005. ICT-pedagogy integration in teacher training: Application cases worldwide. *Educational Technology & Society*, 8(2):94-101.
- Kanaparan, G., Cullen, R., & Mason, D. 2013. Self-Efficacy and Engagement as Predictors of Student Programming Performance. In Proceedings of 17th Pacific Asia Conference on Information Systems PACIS, pp. 282-282. Jeju Island, Korea, June 18-22, 2013.
- Kelleher, C., & Pausch, R. 2007. Using storytelling to motivate programming. *Communications of the ACM*, 50(7):58-64.
- Kluge S. & Riley L. 2008. Teaching in virtual worlds: Opportunities and challenges. *Setting Knowledge Free: The Journal of Issues in Informing Science and Information Technology*, 5:127-135.
- Lattu, M., Tarhio, J. & Meisalo, V. 2000. How a visualization tool can be used – evaluating a tool in a research and development project. 12th Workshop of the Psychology of Programming Interest Group, Cozenza, Italy.
- Laurier, E. 2010. Participant Observation. In N Clifford, S French & G Valentine (eds), *Key Methods in Geography*. 2nd edn, Sage Publications, London: 116-130.
- Li, E. Y. 1990. Software testing in a system development process: A life cycle perspective. *Journal of Systems Management*, 41(8):23-31.
- Linden, T. & Lederman, R. 2011. Creating visualizations from Multimedia building blocks: A simple approach to teaching programming concepts. Proceedings of the Information Systems Educators Conference (ISECON), November 3-6, 2011. Wilmington, USA.
- Linn, M.C., & Dalbey, J. 1989. Cognitive consequences of programming instruction. In E. Soloway & J.C. Spohrer (eds.), *Studying the Novice Programmer*. Hillsdale, NJ: Erlbaum.
- Loh, C. S., & Sheng, Y. 2013. Measuring the (dis-) similarity between expert and novice behaviors as serious games analytics. *Education and Information Technologies*, 1-15.

- McIver, L. 2002 Evaluating languages and environments for novice programmers. J. Kujis, L. Baldwin & R Scoble (eds). *Fourteenth Annual Workshop of the Psychology of Programming Interest Group (PPIG 2002)*, Brunel University, Middlesex, UK. 2002. June 18-21, 2002:100-110.
- Moons, J., & Backer C. D. 2009. Rationale behind the design of the EduVisor software visualization component. *Electronic Notes in Theoretical Computer Science*, 224, 57-65
- Moons, J., & Backer C. D. 2012. The design and pilot evaluation of an interactive learning environment for introductory programming influenced by cognitive load theory and constructivism. *Computers & Education*, 60(1), 368–384.
- Moreno, A., & Joy, M. S. 2007. Jeliot 3 in a demanding educational setting. *Electronic Notes in Theoretical Computer Science*, 178, 51-59.
- Moreno, A., Myller, N., Sutinen, E. & Ben-Ari, M. 2004. Visualizing programs with Jeliot3: Proceedings of the International Working Conference on Advanced Visual Interfaces AVI, Gallipoli (Lecce), Italy, ACM Press. May 25-28, 2004.
- Moskal, B., Lurie, D., & Cooper, S. 2004. Evaluating the effectiveness of a new instructional approach. In *ACM SIGCSE Bulletin* 36(1):75-79.
- Nam, D., Kim, Y., & Lee, T. 2010. The effects of scaffolding-based courseware for the Scratch programming learning on student problem solving skill. 18th International Conference on Computers in Education, ICCE 2010. Putrajaya, Malaysia. November 29 - December 3, 2010.
- Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., & Velázquez-Iturbide, J. Á. 2002. Exploring the role of visualization and engagement in computer science education. In *ACM SIGCSE Bulletin* 35(2):131-152.
- Oliver, R., & Herrington, J. 2003. Exploring technology-mediated learning from a pedagogical perspective. *Interactive Learning Environments*, 11(2):111-126.
- Parry, K. W. 1998. Grounded theory and social process: A new direction for leadership research. *The Leadership Quarterly*, 9(1):85-105.
- Patalong, S. 2003. Using the virtual learning environment WebCT to enhance information skills teaching at Coventry University. *Library review*, 52(3):103-110.
- Patton, M. Q. 1990. *Qualitative evaluation and research methods (2nd ed.)*. Newbury Park, CA: Sage Publications.
- Pears, A., Seidman, S. Mlmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M. & Paterson, J. 2007, A survey of literature on the teaching of introductory programming. Proceeding of the ITiCSE-WGR '07 Working group reports on ITiCSE on Innovation and technology in computer science education. New York, USA. December 4, 2007:204-223

Perkins, D.N., Hancock, C., Hobbs, R., Martin, F. & Simmons, R. 1989. Conditions of learning in novice programmers. *Journal of Educational Computing Research*, 2(1):37-55.

Piccoli, G., Ahmad, R., & Ives, B. 2001. Web-based virtual learning environments: A research framework and a preliminary assessment of effectiveness in basic IT skills training. *MIS quarterly*, 25(4):401-426.

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

Plomp, T. 2009. Educational design research: An introduction. *An introduction to educational design research*, 9-35.

Price, B.A., Small, I.S. & Baecker, R.M. 1983. A Principled Taxonomy of Software Visualisation. *Journal of Visual Languages and Computing*. 211-266

Rajala, T., Laakso, M., Kaila, E. & Salakoski, T. 2008. Effectiveness of program visualization: A case study with the VILLE Tool. *Journal of Information Technology Education: Innovation in Practice*, 7:15-32.

Reeves, T., Baxter, P., & Jordan, C. 2002. Teaching computing courses-computer literacy, business microcomputer applications, and introduction to programming online utilizing WebCT. *Journal of Computing Sciences in Colleges*, 18(1):290-300.

Robins A., Rountree J.& Rountree N. 2003 Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 3(2):137-172.

Rodrigo, M. M. T., Baker, R. S., Jadud, M. C., Amarra, A. C. M., Dy, T., Espejo-Lahoz, M. B. V., & Tabanao, E. S. 2009. Affective and behavioral predictors of novice programmer achievement. *ACM SIGCSE Bulletin*, 41(3):156-160.

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

Romero, P., Good, J., Robertson, J., du Boulay, B., Reid, H., & Howland, K. 2007. Embodied interaction in authoring environments. In *Second International Workshop on Physicality* (p. 43). University of Lancaster, UK, September 3 – 7, 2007

Rowe, G. & Thorburn, G., 2000, VINCE – An on-line tutorial tool for teaching introductory programming. *British Journal of Educational Technology*, 31(4):359-369.

Rowley, J. 2002. Using case studies in research. *Management research news*, 25(1):16-27.

Saeli, M., Perrenet, J., Jochems, W. M., & Zwaneveld, B. 2011. Teaching programming in secondary school: a pedagogical content knowledge perspective. *Informatics in Education - An International Journal*, Volume 10, No 1, 73-88.

Salleh, S.M., Shukur, Z. & Judi, H.M. 2013. Analysis of Research in Programming Teaching Tools: An Initial Review. Proceedings of the 13th International Educational Technology Conference IETC-2013, Kuala Lumpur, Malaysia, 13-15 May 2013:141-148.

Savill-Smith, C., Attewell, J., & Stead, G. 2006. Mobile Learning in Practice: Piloting a Mobile Learning Teacher's Toolkit in Further Education Colleges. Learning and Skills Network (Great Britain), 2006

Scott, E., Weimann, P., & van der Merwe, N. 2011. Reflections on the Role of the Lecturer as Teacher, Researcher and Mentor in a Project-Based Approach for IS/IT Majors at Three Different Academic Institutions. In Proceedings of the 5th European Conference on Information Management and Evaluation, Università Dell'Insubria, Como, Italy, 8-9 September 2011 (p. 446). Academic Conferences Limited.

Sivula, K. 2005. A qualitative case study on the use of Jeliot 3. (Unpublished master's thesis). University of Joensuu, Department of Computer Science, Finland. Retrieved from ftp.cs.joensuu.fi/pub/Theses/2005_MSc_Sivula_Kimmo.pdf [Accessed on May 13, 2010]

Smith, P. A., & Webb, G. I. 2000. The efficacy of a low-level program visualization tool for teaching programming concepts to novice C programmers. *Journal of Educational Computing Research*, 22(2):187-216.

Soloway, E., & Spohrer, J.C. (Eds.). 1989. *Studying the novice programmer*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Sorva, J. 2010. Reflections on threshold concepts in computer programming and beyond. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (pp. 21-30). ACM.

Sorva, J., Karavirta, V. & Malmi, L. 2013. A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education* (TOCE). Volume 13. Issue 2. New York, USA. June 2013

Tangney, B., Oldham, E., Conneely, C., Barrett, S., & Lawlor, J. 2010. Pedagogy and processes for a computer programming outreach workshop—The bridge to college model. *IEEE Transactions on Education*, Volume 53, No. 1, pp. 53-60. February 2010

Tekdal, M. 2013. The Effect of an Example-Based Dynamic Program Visualization Environment on Students' Programming Skills. *Educational Technology & Society*, 16(3):400–410.

University of Kwazulu Natal. n.d. UKZNOnline. <http://www.ukzn.ac.za/ukznonline/>. [Accessed on May 01, 2014].

University of South Africa. n.d. <http://www.unisa.ac.za>. [Accessed on September 23, 2013].

Utting, I., Tew, A.E., McCracken, M., Thomas, L. & Bouvier, D. 2013. A fresh look at novice programmers' performance and their teachers' expectations. Proceedings of the ITiCSE working group reports conference on Innovation and technology in computer science education-working group reports. Innovation and Technology in Computer Science Education conference, 2013. Canterbury, United Kingdom, July 01-03, 2013.

Van Arsdale, J. 2010. Technology, peer collaboration, and ZPD in the dual language classroom (Loyola University Chicago)

Van den Berg M. & Aucamp F. 2007. A practical look at results from two mobile learning pilots in South Africa. IST Africa Conference, Maputo, Mozambique, May 9-11, 2007, pp. 12.

Vogts, D., Calitz, A. P., & Greyling, J. H. 2010. The effects of professional and pedagogical program development environments on novice programmer perceptions. *South African Computer Journal*, 45:53-58.

Welsh, E. T., Wanberg, C. R., Brown, K. G., & Simmering, M. J. 2003. E-learning: emerging uses, empirical results and future directions. *International Journal of Training and Development*, 7(4):245-258.

Wiedenbeck, S. 1985. Novice/expert differences in programming skills. *International Journal of Man-Machine Studies*, 23(4), 383-390.

Winslow, L.E. 1996. Programming pedagogy - A psychological overview. *SIGCSE Bulletin* 28:17-22.

Wolff, L. 2002. The African Virtual University: the challenge of higher education development in sub-Saharan Africa. TechKnowLogia, *International Journal of Technologies for the Advancement of Knowledge and Learning*, Volume 4, Issue 2.

Wood, D., Bruner, J. S., & Ross, G. 1976. The role of tutoring in problem solving*. *Journal of child psychology and psychiatry*, 17(2):89-100.

Wu, W. 2005. Web based English learning and teaching in Taiwan: Possibilities and Challenges. In First Hsiang-shan area Intercollegiate International Conference on English language teaching. Taipei: Crane Publishing, pp. 215-226. Hsinchu, Taiwan. May 2005.

Yacob, A. & Saman, M.Y. 2012, Assessing level of motivation in learning programming among engineering students. The International Conference on Informatics and Applications (ICIA2012). Malaysia. June 3 – 5, 2012.

Yin, R. 1994. *Case study research: Design and methods (2nd ed.)*. Thousand Oaks, CA: Sage Publishing.

Ylirisku, S., & Buur, J. 2007. *Designing with Video: Focusing the user-centred design process*. Springer.

BIBLIOGRAPHY OF ADDITIONAL REFERENCES

This section lists all references reviewed but not included in this thesis.

Amiel, T., & Reeves, T. C. 2008. Design-based research and educational technology: Rethinking technology and the research agenda. *Educational Technology & Society*, 11(4):29-40.

Amory, A. 2007. Game object model version II: a theoretical framework for educational game development. *Educational Technology Research and Development*, 55(1):51-77.

Basit, T. 2003. Manual or electronic? The role of coding in qualitative data analysis. *Educational Research*, 45(2):143-154.

Blignaut A.S. & Lillejord S. 2005. Lessons from cross-cultural online learning community. *South African Journal of Higher Education*. 19:1350-1367.

Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A. & Miller, P. 1997. Mini-languages: A way to learn programming principles. *Education and Information Technologies*, 2:65-83

Čisar, S.M., Radosav, D., Pinter, R. & Čisar, P. 2011. Effectiveness of program visualization in learning Java: A case study with Jeliot 3. *International Journal of Computers, Communications & Control*. VI: 669-682

Corney, M., Teague, D., & Thomas, R. N. 2010. Engaging students in programming. In Proceedings of the Twelfth Australasian Conference on Computing Education-Volume 103:63-72. Brisbane, Australia, January 18-22, 2010

Costello, E. 2004. The Use of a Software Enabled Scaffolding Environment to aid Novice Programmers. Department of Computer Science, Trinity College Dublin, Unpublished Master's Thesis, 1-156.

Coull, N. J. 2008. SNOOPIE: development of a learning support tool for novice programmers within a conceptual framework (Doctoral dissertation, University of St Andrews).

Crews, T., & Butterfield, J. 2002. Using technology to bring abstract concepts into focus: A programming case study. *Journal of Computing in Higher Education*, 13(2):25-50.

Cronjé, J. C., & Brittz, B. 2005. Programming in the real world. *Education as Change*, 9(2):131-161.

Crook C. & Harrison C., 2008. Web 2.0 technologies for learning at key stages 3 and 4: Summary report.

De Raadt, M. 2008. Teaching programming strategies explicitly to novice programmers (Doctoral dissertation, University of Southern Queensland).

Denzin, N. K., & Lincoln Y. S. 1994. *Entering the field of qualitative research*. In N. K. Denzin & Y. S. Lincoln (Eds.), *Handbook of qualitative research* (pp. 1-17).

Donmez, O. & Inceoglu, M.M. 2008. A web based tool for novice programmers: Interaction in use. In *Proceeding of the international conference on Computational Science and Its Applications, Part I (ICCSA '08)*, Osvaldo Gervasi, Beniamino Murgante, Antonio Laganà, David Taniar, Youngsong Mun, and Marina L. Gavrilova (Eds.). Springer-Verlag, Berlin, Heidelberg.

Egan, M.H. & McDonald, C. 2014. Program visualization and explanation for novice C programmers. Proceedings of the sixteenth Australasian Computing in Education Conference (ACE2014). Auckland, New Zealand. January 2014.

Fleischer, R. & Kucera, L. 2001. Algorithm animation for teaching. *Software Visualization, State-of-the-Art Survey, Stephan Diehl (ed.). Springer LNCS 2269, pp. 113-128*. International Seminar, Volume 2269. Dagstuhl Castle, Germany, May 20-25, 2001.

Gomes, A., & Mendes, A. J. 2007. Learning to program-difficulties and solutions. In International Conference on Engineering Education–ICEE (Vol. 2007). University of Coimbra – Portugal, September 3-7, 2007.

Hadjerrouit, S. 2008. Towards a Blended Learning Model for Teaching and Learning Computer Programming: A Case Study. Institute of Mathematics and Informatics, Vilnius, *Information in Education*, 7(2):181-210.

Jenkins, T. 1998. A participative approach to teaching programming, integrating technology into computer science education, Dublin City University, 1998, ACM Press, 125-129.

Jenkins, T. 2001. Teaching programming - A journey from teacher to motivator, LTSN-ICS 2nd Annual Conference 2001, University of North London.

Karavirta, V. Korhonen, A. & Malmi, L. 2006. Taxonomy of algorithm animation languages. Proceedings of the 2006 ACM symposium on Software Visualization. 77-85. New York, USA.

Kelle, U. 2001. Sociological Explanations between Micro and Macro and the Integration of Qualitative and Quantitative Methods. *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research*, 2(1):95-117.

Kelleher, C., & Pausch, R. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*, 37(2):83-137.

Kessler, C. & Anderson, J. 1989. Learning flow of control: recursive and iterative procedures. In Soloway & Spohrer: *Studying the Novice Programmer*, *Human-Computer Interaction*: 229-260.

Kölling, M. & Rosenberg, J. 1996. Blue - A language for teaching object-oriented programming, Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education. Philadelphia, USA. February 15-17, 1996, pp. 190-194

Korhonen, A., Malmi, L., McNally, M., Rodgers, S. & Velázquez-Iturbide, J.Á. 2003. Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin*, 35(2):131–152.

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

Marshall M.N. 1998. Qualitative study of educational interaction between general practitioners and specialists. *British Medical Journal*. 316:442-445.

Mashile E.O & Pretorius F.J. 2003. Challenges of online education in a developing country. *South African Journal of Higher Education*. 17(1).

Matthíasdóttir, Á. 2006. How to teach programming languages to novice students? Lecturing or not. In Proceedings of the International Conference on Computer Systems and Technologies-CompSysTech. Veliko Tarnovo, Bulgaria, June 15-16, 2006.

Mayer, R., Dyck, J. & Vilberg, W. 1986. Learning to program and learning to think: what's the connection? In E. Soloway & J.C. Spohrer (eds.): *Studying the Novice Programmer*. *Communications of the ACM*, 29(7):605-610.

McGill, T. J., & Volet, S. E. 1997. A conceptual framework for analysing students' knowledge of programming. *Journal of Research on Computing in Education*, 29(3):276-297.

Milne, I. & Rowe, G. 2002. Difficulties in Learning and teaching Programming - Views of Students and Tutors. *Education and Information Technologies*, 7(1):55-66.

Milne, I. & Rowe, G. 2004. OGRE: Three-dimensional program visualization for novice programmers. *Education and Information Technologies*, 9(3): 219-237.

Milner, W. W. 2011. Concept development in novice programmers learning Java (Doctoral dissertation, University of Birmingham).

- Munoz C.L. & Towner T.L. 2009. Opening Facebook: How to use Facebook in the College classroom. In I. Gibson et al. (Eds.), *Proceedings of Society for Information Technology & Teacher Education International Conference 2009* (pp. 2623-2627). Chesapeake, VA: AACE.
- Myller, N., Bednarik, R., & Moreno, A. 2007. Integrating dynamic program visualization into BlueJ: The Jeliot 3 extension. In *Seventh IEEE International Conference on Advanced Learning Technologies, 2007. ICAALT 2007: 505-506*. IEEE. July 18-20, 2007, Niigata, Japan.
- Postle G., Sturman A., Mangubhai, F., Cronk, P., Carmichael A., McDonald J., Reushle S., Richardson L. & Vickery B. 2003. Online teaching and learning in higher education: A case study. EIP Project Report, Canberra: DEST.
- Powers, K.Gross, P., Cooper, S., McNally, M.F., Goldman, K.J., Proulx, V.K. & Carlisle, M.C. 2006. Tools for teaching introductory programming: what works? In *proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2006, Houston, Texas, USA, March 3-5, 2006*.
- Randall B., Gilliver R.S. & Ming P.Y. 2000. Rewiring learning on the Web – Shaping education in cyberspace. 15th BILETA Conference: Electronic Datasets and Access to Legal Information. University of Warwick, Coventry, England. April, 14, 2000.
- Reiss, S. P. 1987. A conceptual programming environment. In *Proceedings of the 9th international conference on Software Engineering* (pp. 225-235). IEEE Computer Society Press. California, USA, March 30 - April 2, 1987.
- Rogalski, J. & Samurçay, R. 1990. Acquisition of programming knowledge and skills. In J.M. Hoc, T.R.G. Green, R. Samurçay, & D.J. Gillmore (Eds.), *Psychology of programming, 157–174*. London: Academic Press.
- Rountree, N., Rountree, J., Robins, A. 2002. Identifying the danger zones: Predictors of success and failures in a CS1 class. *Special Interest Group on Computer Science Education (SIGCSE) Bulletin, 34(4):121-124*.
- Ryan P. 2008. A small experiment in online learning. *SA Journal of Higher Education*. Volume 22(4), pp. 877-888.
- Santos, Á., Gomes, A. & Mendes, J. 2010. Integrating new technologies and existing tools to promote programming learning. *Algorithms, 3:183-196*.
- Sensalire, M., Ogao, P., & Telea, A. 2009. Evaluation of software visualization tools: Lessons learned. In *Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009. 5th IEEE International Workshop on Visualizing Software TECHNICAL PROGRAM*, (pp. 19-26). September 25 – 26, 2009.

Simelane S, Blignaut S. & van Ryneveld L. 2007. Preparing lecturers to integrate educational technology into their teaching and learning practices. *South African Journal of Higher Education*. 21(7).

Smith, P. & Webb, G. I. 1998. Overview of a low-level program visualisation tool for novice c programmers. In Proceedings of the Sixth International Conference on Computers in Education (ICCE'98). Beijing, China. Oct 14-17, 1998, pp. 213-216. Springer-Verlag.

Smith, P., & Webb, G. I. 1999. Evaluation of Low-Level Program Visualisation for Teaching Novice C Programmers. In G. Cumming, T. Okamoto, & L. Gomez (Eds.), Proceedings of the Seventh International Conference on Computers in Education (ICCE'99) 2:385-392. IOS Press, 1999. Chiba, Japan.

Tatcher A. 2007. Using the World Wide Web to support classroom lectures in a psychology course. *South African Journal of Psychology*. 37 (2):348–353.

Taylor-Powell, E., & Renner, M. 2003. *Analyzing qualitative data*. University of Wisconsin--Extension, Cooperative Extension.

Tshibalo A.E. 2007. The potential impact of computer aided assessment technology in higher education. *South African Journal of Higher Education*, 21(6).

Utting, I., Cooper, S., Kölling, M., Maloney, J., & Resnick, M. 2010. Alice, Greenfoot, and Scratch--a discussion. *ACM Transactions on Computing Education (TOCE)* Volume 10, Issue 4. November 2010.

Wiedenbeck, S., Ramalingam, V., Sarasamma, S.& Corritore, C.L. 1999. A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers*, 11:255–282.

Xiaohui, H. 2006. Improving teaching in Computer Programming by adopting student-centred learning strategies. *The China Papers*, 47 – 48, November 2006.

Yin, R. K. 1981. The case study crisis: some answers. *Administrative science quarterly*, 26(1), 58-65.

Zuckerman, O., Blau, I., & Monroy-Hernández, A. 2009. Children's participation patterns in online communities: An analysis of Israeli learners in the Scratch online community. *Interdisciplinary Journal of E-Learning and Learning Objects*, 5:263-274.

APPENDICES

Appendix A: Invitation Letter to Participate

PARTICIPATION IN MASTERS DEGREE RESEARCH

Instructional Technology for the teaching of novice programmers at a University of Technology

Dear Student

You are asked to participate in a research study conducted by Mr. G Rudolph (researcher), Prof R de la Harpe (research supervisor), Dr E Pineteh (research co-supervisor) and Mr. I Van Zyl (social scientist and workshop facilitator), from the Faculty Informatics & Design: Department of Information Technology at the Cape Peninsula University of Technology.

The pilot study will be conducted in the form of a workshop on the 2nd Floor (dance floor) of the CPUT Barc Building, Roeland Street, Cape Town on Friday 27th January 2012 from 09h00 until 12h30, registration for the workshop taking place as from 08h30.

The purpose of the study is:

- To investigate how instructional technology can be used to address the skills gap of students learning programming concepts.
- To identify existing technology solutions and how they have contributed to the teaching of programming.

Every effort will be made to ensure confidentiality of any identifying information that is obtained in connection with this study.

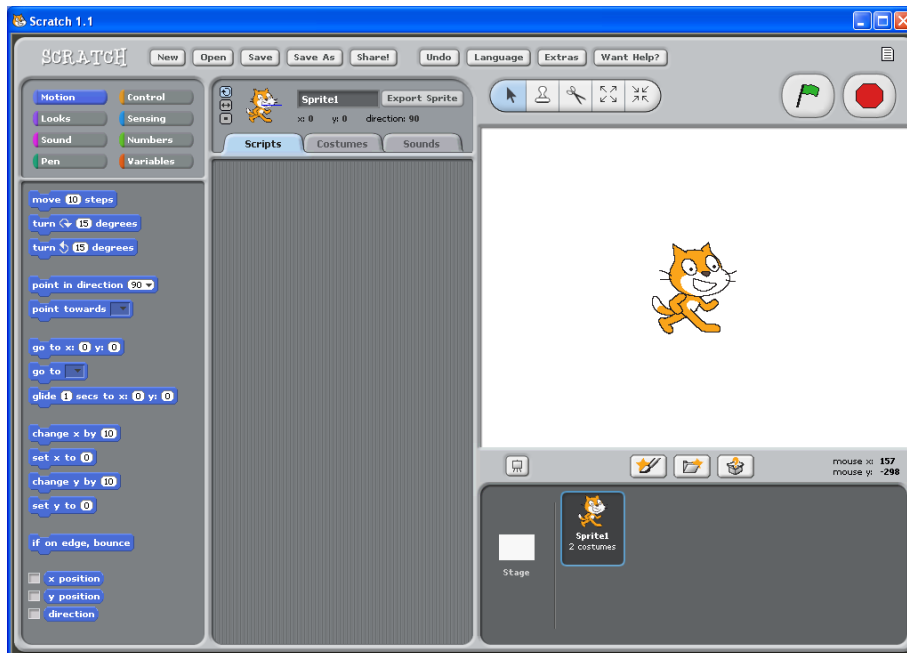
SIGNATURE OF RESEARCH PARTICIPANT

**I give my full consent to participate in this study and give permission to Mr G Rudolph to contact me on my mobile number: _____
or email address _____**

Appendix B: Clock Project Worksheet

In this project you will experiment with importing backgrounds that are not part of the standard set that comes with scratch and also experiment with editing the size of Sprites.

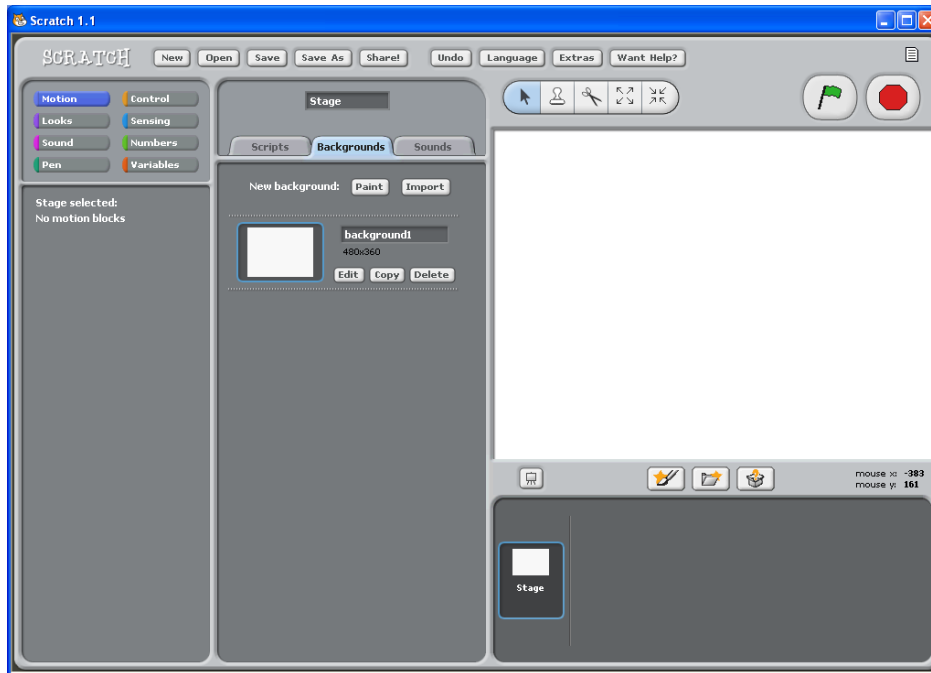
Start a new project



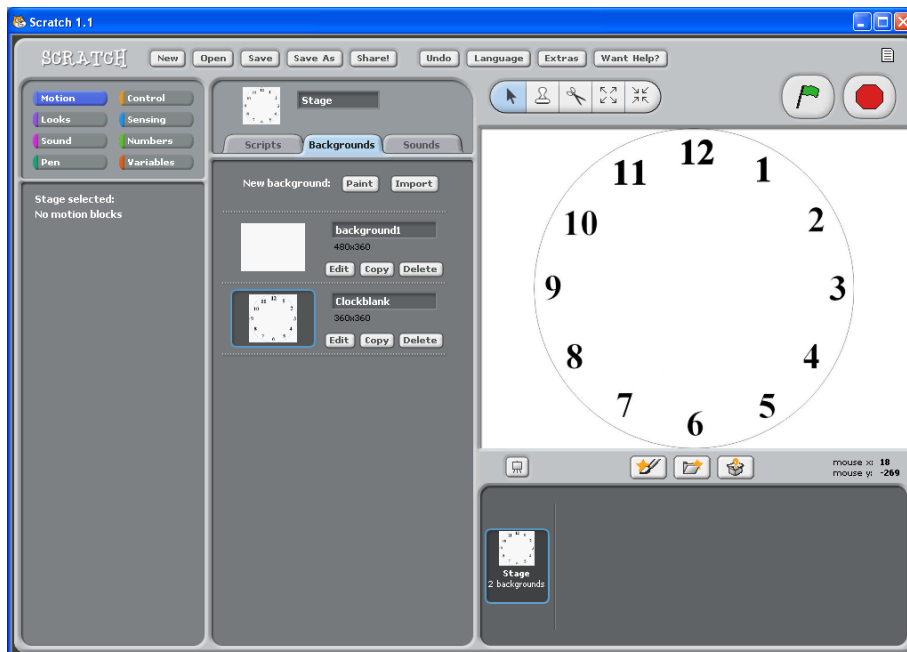
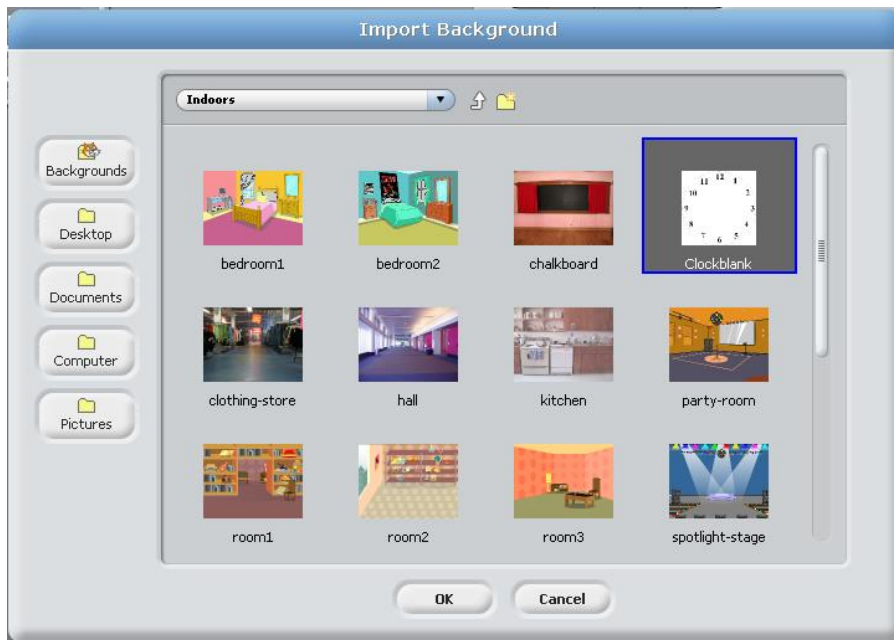
Click on the scissors and select the cat sprite to delete it

Double click the **stage** button

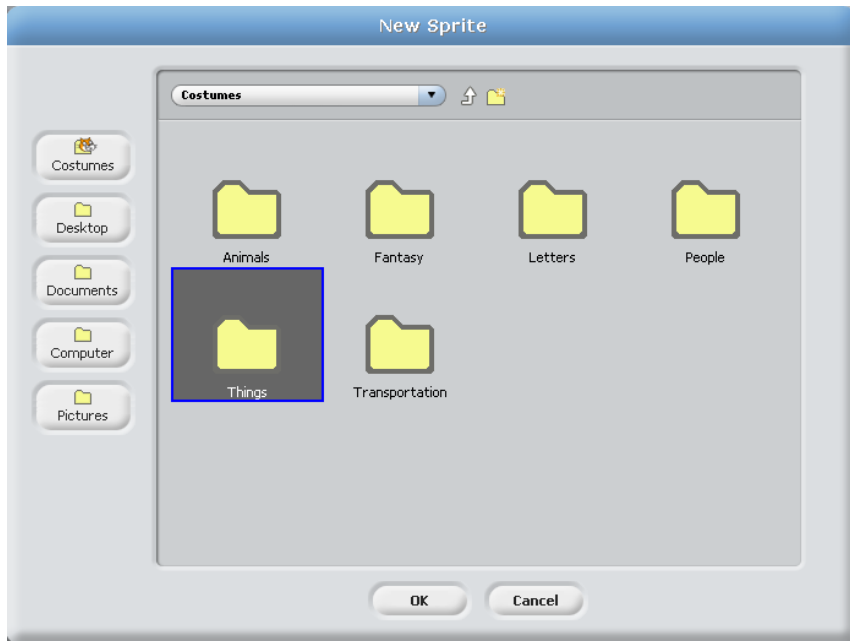
Select the centre tab entitled **backgrounds**



Click the **import** button and browse for the **clock face** (note that this isn't part of the standard files supplied with Scratch and so you may have to browse away from the files initially presented – you can import any background file in jpeg format)



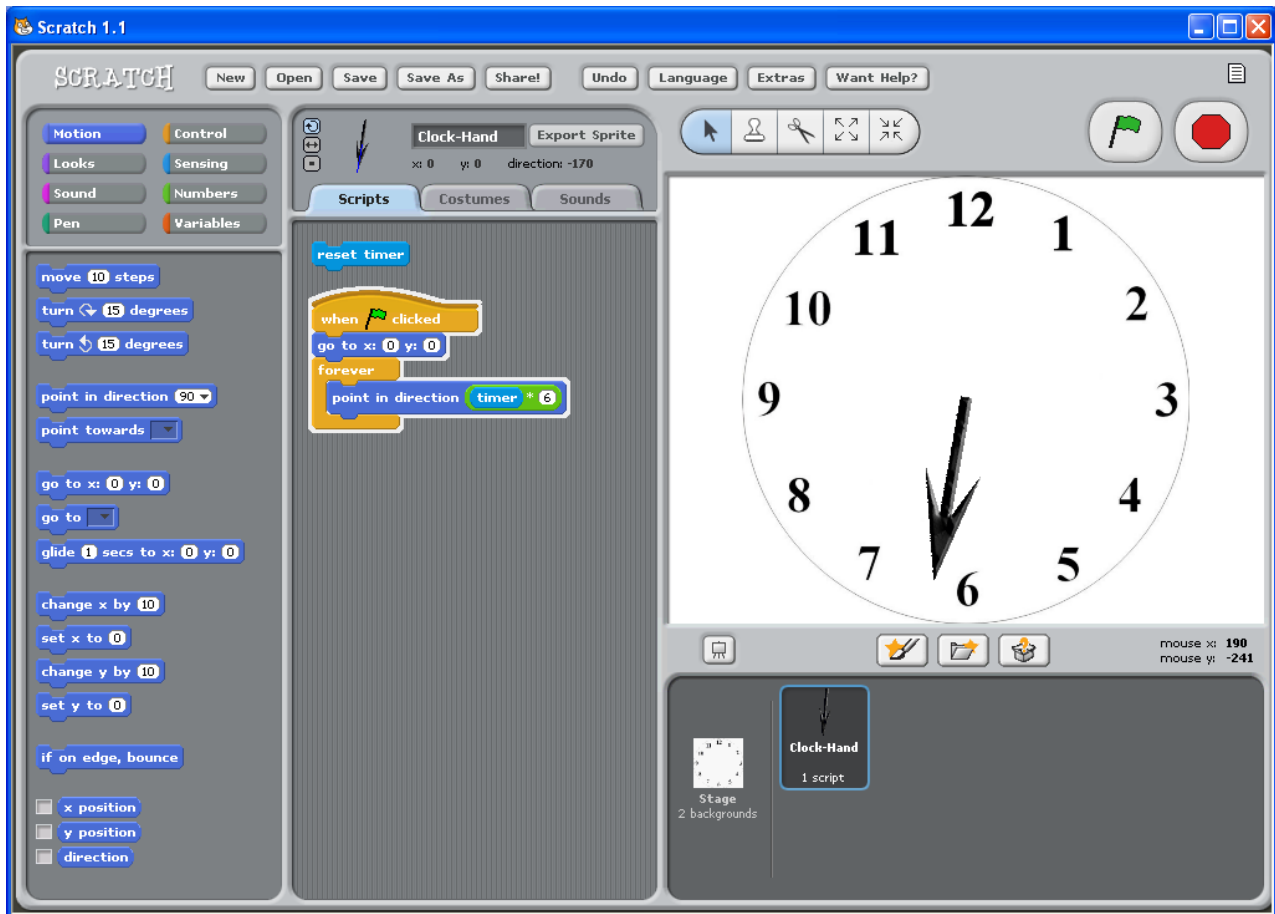
Click the centre button below the stage in order to import a new sprite



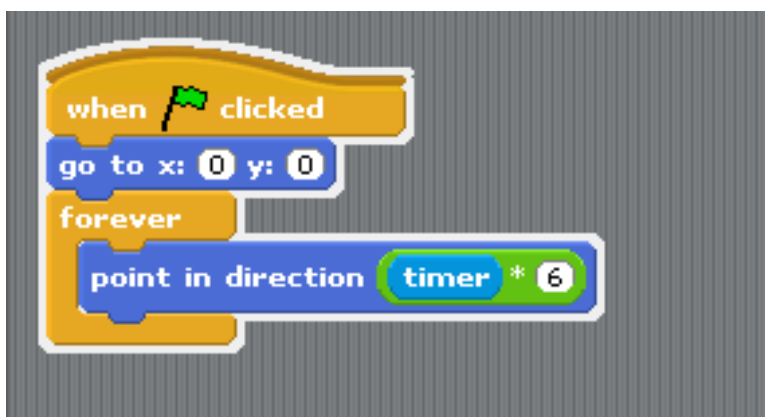
Select the **Things** folder



Select the clock hand and click OK

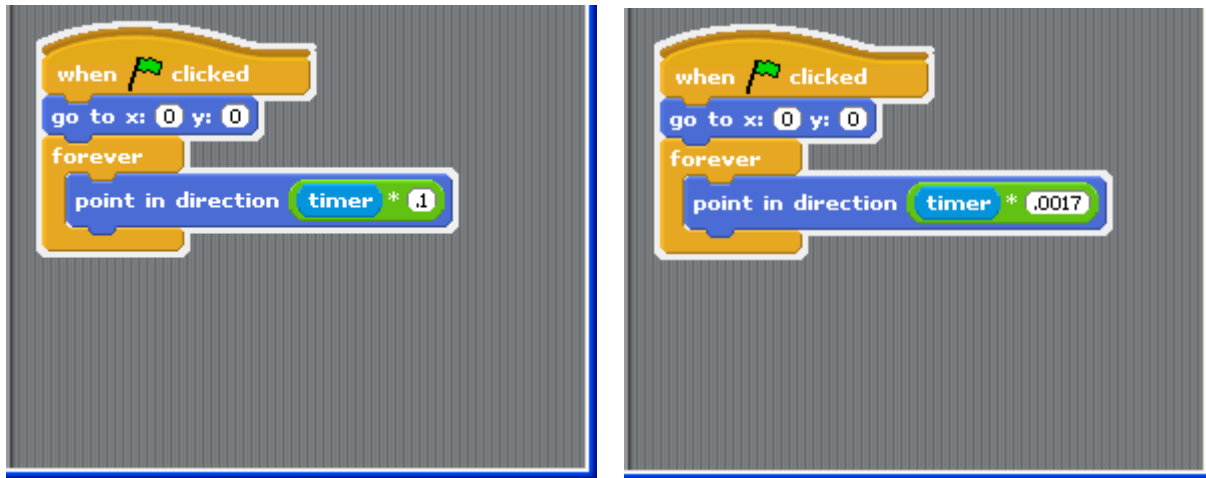


Note that the script automatically centres the hand on the page when you run the script
 Note the time on the clock had is set to 6.
 This roughly corresponds to the speed of a second hand on a clock.

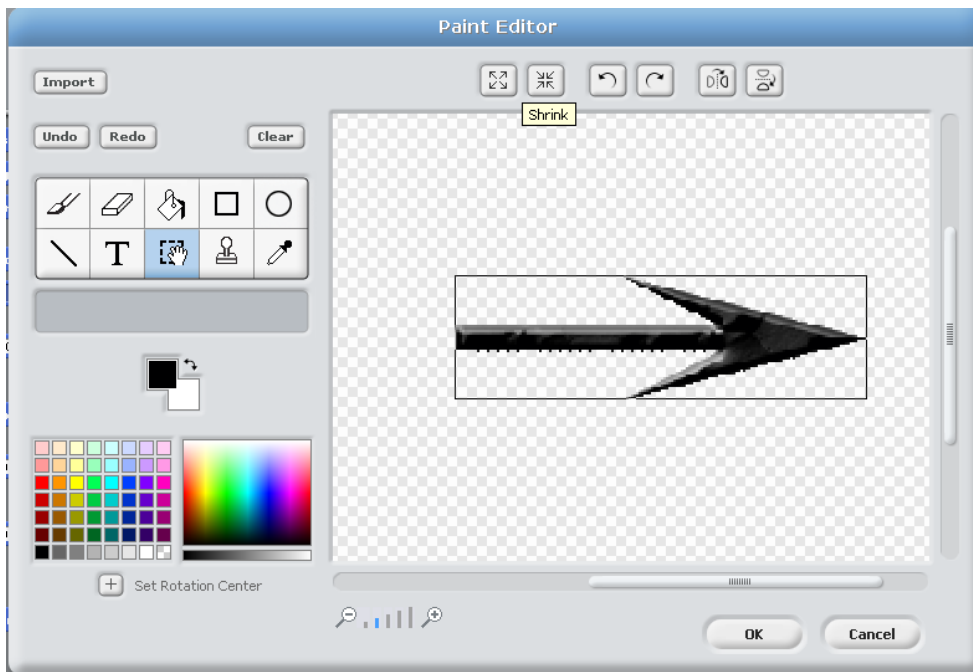


Import two other clock hands
 Rename all the hands as Hour, Minute and Second so that you don't get them mixed up

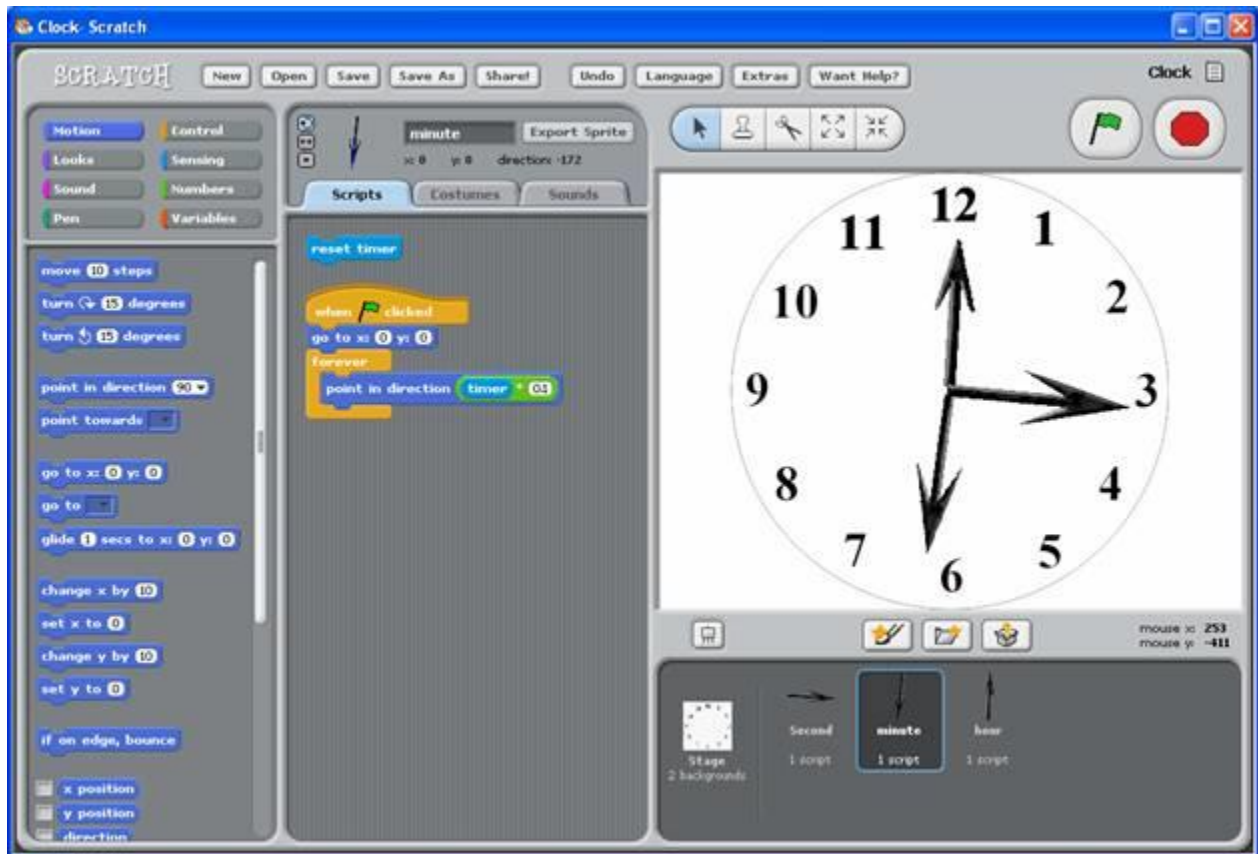
Set the speed of the other hands to 0.1 and 0.0017 (these speeds roughly correspond to the speeds of minute and hour hands on a clock)



Edit the size of the hands



You have made a clock!!



Appendix C: Questionnaire

INSTRUCTIONAL TECHNOLOGY FOR THE TEACHING OF NOVICE PROGRAMMERS AT A UNIVERSITY OF TECHNOLOGY

Evaluation Form

NAME (will be kept anonymous):

Please supply the relevant answer below each question. Avoid brief answers if possible; use as much space as you need.

1. Briefly explain your experience of learning and understanding programming concepts at the beginning of your 1st year.

2. How did you experience this tool (Scratch)? You may want to say something about the navigation and visibility; ease of use; using a single-window user interface / multi-purpose design? **(Scratch uses a single-window user interface which has four main panes)*

3. How could the elimination of the compilation step influence the ability of the learner to learn programming concepts?

4. What is your opinion on help screens, default parameters and illuminating demonstrations for commands in terms of learning the programming concepts?

5. How useful is visual feedback during code/script execution and during troubleshooting?

6. How will the ability of the program that still runs even if it is not correct by eliminating syntax or run time errors influence the learning of programming concepts?

7. What is your opinion on having variables as concrete (visible) objects to assist with learning programming concepts?

8. Please suggest any possible ideas that you think can help 1st year Development Software students to master the programming concepts

you've used today. Please consider any kind of possible instructional tools, even non-technological ones.

- 9. What is your opinion about the suitability of Scratch as an instructional tool to assist new programmers with the learning of new programming concepts? Please also comment on its suitability for our context, keeping in mind the diversity of our students.**