THE USE OF PAIR-PROGRAMMING TO ENHANCE THE ACADEMIC PERFORMANCE OF TERTIARY LEVEL SOFTWARE DEVELOPMENT STUDENTS

Kindu Wa Mulumba Kafilongo
Baccalaureus Technologiae in Information Technology

Dissertation submitted in fulfilment of the requirements for the degree Magister Technologiae in Information Technology, Faculty of Applied and Computer Sciences, Vaal University of Technology

Supervisor: Prof A Jordaan
Co- Supervisor: Mr R Baxter

April 2016
DECLARATION

I, Kindu Wa Mulumba Kafilongo, hereby declare that the work which is submitted here is the product of my own independent research and that all the sources I have used and quoted have been pointed out and acknowledged by means of complete references. In addition I declare that the work is submitted for the first time at this university/faculty towards the Masters Technologiae (MTech) degree in the Information Technology department and that it has never been submitted to any other university/faculty for the purpose of obtaining a degree.

__________________________  _______________________
Signature                       Date
ACKNOWLEDGEMENTS

I hereby express my heartfelt gratitude to:

- The Almighty God, for His protection, wisdom and strength bestowed upon me to complete this research. Glory to the Holy Trinity.

- To my family, thank you for walking this road with me, sharing your valued opinions and suggestions from an educationist point of view.

- To my wife to be, Martha Mathethe, for her unconditional support, love, kisses and sitting by my side during this research project. It gave me strength to carry on.

- My supervisor, Prof Annelie Jordaan and my co-supervisor, Mr Roger Baxter who I was fortunate to have been able to work with, many thanks for their patience, support, guidance and assistance throughout the research.

- Dr Pieter Conradie, who assisted me with my first article by encouraging me and being supportive.

- Ms Christa van Wyk, for the technical editing and proofreading, thank you.

- The Vaal University of Technology (VUT) main campus students that took part in the research and the institution for enabling me to conduct this study.
DEDICATION

This dissertation is dedicated to my whole family.
ABSTRACT

The number of students passing computer programming modules at Higher Education Institutions (HEIs) in South Africa at first year level is low. Only with the second attempt do most students pass. This delay results in students completing their three-year undergraduate qualification in four or even five years. One potential contribution towards addressing this problem is the introduction of a collaborative (cooperative) pedagogical approach where students develop software in teams, known in the Information Technology (IT) sector as pair-programming. This study endeavoured to investigate the impact of pair-programming on the academic progress of students registered for the Information Technology qualification at HEIs in South Africa.

The study warranted the selection of action research as the most appropriate research strategy. Multi-methods data collection was carried out over two consecutive semesters. The data collection methods included a semi-structured interview, observations and empirical assessment. The participants were students registered for the Information Systems module, which focuses on software development. Pair-programming was introduced to one group of software development students, while a second group continued with the normal solo-programming approach. Semi-structured interviews were held with the students before commencement and after completion of the pair-programming intervention, to establish a change, if any, in the academic performance, attitude and enjoyment level of students introduced to pair-programming compared to those who continued with solo-programming. Observations were conducted throughout the course of the practical sessions over both semesters. Empirical assessments were done by means of tests given to both groups of students during the practical sessions, three tests per semester. Data analysis techniques included t-tests and thematic analysis.

The findings concluded that pair-programming had a significant positive impact on the academic progress of IT students, including an increase in the enjoyment level and a more positive attitude towards software development.

Keywords: Pair-programming, eXtreme Programming (XP), collaborative learning, agile approach, academic progress, research methodology, action research.
TABLE OF CONTENTS

DECLARATION ........................................................................................................... ii
ACKNOWLEDGEMENTS ........................................................................................... iii
DEDICATION ............................................................................................................... iv
ABSTRACT .................................................................................................................. v
LIST OF FIGURES ...................................................................................................... x
LIST OF TABLES ......................................................................................................... xi

CHAPTER 1: INTRODUCTION, PROBLEM STATEMENT AND OBJECTIVES ......1
1.1  INTRODUCTION ................................................................................................... 2
1.2  BACKGROUND ...................................................................................................... 2
   1.2.1  Traditional systems development ............................................................... 5
       1.2.1.1  Waterfall methodology ...................................................................... 5
       1.2.1.2  Spiral methodology ........................................................................... 6
   1.2.2  Agile systems development ....................................................................... 8
1.3  RESEARCH PROBLEM ....................................................................................... 10
1.4  RESEARCH QUESTIONS .................................................................................. 10
   1.4.1  Primary Research Question (PRQ) ........................................................... 10
   1.4.2  Secondary Research Questions (SRQs) ..................................................... 11
1.5  RESEARCH OBJECTIVES ................................................................................ 11
   1.5.1  Primary objective ...................................................................................... 11
   1.5.2  Secondary objectives ................................................................................ 11
1.6  RESEARCH DESIGN ......................................................................................... 12
   1.6.1  Research philosophy ................................................................................. 12
   1.6.2  Research methodology ............................................................................ 14
       1.6.2.1  Qualitative research methodology .................................................... 15
       1.6.2.2  Quantitative research methodology .................................................. 15
       1.6.2.3  Mixed methods research methodology ............................................ 15
   1.6.3  Research strategy ....................................................................................... 15
   1.6.4  Data collection .......................................................................................... 16
       1.6.4.1  Unit of observation .......................................................................... 16
       1.6.4.2  Unit of analysis ................................................................................. 16
       1.6.4.3  Data collection techniques ................................................................ 16
   1.6.5  Data analysis ............................................................................................. 17
CHAPTER 2: LITERATURE REVIEW ..............................................................21
2.1 INTRODUCTION ..................................................................................22
2.2 IT STUDENTS IN A SOFTWARE DEVELOPMENT ENVIRONMENT ..........22
   2.2.1 Enjoyment of software development ...........................................22
   2.2.2 Attitude towards software development .....................................23
2.3 INNOVATIVE AGILE SOFTWARE DEVELOPMENT METHODS ..........23
   2.3.1 Agile Unified Process (AUP) ......................................................26
   2.3.2 Adaptive Software Development (ASD) .....................................27
   2.3.3 Dynamic Systems Development Method (DSDM) .......................27
   2.3.4 eXtreme Programming (XP) .................................................28
2.4 PAIR-PROGRAMMING .......................................................................30
   2.4.1 How does pair-programming work? .........................................33
   2.4.2 Pair-programming as collaborative learning approach ...............35
   2.4.3 Advantages of pair-programming .............................................37
      2.4.3.1 Enjoyment ........................................................................37
      2.4.3.2 Confidence .......................................................................38
      2.4.3.3 Program quality ................................................................38
      2.4.3.4 Student academic performance .........................................38
      2.4.3.5 Attitude ............................................................................38
      2.4.3.6 Communication .................................................................39
2.5 CONCLUSION .....................................................................................39
2.6 SUMMARY .........................................................................................39

CHAPTER 3: RESEARCH METHODOLOGY .................................................42
3.1 INTRODUCTION ..................................................................................43
3.2 MEANING OF RESEARCH .................................................................43
3.3 RESEARCH PROBLEM ......................................................................44
3.4 RESEARCH DESIGN ..........................................................................45
   3.4.1 Research philosophy .................................................................46
CHAPTER 5: FINDINGS AND RECOMMENDATIONS

5.1 INTRODUCTION

5.2 RESEARCH QUESTIONS

5.2.1 Secondary Research Question 1

5.2.2 Secondary Research Question 2

5.2.3 Secondary Research Question 3

5.2.4 Secondary Research Question 4

5.2.5 Secondary Research Question 5

5.2.6 Secondary Research Question 6

5.3 SUMMARY OF FINDINGS

5.4 PRACTICAL CONTRIBUTION

5.5 LIMITATIONS

5.6 FUTURE RESEARCH

5.7 SUMMARY

REFERENCES

ANNEXURE A: Interview guide

ANNEXURE B: Interview responses from participants (before and after)

ANNEXURE C: Feedback from students - objectives

ANNEXURE D: Published journal article

ANNEXURE E: Language editing certificate
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Graphical representation of dissertation framework</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>Graphical representation of Chapter 1</td>
<td>1</td>
</tr>
<tr>
<td>1.3</td>
<td>Waterfall phases during development process</td>
<td>6</td>
</tr>
<tr>
<td>1.4</td>
<td>Spiral Model</td>
<td>7</td>
</tr>
<tr>
<td>2.1</td>
<td>Graphical representation of Chapter 2</td>
<td>21</td>
</tr>
<tr>
<td>2.2</td>
<td>Agile Unified Process phases</td>
<td>27</td>
</tr>
<tr>
<td>2.3</td>
<td>DSDM phases</td>
<td>28</td>
</tr>
<tr>
<td>2.4</td>
<td>Student performance in pair-programming assignments</td>
<td>31</td>
</tr>
<tr>
<td>2.5</td>
<td>Elapsed time spent on the project</td>
<td>33</td>
</tr>
<tr>
<td>3.1</td>
<td>Graphical representation of Chapter 3</td>
<td>42</td>
</tr>
<tr>
<td>3.2</td>
<td>Meaning of research</td>
<td>43</td>
</tr>
<tr>
<td>3.3</td>
<td>Research onion</td>
<td>45</td>
</tr>
<tr>
<td>3.4</td>
<td>A framework for research philosophy</td>
<td>46</td>
</tr>
<tr>
<td>3.5</td>
<td>Social theory analysis using four paradigms</td>
<td>49</td>
</tr>
<tr>
<td>3.6</td>
<td>Deductive approach versus Inductive research approach</td>
<td>51</td>
</tr>
<tr>
<td>3.7</td>
<td>Action research cycle</td>
<td>52</td>
</tr>
<tr>
<td>3.8</td>
<td>Example of sample</td>
<td>59</td>
</tr>
<tr>
<td>3.9</td>
<td>Methods of sampling</td>
<td>60</td>
</tr>
<tr>
<td>3.10</td>
<td>Streamlined Codes-to-Theory model for qualitative inquiry</td>
<td>62</td>
</tr>
<tr>
<td>3.11</td>
<td>Stages of coding in thematic analysis to theory</td>
<td>64</td>
</tr>
<tr>
<td>4.1</td>
<td>Graphical representation of Chapter 4</td>
<td>67</td>
</tr>
<tr>
<td>4.2</td>
<td>Group A: Students using pair-programming</td>
<td>74</td>
</tr>
<tr>
<td>4.3</td>
<td>Group B: Individual programming</td>
<td>75</td>
</tr>
<tr>
<td>4.4</td>
<td>Students’ attitude towards programming</td>
<td>78</td>
</tr>
<tr>
<td>5.1</td>
<td>Graphical representation of Chapter 5</td>
<td>80</td>
</tr>
<tr>
<td>5.2</td>
<td>Proposed Pair-programming Framework</td>
<td>86</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 1.1: Boehm-Spiral Methodology stages ..............................................................7
Table 1.2: Agile vs. Traditional Methodologies .............................................................9
Table 1.3: Summary overview of research questions and objectives .............................20
Table 2.1: Summary of Beck’s (2000) twelve principles of agility ...............................24
Table 2.2: Profile of agile development models ................................................................25
Table 2.3: Students’ views towards pair-programming (Summary) ...............................32
Table 2.4: Percentage of test cases passed ....................................................................32
Table 3.1: Positivism and Interpretivism ......................................................................48
Table 3.2: Qualitative vs Quantitative research methods ..............................................54
Table 3.3: Apposite research design for pair-programming research at HEIs ...............65
Table 4.1: Enjoyment level: Before pair-programming ...............................................71
Table 4.2: Enjoyment level: After pair-programming ..................................................72
Table 4.3: Paired $t$-test results for solo-programming ..............................................76
Table 4.4: Paired $t$-test results for agile programming (pair-programming) .............76
Table 4.5: Attitude: Before pair-programming .............................................................77
Table 4.6: Attitude: After pair-programming ...............................................................78
CHAPTER 1: INTRODUCTION, PROBLEM STATEMENT AND OBJECTIVES

CHAPTER 2: LITERATURE REVIEW

CHAPTER 3: RESEARCH METHODOLOGY

CHAPTER 4: ANALYSIS AND DISCUSSION

CHAPTER 5: FINDINGS AND RECOMMENDATIONS

Figure 1.1: Graphical representation of dissertation framework

CHAPTER 1: INTRODUCTION, PROBLEM STATEMENT AND OBJECTIVES

Introduction

Background

Research problem

Research questions

Research objectives

Research design

Delimitation

Ethical considerations

Chapter outline

Summary

Figure 1.2: Graphical representation of Chapter 1
1.1 INTRODUCTION

According to the previous Minister of Education, the honourable Naledi Pandor (2006), science, mathematics and information and communications technology(ies) (ICT) play a vital role in social and economic development. Developing countries need to advance their human and institutional capacity in mathematics, science and technology if they are to succeed in their developmental goals. Pandor (2006) further states that the performance of South African students, as indicated in the Trends in International Mathematics and Science Study (TIMMS) in 2003, 1995 and 1999, confirm that South Africa's most important educational priority is to expand the mathematical and scientific capacity of students.

The current Minister of Higher Education and Training, Dr Blade Nzimande, in his keynote address at the Conference of the South African Heads of Mission (Nzimande, 2014), builds upon the promptings of Ms Pandor. Dr Nzimande (2014) lists the higher education objectives of the National Development Plan (NDP) of South Africa, which include:

“In higher education, the objectives are to increase university enrolments by at least 70% to about 1.6 million; increasing the number of students eligible to study towards maths- and science-based degrees to 450 000…”

1.2 BACKGROUND

In South Africa, provision is made for students to prepare for a career in Information Technology (IT) by offering them this qualification in the Further Education and Training (FET) and Higher Education (HE) bands of the National Qualifications Framework (NQF). One of the four learning outcomes of IT offered in these HE bands focuses on the design and development of appropriate computer-based solutions to specific problems using programming (i.e. software development). A historical perspective shows that Software Development Methodologies (SDMs) came into being to address deficiencies in existing techniques of software development and introduce rigour into the software development process (Avison & Fitzgerald 2003).

In this study, Software Development refers to a course where the students learn how to design and/or develop appropriate and effective computer-based solutions to
specified problems by using programming languages such as VB, PHP, Java, C++ and C#, offered by higher education institutions in South Africa.

The systems (or software) development life cycle (SDLC) was one of the first software development methodologies to be introduced into the academic community and it subsequently became the SDM for the 1970s (Lee 1987). It was designed in an attempt to deliver information systems on time, within budget, and more aligned to the requirements of the user, analogous to the goals of present day SDMs.

“A software development life cycle (SDLC) is a process of building and/or maintaining software systems” (Bender 2003).

The process includes many phases, from prerequisites to the development of the software system, including testing and evaluation. SDLC also consists of the methodologies used by the systems development teams to develop the systems. These methodologies are essential because they constitute the framework for the whole development process. A software system is designed to perform a particular task based on the requirements of the users. Often, the tasks that the system will perform involve complex phases or methodologies; it needs a high level of understanding the requirements of the users in order to develop a successful system (Bender 2003). Figure 1.2 describes the various steps and activities in the SDLC and its associated purpose (Hoffer, George & Valachic 2010).

Therefore, defining the term Systems Development Methodology is not a clear-cut task. Definitions range from the simple to the complex and there is no generally accepted, exact and concise definition of an Information Systems Development Methodology (ISDM) (Avison & Fitzgerald 2003; Livari, Hirscheim & Klein 1999; Wynekoop & Russo 1997).

Avison and Fitzgerald (2003) define SDM as follows:

“A systems development methodology is a collection of procedures, techniques, tools and documentation aids that assists systems developers in their efforts to implement a new information system... A methodology is based on a philosophical view” (Avison & Fitzgerald 2003).
Figure 1.2: Development activities during the systems development life cycle (SDLC) 
(Source: Hoffer et al. 2010)

Avison and Fitzgerald (2003) continue by stating that a methodology contains a number of phases and sub-phases which direct the developers towards selecting relevant techniques for each project stage. These phases assist developers with planning, managing, control, and evaluation of information systems projects. Brinkkemper (1996) equates a method to an approach used to carry out a systems development project. This approach is founded upon:

“…a specific way of thinking, consisting of directions and rules, structured in a systematic way in development activities, with corresponding development products” (Brinkkemper 1996).
Livari et al. (1999) parallels a systems development methodology to a systematic procedure applied to finalise a system or one of several phases of the SDLC, and it consists of “goals, principles, and specific methods and tools, which are selected on the basis of an underlying rationale…” (Livari et al. 1999). Wynekoop and Russo (1997) indicate that methods for developing systems include both process models and methodologies.

From the definitions above it can be concluded that a systems development methodology is a set of phases that must be followed to develop information systems, and it includes specific methods, tools and documentation aids.

The two SDLC methodologies mostly used by system developers are traditional systems development and agile systems development.

1.2.1 Traditional systems development

The most commonly known traditional systems development methods include the Waterfall method and the Spiral method which are classified into the heavyweight methodologies (Nikiforova, Nikulsins & Sukovskis 2009). Traditional systems development methodologies define and document all the requirements at the beginning of a project.

According to Leau (2012), typical traditional systems development consists of four steps: i) Define the project specifications and establish the duration it will take to implement the different stages of the development process; ii) design the architectural plan where a technical infrastructure is generated in the form of diagrams. This phase outlines the map used by the system developers to implement the system; iii) the system developers code until they reach the requirements defined by users. This phase is often subdivided into smaller activities which are disseminated among different developers based on their skills; and iv) provide feedback to the customers and deliver the system once the customers are satisfied.

1.2.1.1 Waterfall methodology

The Waterfall methodology (figure 1.3) is the oldest technique for developing systems and some companies are still using it. It divides the development process into formal steps which have to take place sequentially. Thus, the tasks in each step
have to be fully completed before progressing to the next step. The emphasis of the waterfall method is on the formal specifications (Hughes & Cotterell 2009). The main drawback of the Waterfall method is that too many documents are produced during the development process.

1.2.1.2 Spiral methodology

The Spiral methodology (figure 1.4) was introduced by Boehm (2000) to address problems with the Waterfall Method. As mentioned in Section 1.2.1.1, the emphasis of the Waterfall method is on developing formal documentation—which is extremely time consuming. The Spiral method echoes the relationship of tasks with rapid prototyping for faster completion. This reduces the time and increases the concurrency in designing and building activities. The Spiral methodology develops the system through the layers of the development process and releases in each layer a prototype to users to establish whether the project is on track.
Table 1.1 explains in detail the activities of each of the four cycles as indicated in figure 1.4.

Table 1.1: Boehm-Spiral Methodology stages
(Source: Boehm 2000)

<table>
<thead>
<tr>
<th>CYCLE</th>
<th>STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle 1: Early Analysis</td>
<td>• Step 1: Objectives, alternatives and constraints &lt;br&gt; • Step 2: Risks analysis and prototype &lt;br&gt; • Step 3: Concept of operation &lt;br&gt; • Step 4: Requirement and life cycle plan &lt;br&gt; • Step 5: Objectives, alternatives and constraints &lt;br&gt; • Step 6: Risks analysis and prototype</td>
</tr>
<tr>
<td>Cycle 2: Final Analysis</td>
<td>• Step 7: Simulation, models and benchmarks &lt;br&gt; • Step 8: Software requirements and validation &lt;br&gt; • Step 9: Development plan &lt;br&gt; • Step 10: Objectives, alternatives and constraints &lt;br&gt; • Step 11: Risks analysis and prototype</td>
</tr>
</tbody>
</table>
### 1.2.2 Agile systems development

Agile systems development uses iterative and incremental development wherein the steps within the development process are revised continuously. The iterative method enhances the development of the system by using the feedback from customers to optimise on solutions (Szalvay 2004, Ambysoft, 2012).

The most important characteristic for a methodology to be viewed as being agile is its capability to adjust rapidly to change. This flexibility is obtained through the tools and techniques of the particular methodology. The most generally known agile methodologies are eXtreme Programming (XP); Agile Unified Process (AUP); Adaptive Software Development (ASD); Dynamic Systems Development Method (DSDM); and Lean Software Development (LSD). Of these, eXtreme Programming (XP) is considered the most popular.

Created by Kent Beck in 1996, the purpose of XP was to fulfil a need for a faster, simpler and cheaper way to design software. Beck (2000) argued that the use of XP in industry has been claimed to provide significant benefits and there seems to be potential in the use of the methodology for student projects. In addition, the use of XP is common in most fields of software development (Adams, Goold, Lynch, Daniels, Hazzan & Newman 2003).

In a study done by Zhang (2010), it was found for example that the manufacturing industry widely accepts agility as a new competitive concept. The three elementary types of the agility strategies have been suggested by the taxonomy, namely quick,

<table>
<thead>
<tr>
<th>CYCLE</th>
<th>STEP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle 3: Design</td>
<td>• Step 12: Simulation, models and benchmarks</td>
</tr>
<tr>
<td></td>
<td>• Step 13: Software product design, validation and verification</td>
</tr>
<tr>
<td></td>
<td>• Step 14: Integration and test plan</td>
</tr>
<tr>
<td></td>
<td>• Step 15: Objectives, alternatives and constraints</td>
</tr>
<tr>
<td></td>
<td>• Step 16: Risks analysis and operational prototype</td>
</tr>
</tbody>
</table>

| Cycle 4: Implementation and Testing | • Step 17: Simulation, models and benchmarks                          |
|                                     | • Step 18: Detailed design                                            |
|                                     | • Step 19: Code                                                       |
|                                     | • Step 20: Unit, integration and acceptance testing                    |
|                                     | • Step 21: Implementation (deployment)                                |

| Cycle 4: Implementation and Testing | • Step 17: Simulation, models and benchmarks                          |
|                                     | • Step 18: Detailed design                                            |
|                                     | • Step 19: Code                                                       |
|                                     | • Step 20: Unit, integration and acceptance testing                    |
|                                     | • Step 21: Implementation (deployment)                                |
responsive, and proactive. In relation to software development, all three agile strategies are relevant. In a study performed by Baskerville, Pries-Heje and Madsen (2010), it was found that most software companies are combining agile and plan-driven approaches to achieve the benefits of both during software development.

But, can agile software development methods also play a role in education, and more specifically, can pair-programming, a practice used in XP, be of benefit to IT students who are required to develop software?

Williams et al. (2007, 2008) state that it does, but highlight that this premise must be further tested in different educational environments with different levels of students before this hypothesis can be considered valid. According to Cho (2008), “agile software development methods were developed to provide more customer satisfaction, shorten the development process and allow changing the business requirements during the development process without starting afresh”.

Table 1.2 shows the various differences between agile and traditional methodologies.

<table>
<thead>
<tr>
<th>ASPECTS</th>
<th>AGILE</th>
<th>TRADITIONAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>User requirements</td>
<td>Iterative acquisition</td>
<td>In-depth user specifications are properly defined before implementation</td>
</tr>
<tr>
<td>Cost of rework</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Direction of development</td>
<td>Changeable</td>
<td>Fixed</td>
</tr>
<tr>
<td>Testing</td>
<td>On every iteration</td>
<td>Upon completion of the coding phase</td>
</tr>
<tr>
<td>Involvement of customers</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Suitable Project (size)</td>
<td>Small to medium</td>
<td>Large</td>
</tr>
</tbody>
</table>

Based on Table 1.2, agile systems development is an extremely useful methodology to adopt at tertiary level and used by IT students to develop software or systems. However, it still faces several barriers in putting it into practice.

Various authors suggest that pair-programming, which is an important technique of the eXtreme Programming (XP) Agile Systems Software Development Methodology (SDM), whereby two programmers work at one computer on the same programming task, shows several promising properties for educational purposes (Ambysoft, 2012;
Pair-programming seems to have a positive effect in general on computer science students at universities abroad (Ho, Slaten, Williams & Berenson 2004; Werner, Denner & Bean 2004), specifically in terms of enjoyment (McDowell, Werner, Bullock & Fernald 2006; Werner et al. 2004; Ho et al. 2004), and on their view of the importance or usefulness of the subject (McDowell et al. 2006; Werner et al. 2004).

1.3 RESEARCH PROBLEM

Students in general display a lack of knowledge in solving problems involving various programming languages and technologies (Henson 2002; McMahon 2009). This often results in students dropping out of IT courses because they are struggling on their own without any individual attention or guidance (University World News 2015). Software Development Methodologies (SDMs) came into being to address deficiencies in existing techniques of software development and introduce rigour into the software development process (Avison & Fitzgerald 2003). This research endeavours to build on the statement of Williams et al. (2007, 2008) that although agile software development methods have been proven to play a role in education, the premise needs be tested in different educational environments with different levels of students before the hypothesis can be considered valid.

The main aim of this research is therefore to determine whether the general findings mentioned in section 1.2.2 on agile software development, with the emphasis on pair-programming, are applicable to Information Technology (IT) students at a higher education institution (HEI) in South Africa where diversity in terms of culture, language and upbringing has a major impact on the progress and success of students.

1.4 RESEARCH QUESTIONS

1.4.1 Primary Research Question (PRQ)

The primary research question (PRQ) of this study is stated as follows:
PRQ: How does pair-programming as agile software development method shape the experience of tertiary level IT students with regard to their academic performance in developing software?

1.4.2 Secondary Research Questions (SRQs)

The secondary research questions (SRQs) based on the primary research question, are stated as follows:

SRQ1: How prominent is eXtreme Programming, specifically pair-programming, as educational tool at Higher Education Institutions (HEIs) in general?

SRQ2: What is the impact of pair-programming on IT students’ enjoyment level of software development?

SRQ3: How does pair-programming impact the academic progress of IT students at HEIs in South Africa with regard to software development?

SRQ4: How does pair-programming impact the attitude of IT students towards software development?

SRQ5: What model can be proposed to shape the experience of tertiary level IT students with regard to their academic performance in developing software?

SRQ6: How can pair-programming be implemented optimally in a controlled learning environment?

1.5 RESEARCH OBJECTIVES

1.5.1 Primary objective

The primary objective is to establish how pair-programming can shape the experience of tertiary level IT students in South Africa with regard to their academic performance in developing software.

1.5.2 Secondary objectives

Six secondary objectives have been defined:
i) To determine if eXtreme Programming (XP), specifically pair-programming, is used as educational tool at Higher Education Institutions (HEIs) in general, and if so, how prominent XP is.

ii) To determine the impact of pair-programming on IT students’ enjoyment level of software development.

iii) To determine how pair-programming impacts the academic progress of IT students at HEIs in South Africa with regard to software development.

iv) To determine the impact of pair-programming on the attitude of IT students towards software development.

v) To propose a model that can be used to shape the experience of tertiary level IT students with regard to their academic performance in developing software.

vi) To determine if and how pair-programming can be optimally implemented in a controlled learning environment.

1.6 RESEARCH DESIGN

The research design, also referred to in literature as the research approach, comprises the research philosophy, methodology, strategy, unit of analysis, unit of observation, and data analysis, each outlined in the sub-sections below.

1.6.1 Research philosophy

Academic research is generally classified into the following research philosophies: positivism, interpretivism/constructivism, transformativeism, pragmatism, realism, objectivism and subjectivism (Mackenzie & Knipe 2006; Saunders, Lewis & Thornhill 2009). A study must therefore specify which philosophy will be adopted as each philosophy is constrained to specific ontological, epistemological and methodological prerequisites.

According to Saunders et al. (2009), research is underpinned by the philosophical assumptions which show the particular way in which the world is viewed and understood. Hence, it is important to understand those philosophical assumptions in
order to choose the appropriate approach and to ensure that the researcher adopts the appropriate method to conduct the research.

Each of the research philosophies as indicated by Mackenzie and Knipe (2006) and Saunders et al. (2009), are briefly discussed next.

i) **Positivism**

Researchers supporting positivism strive to attain generalisations that are based on a recurring fact or event (Neuman 2011) by carrying out objective research to quantify social phenomena. Positivists believe that different researchers who measure the same verifiable problem will generate the same or a similar result by cautiously applying statistical tests and following an equivalent research process when exploring a large sample (Creswell 2009). Positivists make use of “observation and measurement in order to predict and control forces that surround us” (O'Leary 2004) to test theories or describe a particular experience.

Positivism was replaced by post-positivism after World War II (Mertens 2005), which focused on the assumption that research is shaped by a number of well-developed theories, including the theory that is being tested. Khun (1962) also held that new understandings may challenge the theoretical framework as a whole. Positivist and post-positivist research is generally aligned with quantitative methods of data collection and analysis (Mackenzie & Knipe 2006).

ii) **Interpretivism/Constructivism**

“Interpretive methods of research start from the position that our knowledge of reality, including the domain of human action, is a social construction by human actors and that this applies equally to researchers” (Walsham 1993). Interpretivists support the idea that the reality is constructed by social actors and that researchers are included in this reality, thus, social phenomena is subjective in nature.

iii) **Transformativism**

Transformative methods recognise that knowledge is constructed in a complex cultural context of power and privilege. Evaluators have to understand realities of communities and social groups they work with (Mertens 2009).
iv) **Pragmatism**

Pragmatism is a subdivision of research philosophy which does not want to join the positivism and interpretivism research philosophies (Tashakkori & Teddlie 1998). Pragmatists start with the research question to ascertain their research framework, thus, they do not question ontology and epistemology as step one.

v) **Realism**

Realism stems from both positivism and interpretivism. Realists believe that the real structure exists separate from human being consciousness and our knowledge is the result of social conditioning (Saunders *et al.* 2009). According to Blaikie (1993), realists admit the possibility of the existence of reality despite science or observation, thus, “there is validity in recognising realities that are simply claimed to exist or act, whether proven or not”.

vi) **Objectivism**

According to Blaikie (1993), objectivism depicts the position that “social entities exist in reality external to social actors concerned with their existence”. Thus, the objectivists support the idea that the social entities exist independently of human influence or manipulation.

vii) **Subjectivism**

Subjectivism holds that “social phenomena are created from the perceptions and consequent actions of those social actors concerned with their existence” (Blaikie 1993). Subjectivists accept that social phenomena are derived from the activities of social entities that interact.

This research study is based on interpretivism as an approach towards social sciences, and opposed to positivism which is based on natural science.

1.6.2 **Research methodology**

Research methodologies are mainly categorised into three types, namely *quantitative, qualitative and mixed methods* (Saunders *et al.* 2009).
1.6.2.1 **Qualitative research methodology**

Researchers assign various definitions to qualitative research methodology, ranging from basic to complex. According to Myers (1997), a qualitative research methodology aims to assist researchers in understanding the behaviour of people inside the social, economic and cultural environment where they belong. With a qualitative research methodology the researcher collects, analyses and interprets data in a non-numerical manner.

1.6.2.2 **Quantitative research methodology**

As with qualitative research, researchers assign various definitions to quantitative research methodology. Saunders *et al.* (2009) captures the essence by stating that a quantitative research methodology uses numbers to collect data by means of questionnaires, and data is represented using graphs or statistics. It is thus an empirical research method.

1.6.2.3 **Mixed methods research methodology**

With mixed methods research methodology, a qualitative or quantitative approach is selected as the initial research methodology; the second approach is adopted along the way of the research due to the inadequacy of the first approach (Creswell 2009). Saunders *et al.* (2009) argue that a research method can consist of both quantitative and qualitative approaches to source both primary and secondary data regarding a mutual subject in the same study.

The nature of this study warrants the use of mixed methods research methodology.

1.6.3 **Research strategy**

“Research strategy is a methodology that helps the researcher to investigate the research issue... An effective research strategy helps the researcher to ...employ a particular research strategy to conduct the research study in an effective manner” (dissertationhelpservice.com).

Sagor (2000, as cited by ASCD 2015) defines action research as “a disciplined process of inquiry conducted by and for those taking the action. The primary reason for engaging in action research is to assist the ‘actor’ in improving and/or refining his
or her actions”. One main aspect of action research is that it assists educators to be more efficient with what they care most about—their teaching and the development and progress of their students (Sagor 2000, as cited by ASCD 2015).

“Action research demands some form of intervention” (Herr & Anderson 2015). For this study, action based research been selected as the most appropriate strategy.

1.6.4 Data collection

1.6.4.1 Unit of observation

The research participants, also known as the unit of observation, are the students who were selected through non-random purposive sampling to partake in this study. With non-random sampling, the probability of choosing any one individual or sample cannot be determined (Coloss Institute 2015). With purposive sampling, the researcher is able to select unit(s) of observation that best meets the research aims (Bless, Higson-Smith & Sithole, 2013). Also, purposive sampling techniques work well with case study research (Neuman, 2005, 2011). The researcher used *judgemental or purposive sampling* to select a sample which is representative of what he thought is a suitable mix of participants for the study (Coloss Institute 2015). The thoughts of the researcher were informed by the literature review in which similar types of case studies were conducted.

1.6.4.2 Unit of analysis

A unit of analysis is representative of the targeted population (Saunders *et al.* 2009). For this research study, the unit of analysis is the pair-programming technique administered to students during their practical software development sessions.

1.6.4.3 Data collection techniques

The multi-methods data collection technique was selected for this research study. *Semi-structured interviews* were held with the participants before commencement and after completion of the pair-programming intervention, to establish a change, if any, in the academic performance, attitude and enjoyment level of students introduced to pair-programming compared to those who continued with solo-programming. *Observations* were conducted by the researcher throughout the course
of the practical sessions over both semesters. *Empirical assessments* were done by means of tests administered to both groups of students during the practical sessions, three tests per semester.

1.6.5 Data analysis

The types of data collected during the research warranted the use of both quantitative and qualitative data analysis techniques.

Qualitative data collected and recorded through semi-structured interviews was methodically transcribed into text (MSWord). The interviews were conducted in English. The researcher used a coding framework developed by Saldana (2009) to transform key words/concepts into themes and categories. Qualitative thematic coding and hermeneutics were combined to form a meaningful, interpretative, descriptive tool to analyse the data collected from interviews and transcribe into text.

For quantitative data collected (assessment of tests written by the students), the researcher conducted two *t-tests*, one for the pair-programming and one for solo-programming groups respectively, and compiled graphs and tables to draw relevant conclusions from the analysed data.

1.7 DELINEATION

For the purpose of this study, the research participants were delineated to Information technology students registered at a purposively selected university of technology in Gauteng. The students were enrolled for the *Information Systems* software development module in both semesters of 2013.

1.8 ETHICAL CONSIDERATIONS

All the students at the selected HEI in Gauteng who were registered for the *Information Systems* module in 2013 partook in this research study. The HEI was selected through convenience sampling due to the researcher’s affiliation to the HEI. The researcher was the lecturer for the *Information Systems* students at this HEI in 2013. Pair-programming was introduced to one group of the *Information Systems*
students, while the second group continued with the normal solo-programming approach. The groups were selected through purposive sampling.

The researcher was actively involved throughout the research project in his capacity as observer and lecturer for both groups. Both groups worked from the same learning material and received the same projects, assignments and tests. The only difference was that the students in one group worked individually on their assignments and projects while the second group worked in pairs. The purpose of the study was explained to all participants before commencement of the research, and anonymity was guaranteed. In addition, participants were assured that the information gathered would be for research purposes only and could not be used against them. The researcher ensured that no names were mentioned while recording the interviews. The names were also blocked out in the transcripts.

1.9 CHAPTER OUTLINE

Chapter 1: Introduction, problem statement and objectives

Chapter 1 provides a background on systems development methodologies and an introduction to pair-programming. A brief investigation into how this technique has been used in tertiary institutions is undertaken. The primary and secondary research questions, translated into primary and secondary objectives, are discussed. A summary of the research design selected for this research project is provided.

Chapter 2: Literature Review

The aim of Chapter 2 is to address SRQ1, SRQ2, SRQ3 and SRQ4 through conducting a literature review. The literature review was compiled from sources including academic books and journals (hard copy and online); published and unpublished dissertations and theses; reports; conference proceedings; and scientific databases such as EbscoHost, AJOL, BASE, Google Scholar, among others. Innovative methods to systems development are discussed, with the focus on Agile Systems Development Methods (ASDMs). Pair-programming, which is categorised under the eXtreme Programming ASDM, is elaborated on. Case studies on the successful use of pair-programming in a teaching and learning environment are discussed.
Chapter 3: Research Methodology

Chapter 3 outlines the research methodology, based on the ‘Research Onion’ concept of Saunders et al. (2009). It includes the research philosophy, paradigm, approach and strategy. The research participants, data collection and analysis techniques are discussed. The chapter closes with a summarised table outlining an apposite research methodology that can be used to explore the effect of pair-programming on the academic progress of software development students at HEIs in South Africa.

Chapter 4: Analysis and discussion

The aim of Chapter 4 is to explore the impact of implementing an agile programming approach, specifically pair-programming, on the enjoyment level, attitude and academic performance of IT students at HEIs in South Africa. It addresses SRQ2, SRQ3 and SRQ4, and secondary objectives ii), iii) and iv) by implementing the research design and discussing the research analysis. After having analysed the data gathered through interviews, observations and assessments, the results indicate that an innovative agile programming approach has a positive impact on the academic progress of software development students at HEIs in South Africa.

Chapter 5: Findings and recommendations

Recommendations and findings are stated in this chapter. The viability of using pair-programming in the teaching and learning process of software development students at HEIs in South Africa is compared to that of using the single-student (solo) programming approach. Conclusions are drawn on whether the use of pair-programming has an effect on the students’ enjoyment level of programming and an improvement in the academic performance of students. A model for pair-programming at HEIs in SA is proposed, thus addressing SRQ5 and objective v), and recommendations are made, which addresses SRQ6 and objective vi).

1.10 SUMMARY

Table 1.3 provides a summary overview of the research questions and objectives, as well as the chapters containing and addressing these questions and objectives.
### Table 1.3: Summary overview of research questions and objectives

<table>
<thead>
<tr>
<th>Primary Research Question</th>
<th>Secondary Research Questions</th>
<th>Secondary Research Objectives</th>
<th>Chapters</th>
</tr>
</thead>
</table>
| **PRQ:** How does pair-programming as agile software development method shape the experience of tertiary level IT students with regard to their academic performance in developing software? | **SRQ1:** How prominent is eXtreme Programming (XP), specifically pair-programming, as educational tool at Higher Education Institutions (HEIs) in general? | i) To determine if eXtreme Programming (XP), specifically pair-programming, is used as educational tool at Higher Education Institutions (HEIs) in general, and if so, how prominent XP is. | Defined in Chapter 1  
Addressed in Chapter 2  
(Theory)  
Summarised in Chapter 5 |
| **PRQ:** The primary objective is to establish whether the use of pair-programming contributes significantly towards improving the academic performance of tertiary level IT students in SA. | **SRQ2:** What is the impact of pair-programming on IT students’ enjoyment level of software development? | ii) To determine the impact of pair-programming on IT students’ enjoyment level of software development. | Defined in Chapter 1  
Addressed in Chapter 2  
(Theory)  
Addressed in Chapter 4  
(Research Design)  
Summarised in Chapter 5 |
| **SRQ3:** How does pair-programming impact the academic progress of IT students at HEIs in South Africa with regard to software development? | **SRQ3:** How does pair-programming impact the academic progress of IT students at HEIs in South Africa with regard to software development? | iii) To determine how pair-programming impacts the academic progress of IT students at HEIs in South Africa with regard to software development. | Defined in Chapter 1  
Addressed in Chapter 2  
(Theory)  
Addressed in Chapter 4  
(Research Design)  
Summarised in Chapter 5 |
| **SRQ4:** How does pair-programming impact the attitude of IT students towards software development? | **SRQ4:** How does pair-programming impact the attitude of IT students towards software development? | iv) To determine the impact of pair-programming on the attitude of IT students towards software development. | Defined in Chapter 1  
Addressed in Chapter 2  
(Theory)  
Addressed in Chapter 4  
(Research Design)  
Summarised in Chapter 5 |
| **SRQ5:** What model can be proposed to shape the experience of tertiary level IT students with regard to their academic performance in developing software? | **SRQ5:** What model can be proposed to shape the experience of tertiary level IT students with regard to their academic performance in developing software? | v) To propose a model that can be used to shape the experience of tertiary level IT students with regard to their academic performance in developing software. | Defined in Chapter 1  
Addressed in Chapter 5 |
| **SRQ6:** How can pair-programming be implemented optimally in a controlled learning environment? | **SRQ6:** How can pair-programming be implemented optimally in a controlled learning environment? | vi) To determine if and how pair-programming can be optimally implemented in a controlled learning environment. | Defined in Chapter 1  
Addressed in Chapter 5 |
CHAPTER 2: LITERATURE REVIEW

Figure 2.1: Graphical representation of Chapter 2
2.1 INTRODUCTION

The aim of Chapter 2 is to address SRQ1, SRQ2, SRQ3 and SRQ4 (see section 1.4.2) through conducting a literature review. The chapter is broadly divided into two sections. The first part of the literature review focuses on IT students’ attitudes, perceptions and enjoyment of development software. The second half of the chapter provides a background of the learning outcomes for an IT qualification at HEIs in South Africa, the rapid changes in technology, and the skills set IT students need to keep up with the advances in technology. Innovative methods for systems development are discussed, with the emphasis on pair-programming.

A case study conducted by Chigona and Pollock (2008) on students’ attitude towards pair-programming, are elaborated on. The outcomes of a case study conducted by Williams (1999) on the effect of pair-programming on the academic performance of students are discussed. The advantages of pair-programming are indicated, and the chapter closes with a summary of the outcomes obtained through literature.

2.2 IT STUDENTS IN A SOFTWARE DEVELOPMENT ENVIRONMENT

The primary learning outcomes for an IT qualification focus on the design and development of feasible computer-based solutions to specified problems using programming (i.e. software development) (Conradie 2013).

According to Zoghbi and Kumar (2009) software development is often regarded as a course that students should know and understand by themselves. The most significant barrier to software development, especially with first year students, is the fact that most students have never written a single code before entering a HEI (Zoghbi & Kumar, 2009). Because the traditional way of teaching students how to program is not sufficient, it leads to students not always enjoying the course.

2.2.1 Enjoyment of software development

Many students display feelings of fear towards programming. According to Gomes and Mendes (2007), the origin of these feelings could be attributed to the fact that most first year students are exposed to software development for the first time at tertiary level, having no programming experience at all. Furthermore, those who do
have prior experience of programming may be confronted with a high level of expectation without sufficient supervision by the lecturer. Gomes and Mendes (2007) argue that the instability and change experienced by students in tertiary institutions probably give rise to a negative feeling towards software development.

2.2.2 Attitude towards software development

Howard (2006) found that as programming becomes more complex during a software development course, more students tend to become frustrated. This frustration results in a snowball effect where students develop a negative attitude towards programming and traditional ways of teaching programming because they are not afforded the opportunity of a collaborative approach where they have enough time to discuss and share code under the supervision of the instructor.

2.3 INNOVATIVE AGILE SOFTWARE DEVELOPMENT METHODS

Defining the term *Systems Development Methodology* is not a clear-cut task. Definitions range from simple to complex, without a universally accepted and concise definition (Conradie & Huisman, 2012; Livari, Hirscheim & Klein 1999). An often used definition is:

“Systems development methodology is a collection of procedures, techniques, tools and documentation aids that assist systems developers in their efforts to implement a new information system” (Conradie & Huisman 2012).

The term *Agile Software Development Methodology* (ASDM) refers to specified methodologies that share the standards and norms as stated in the *Agile Manifesto* (Beck 2000), which highlights twelve principles through which methodologies can be ascertained to be agile.

“In order for a methodology to be deemed agile, the most important characteristic is the ability to adapt quickly to change. This adaptability is achieved through the techniques and tools of the particular methodology” (Beck 2000).

Agile programming is based on the premise that more advanced software can be developed by iterative and incremental software development methodologies which
include eXtreme Programming (XP), Dynamic Systems Development Method (DSDM) and Feature-Driven Development (FDD).

According to Beck (2000), the use of XP in Industry has been claimed to provide significant benefits and there seems to be potential in the use of this methodology for student projects. Furthermore, the use of XP is common in most fields of innovative software development (Adams et al. 2003). In a study done by Zhang (2010), it was found that the manufacturing industry widely accepts agility as a new and competitive concept. Beck (2000) identifies twelve principles through which an agile methodology can be recognised. These principles are summarised in Table 2.1:

**Table 2.1: Summary of Beck’s (2000) twelve principles of agility**

<table>
<thead>
<tr>
<th>PRINCIPLES FOR AGILE METHODOLOGY</th>
</tr>
</thead>
<tbody>
<tr>
<td>i) The ultimate priority is customer satisfaction with early and continuous software.</td>
</tr>
<tr>
<td>ii) Amended requirements are always welcomed. Change through agility equals competitive advantage for customers.</td>
</tr>
<tr>
<td>iii) Frequent delivery with a shorter time scale of working software.</td>
</tr>
<tr>
<td>iv) It is essential that developers and business people work together throughout the entire project.</td>
</tr>
<tr>
<td>v) Motivated developers should be afforded the backing and setting they need, and then trusted to get the work done.</td>
</tr>
<tr>
<td>vi) The most effective method of information transfer to and within a development team is face-to-face dialog.</td>
</tr>
<tr>
<td>vii) The most important measure of progress is working software.</td>
</tr>
<tr>
<td>viii) Agile processes advance maintainable innovative development. It is essential for developers, sponsors and end-users to sustain a persistent pace for as long as it takes.</td>
</tr>
<tr>
<td>ix) By steadfastly concentrating on technical excellence and sound design, agility is enhanced.</td>
</tr>
<tr>
<td>x) Simplicity has to be maintained always.</td>
</tr>
<tr>
<td>xi) For the design, development and delivering of the best prerequisites and architectures, self-organisation is essential.</td>
</tr>
<tr>
<td>xii) The development team regularly has to reconsider their efficiency and then adjust their performance accordingly.</td>
</tr>
</tbody>
</table>

The agile manifesto of Beck (2000) expresses many of the defining characteristics of agile systems development methodology as seen in the table 2.2.
Table 2.2: Profile of agile development models
(Source: Beck 2000)

<table>
<thead>
<tr>
<th>Category</th>
<th>Specifics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evolution of goals</td>
<td>Increase responsiveness and decrease turnaround time on development decisions</td>
</tr>
<tr>
<td>Methodology</td>
<td>Cumulative development of operational software; pair-programming</td>
</tr>
<tr>
<td>Technology</td>
<td>UML tools</td>
</tr>
<tr>
<td>Critical factors</td>
<td>Individual capability and reciprocal trust</td>
</tr>
<tr>
<td>Interdisciplinary effects</td>
<td>Human interactions and management practices</td>
</tr>
<tr>
<td>Behavioural considerations</td>
<td>Intense and close people relations required</td>
</tr>
<tr>
<td>Nature of the problem</td>
<td>General</td>
</tr>
<tr>
<td>Application domain</td>
<td>Smaller projects</td>
</tr>
</tbody>
</table>

According to Johnson, Johnson and Smith (1998), one of the most important reasons for using an agile approach in innovative programming is grouping students together in teams whereby collaborative or cooperative learning is applied. Collaborative learning used in agile programming is an approach to group work that maximises the learning and satisfaction resulting from working as part of a high-performance innovative team. Relative to students being taught traditionally with instructor-led lectures, individual assignments focus on concepts which have limited opportunities for students to practice programming skills (Johnson et al. 1998).

Johnson et al. (1998) continue by explaining that for many students, especially beginners and those without the relevant background, it is not easy to learn programming concepts and languages. Students who are taught through the use of agile programming tend to learn collaboratively because they are working in a group. Cooperatively taught students tend to have lower levels of stress and anxiety, a higher self-esteem, a more in-depth understanding of learning material, and greater inherent motivation to learn and achieve. They also display higher academic achievement, advanced high-level thinking and analytical skills, with a more positive attitude toward programming subjects.

Agile development is dominated by two objectives which make the development team more effective (Cockburn & Highsmith 2001). The two objectives are to reduce the expenses contained in transferring the information between people involved, and to reduce the time it takes between making decisions. Dagnino (2002) observes that
the characteristics making a development methodology more agile include methods to reduce risk, interfacing or collaboration with clients, participation of people, interactive and real-time development, a team that is able to adjust and adapt, thorough emphasis on developing software that works, and continuous testing.

Beck (2000) identifies four of the most generally known agile methodologies as i) Agile Unified Process (AUP); ii) Adaptive Software Development (ASD); iii) Dynamic Systems Development Method (DSDM); and iv) eXtreme Programming (XP). Of these, XP is considered the most popular. The purpose of XP was to fulfil a need for a faster, simpler and cheaper way to design software. Each of the four agile methodologies is discussed next.

2.3.1 Agile Unified Process (AUP)

“The Agile Unified Process is a hybrid modeling approach created by Scott Ambler when he combined the Rational Unified Process (RUP) and agile methods” (Christou, Ponis & Palaiologou 2010).

According to Ambler (2005), AUP is a process that is repetitive and cumulative. It consists of workflows, and all the projects have to follow four phases. During the Inception phase, the developers and customers meet for a discussion on the scope of the project where initial requirements are collected and divided into separate tasks. The elaboration phase is where the development plan is compiled and the team is formed. The developers and customers work together to compile this plan. The team dissects each task and establishes an estimated time frame for each task to be implemented. The construction phase is ushered in when the work commences and continues until all the tasks are successfully completed. This phase is filled with iterations, i.e. a series of steps performed over a short period of time to complete the tasks. This development phase will continue until the customer is satisfied with the product. During the transition phase, the end-product is delivered and ready for use by the customer, and moved to the support phase. If the customer wants to add some requirements, the process has to be restarted afresh.

Ambler (2005) explains that the phases include identifying the stakeholders, understanding the problem of the users and outlining the user interface for the system. AUP contains seven workflows and each of these has four phases. The AUP
workflows consist of “model, implementation, test, deployment, configuration management, project management, and environment” (Ambler 2005) (figure 2.2).

![Figure 2.2: Agile Unified Process phases](Source: Ambler 2005)

### 2.3.2 Adaptive Software Development (ASD)

Jim Highsmith and Sam Bayer formally defined Adaptive Software Development (ASD) in 2000. ASD focuses on rapid creation and evolution of software systems (Highsmith 2000). It offers solutions for the development of large and complex software systems through an iterative development process, with constant prototyping. The development process involves “product initiation, adaptive cycle planning, quality review and final quality assurance” (Highsmith 2000).

Pressman (2009) attributes the following characteristics to ASD: purpose-directed planning; component-centredness; purposive risk consideration; emphasis on ‘learning within the process and cooperation for requirement gathering; and the employing a technique called time-boxing where a task is divided according to a specific time slot and each task has its own budget.

### 2.3.3 Dynamic Systems Development Method (DSDM)

The Dynamic Systems Development Method (DSDM) aligns each project to clear strategic objectives and focus on early delivery. DSDM has eight guiding principles which distinguish it from XP and ASD (Pressman 2009).
These principles are: i) active user involvement is essential; ii) empower DSDM teams to make informed decisions; iii) focus on regular product delivery; iv) fitness for business purpose is a crucial benchmark norm for acceptance of deliverables; v) apply iterative and incremental development in order to deliver accurate business solutions; vi) reversibility of all changes encountered during development; vii) requisites are base-lined at a high level; and viii) incorporation of testing throughout the life-cycle (Pressman 2009).

According to Voigt (2004), The DSDM development process consists of seven phases, namely pre-project, feasibility study, business study, functional model iteration, design and build iteration, implementation, and post-project. Each phase has several tasks. However, a phase can be modified to include more tasks, which might be required during the development process as shown in figure 2.3.

![DSDM phases](source: Voigt 2004)

2.3.4 eXtreme Programming (XP)

A systems development methodology consists of phases and sub-phases which direct the systems developer in technique choices suitable for systems development. It also helps developers with the planning, managing, controlling and assessment of
information systems projects (Avison & Fitzgerald 2003). Pair-programming is a technique used in eXtreme Programming (XP). Munro (2003) identifies the following five practices or techniques that make XP uniquely different from other methodologies:

i) **Continuous integration**: Minor changes in the code are regularly integrated into the common source base on a daily basis. Incorporation of changes, one set at a time, streamlines the integration process and makes it evident who is accountable for correcting the code when integration tests are unsuccessful.

ii) **Collective ownership**: The code and all development documents are owned by the entire team. Any member is free to modify any part of the documentation or code at any time. This method differs from the traditional one where a single developer owns a set of code. XP supporters claim that the number of bugs will decrease as the number of people who are working on a piece of code, increases.

iii) **Small releases**: The system is planned with short release cycles containing the most valuable business requirements. Typically, one cycle is less than three months and this allows the user to view and touch the working product frequently.

iv) **Testing**: Two types of tests are conducted continuously. First, unit tests ensure that classes do what developers expect them to do. These tests are typically written by the developer. Second, acceptance tests, written before the code that they will test, ensure that the system functions accordingly. These tests are derived from the customer ‘stories’ or scenarios. “All code has an associated test and new tests are added to old ones in a testing framework, creating a comprehensive test suite” (Munro 2003).

v) **Pair-programming**: Two developers program at one workstation together. Programmers switch seats periodically and regularly deliberate on each other’s code. This increases the number of people that are familiar with the source code and results in a collaborative (cooperative) learning environment.

The five principles discussed need not be followed to the letter; instead, they act as guidelines to developers during the development of systems. As a consequence, these core practices can be adapted and modified by developers to accommodate
systems development in their organisation. According to Beck (2000), the use of XP in industry has been claimed to provide significant benefits and there seems to be potential in the use of the methodology for student projects. In addition, according to Adams et al. (2003), the use of XP is common in most fields of software development. In a study done by Zhang (2010), it was found for example that agility is commonly acknowledged and recognised in the manufacturing industry as a new competitive concept.

In educational institutions, traditional programming as a rule is conducted in a computer laboratory where the lecturer focuses on syntax, reasoning, concepts and exploration of program codes through teaching and discussions. Such methods of instruction limit learning efficiency as students have restricted opportunities to practice programming skills, and lecturers are not certain if the learning context brings out the best academic performance in each student (Williams et al. 2007, 2008). For many students it is difficult to learn programming, especially those who have not been introduced to computers and programming before. The use of shared programming activities into a teaching and learning environment may contribute to addressing this problem. Furthermore, possible benefits include students benefiting from one another’s resources and skills, assessing one another’s ideas and monitoring one another’s work. However, Williams et al. (2007, 2008) highlight that these premises must be further tested in different educational environments with different levels of students.

2.4 PAIR-PROGRAMMING

XP is a systems development methodology that was developed to fulfil a need for a faster, simpler and cheaper way to design software (Beck 2000). One of the twelve techniques, also referred to as practices, as stipulated by Beck, is pair-programming.

Pair-programming is a technique used in XP where two developers—in reference to this study, two students—program together at one workstation. The students switch seats periodically, generally every 10-15 minutes, one being the coder and the other being the quality controller or advisor. Discussion, collaboration and cooperation between coder and advisor are encouraged.
A study conducted by Chigona and Pollock (2008) shows positive reactions of students towards pair-programming. The average scores of the pair-programmers were higher than the average scores of the solo-students for both the first and second assignment. In the first pair-programming assignment the pair-programmers obtained an average score of 94.86% while the solo-students scored an average of 82.46%. In the second pair-programming assignment the pair-programmers averaged a score of 90% while the solo-students scored a 77.38% average (Chigona & Pollock 2008), as shown in figure 2.4.

![Bar chart showing student performance in pair-programming assignments](image)

**Figure 2.4: Student performance in pair-programming assignments**  
(Source: Chigona & Pollock 2008)

Based on the results in figure 2.4, it is clear that the students in pairs submitted higher quality work and received higher marks. This indicates that pair-programming indeed contributed to improved work of students who took part in the study. It is therefore reasonable to propose that pair-programming is likely to assist in improving the academic performance and quality of the work of tertiary level IT students.

The students also confirmed that pair-programming contributed to improving the quality of their work by developing better projects of higher quality with fewer errors, as shown in table 2.3 (Chigona & Pollock 2008).
Table 2.3: Students’ views towards pair-programming (Summary)  
(Source: Chigona & Pollock 2008)

<table>
<thead>
<tr>
<th>Likert Scale: 1 = Agree, 3 = Neutral, 5 = Disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Quality</strong></td>
</tr>
<tr>
<td>I find that pair-programming develops better projects than programming by myself</td>
</tr>
<tr>
<td>The quality of the work we produced was better because we pair-programmed</td>
</tr>
<tr>
<td>More errors were found and fixed when we pair-programmed</td>
</tr>
<tr>
<td><strong>Productivity</strong></td>
</tr>
<tr>
<td>The pair-pressure helped me work better; I could not let my partner down</td>
</tr>
<tr>
<td>The work was finished quicker because of the pair-programming</td>
</tr>
<tr>
<td><strong>Enjoyment</strong></td>
</tr>
<tr>
<td>I was more confident in the work when we pair-programmed</td>
</tr>
<tr>
<td>I enjoy working in a pair-programming team</td>
</tr>
<tr>
<td>I find pair-programming to be more successful than programming by myself</td>
</tr>
<tr>
<td>I enjoyed the work more because of the pair-programming</td>
</tr>
<tr>
<td>If I had the choice I would work in a pair-programming team again</td>
</tr>
<tr>
<td><strong>Knowledge</strong></td>
</tr>
<tr>
<td>I learnt more from doing the work because of the pair-programming</td>
</tr>
<tr>
<td>Between my pair-programming partner and I, we can figure everything out</td>
</tr>
</tbody>
</table>

Based on the results in table 2.3, it is reasonable to propose that pair-programming positively influences the attitude and enjoyment level of students towards programming.

A study conducted by Williams (1999) shows that pair-programming enhances the academic performance of the student. In his experimental classes, the students completed four assignments. Thirteen individuals (solo-programming) and fourteen collaborative pairs (pair-programming) completed each assignment. Williams (1999) states that the paired students continuously passed more of the automated post-development test cases run by an impartial teaching assistant as shown in table 2.4.

Table 2.4: Percentage of test cases passed  
(Source: Williams 1999)

<table>
<thead>
<tr>
<th>Programmers</th>
<th>Individuals</th>
<th>Pair-programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 1</td>
<td>73.4%</td>
<td>86.4%</td>
</tr>
<tr>
<td>Program 2</td>
<td>78.1%</td>
<td>88.6%</td>
</tr>
<tr>
<td>Program 3</td>
<td>70.4%</td>
<td>87.1%</td>
</tr>
<tr>
<td>Program 4</td>
<td>78.1%</td>
<td>94.4%</td>
</tr>
</tbody>
</table>
Williams (1999) indicates that in the competitive markets of today, producing quality software as fast as possible is a competitive advantage; it can even signify survival. In order to produce quality software fast, pair-programming seems a viable option to pursue, as indicated in figure 2.5. Based on the results in table 2.4 and figure 2.5, it is clear that pair-programming increases the pass rate of students and enable paired students to deliver a high quality project in a short time.

![Elapsed Time](image)

**Figure 2.5: Elapsed time spent on the project**  
(Source: Williams 1999)

### 2.4.1 How does pair-programming work?

For this research study, pair-programming will be investigated by focusing on the steps proposed by Williams and Kessler (2000). The researcher finds these steps useful and seemingly easy to understand and implement in a programming class situation. The steps are summarised as follow:

- **Share everything:** Both programmers share the entire application. Both are responsible for the success of the application. One is typing code and another one is reviewing, and they have to switch roles periodically.

- **Play fair:** Both programmers contribute to the success of the application. This is possible because both programmers switch roles periodically. The person
who is reviewing the code should not be a passive reviewer, instead, s(he) is always active and engaged by thinking strategically and checking if the coding process is heading in the right direction.

- **Do not be aggressive towards your learning partner:** Both programmers should be focused on the task without causing unnecessary difficulties. By working in a team, both learning partners develop a collaborative environment where they have to share their time in doing the required task rather than spending most of the time communicating via email. Sharing their time on the task increases their focus and leads to a higher quality application.

- **Restore thoughts to where they belong:** When performing solo-programming, negative thoughts might cross one’s mind, for example, “I am a bad programmer”, or “I cannot do it”. A major advantage of pair-programming is the continuous discussion between the two learning partners to formulate thoughts and encourage each other. Obtaining feedback from a peer restores one’s thoughts to a positive state and increases confidence.

- **Clean up your “mess”**: Pair-programming assists to efficiently clean up the mess. In this case “mess” means the mistakes in the coding. With pair-programming, the unnoticed mistakes made by the driver become noticed by the reviewer who is monitoring the coding process.

- **Do not be overly serious:** Pair-programming can work effectively if the self-esteem of both programmers (learning partners) are curbed. For constructive exchange of ideas or debate and reviewing of code to occur, both programmers should have balanced egos.

- **Be aware of the strength of two brains:** Each programmer enters the team with his or her own unique skills and knowledge. Both programmers combine their unique skills to have a larger subset of skills and knowledge. This subset of skills and knowledge will become common between the two programmers and assist both to deliver a high quality task.

Pair-programming leads to a collaborative environment by creating effective communication between the two programmers. Without much effort, learning partners will have discussions and make combined decisions to improve their work. According
to Hargrove (1998), “collaborative people see others not as creatures who force them to compromise, but as colleagues who can help them augmenting their talents and skills”.

2.4.2 Pair-programming as collaborative learning approach

As already mentioned, the nature of software development has dramatically changed over the last few years. To adapt to these rapid changes, there is a need for educated lecturers and students who have a set of skills that go beyond simply using IT as a tool. Learning how to develop computer-based solutions is a complex task that requires more conceptual understanding and mastering than just memorising and interacting with the computer.

These skills might be enhanced or developed by introducing a collaborative pedagogical approach. Lovat (2003) defines a pedagogical approach as a framework that specifies the teaching and learning processes between lecturers and students. A collaborative pedagogical learning approach is viewed as a motivation for individual cognitive development through a capability to enhance collaborative learning among lecturers and students (Vygotsky 1978).

Collaborative (cooperative) learning is a pedagogical approach in which cooperation between students are encouraged. Forming part of the constructivism paradigm, it is postulated that students’ performance is improved when students work together to obtain module outcomes. In a learning environment, module outcomes relate to specific practical computer programming outcomes that students must obtain.

In pair-programming, a team can be formed at any time with two or more members (students) working together towards a shared goal. Once a team has grown larger than four members, it is recommended to divide the team into sub-teams where all are working towards the same goal, although sub-teams will not necessarily be in communication with the other sub-teams. Various limitations can however have an influence on the size of the group. For effective team work, it is important to recognise that there are specific steps to follow, and that the tasks and interpersonal behaviours of the team might change over time (MacPherson 2000).
Dr. Bruce Tuckman developed a model on how teams advance and display behaviours around their interpersonal interactions and the assignment being carried out. The model consists of five stages, namely forming, storming, norming, performing and adjourning (MacPherson 2000).

The *forming stage* is specifically for task organisation and orientation. The task(s) as well as information about these tasks are ascertained. The question to be answered is: “What is the assignment of the group and how will I be able to contribute to that assignment?”

During the *storming stage*, individual team members display emotional reactions towards their team. The assignment specifications will spark one part of this response. The more complex the assignment appears relative to an individual’s self-perceived capabilities, the greater the possibility of a ‘storm’. The question to be answered is: “Am I emotionally ready to deal with this task?” A wide range of understandings of the assignment and roles are voiced or become obvious.

During the *norming stage*, communication becomes more open and is progressing. Exchange of information and sharing of ideas and views are happening. The focus is on the assignment and members are answering the following question: “What do I have that will help us accomplish this assignment?” Feasible and practical guidelines are drafted. There is a feeling of accord and people are looking at, “how can I help contribute to group unity?”

For the *performing phase*, the attention of all the team members is on positive and productive action directed towards effective accomplishment of the assignment on hand. Interactive and assignment behaviour with collective insight begin to emerge and the focus is on functionality. Problem-solving is primarily directed towards the assignment and deliverable(s).

*Adjourning* commences when teams have accomplished their assignments, they conclude and then proceed to other teams elsewhere. It is essential that the team take the time to view its process one last time. “What went well?” “What could we do better in another situation so that the loose ends are wrapped?” The wrapping up of the interpersonal behaviours includes an opportunity to say thank you and good-bye to fellow team members.
According to MacPherson (2000), “closure is a final essential part of the team process”. Thus, in order to have a positive result, it is necessary to realise that each stage builds upon the previous one; each stage prepares for the performing stage. Any attempt to bypass a stage influences accomplishment negatively; and with every new challenge, the process repeats.

Collaborative learning is effective because it stimulates socio-cognitive conflicts due to diverse views and approaches applied by the learning partners (Doise & Mugny 1984; Perret-Clermont 1991). Williams and Upchurch (2001), Williams and Kessler (2001) and Williams et al. (2002, 2006, 2007, 2008) suggest that pair-programming, as a strategy used in eXtreme Programming (XP) and classified as an Agile Systems Software Development Methodology (SDM), displays several promising properties for educational purposes. Pair-programming, whereby two programmers (i.e. students) work at one computer on the same programming task, appears to have a positive effect on students at universities (Ho et al. 2004; Werner et al. 2004), specifically in terms of enjoyment (McDowell et al. 2006; Werner et al. 2004; Ho et al. 2004), and on the students’ view of the importance or usefulness of the subject (McDowell et al. 2006; Werner et al. 2004).

2.4.3 Advantages of pair-programming

Researchers have proven that pair-programming displays benefits over traditional programming in Higher Education. According to Williams et al. (2007), pair-programming increases quality of product, reduces the time to complete the task, and increases the academic performance of the students and the enjoyment of student towards programming.

2.4.3.1 Enjoyment

Students who work in pairs enjoy programming more than those who do solo-programming and therefore they are happier and less frustrated than solo programmers (McDowell, Hanks & Werner 2003; McDowell et al. 2006; Williams & Upchurch 2001; Bishop-Clark, Courte & Howard 2006; Cliburn 2003).
2.4.3.2 **Confidence**

Students who work in teams develop confidence in their programming team and get more satisfaction than students who work alone (solo-programming) (McDowell *et al.* 2003; McDowell *et al.* 2006; Bishop-Clark *et al.* 2006; Hanks, McDowell, Draper & Krnjajic 2004). According to Thomas, Ratcliffe and Robertson (2003), students who develop confidence, enjoy programming.

2.4.3.3 **Program quality**

The quality of the applications delivered by paired students is distinctively higher than the applications delivered by solo-programmers (McDowell *et al.* 2003). Students working in teams are able to produce an application of a higher quality which is less complex and easier to read (Bipp, Lepper & Schmedding 2008), and shorter and easier to understand and extend (Williams & Kessler 2001; Thomas *et al.* 2003; Jensen 2005; McDowell, Werner, Bullock & Fernald 2002).

2.4.3.4 **Student academic performance**

Working in pairs improves the academic performance of the student. It is consistent with collaborative learning research which demonstrates that the academic performance of the student is improved when an individual is learning with others (Bevan, Werner & McDowell 2002; McDowell *et al.* 2002). Braught, Eby and Wahls (2008) found that pair-programming seems to enhance the level of the individual programming skills since lower achieving students are able to achieve higher scores through pair-programming.

2.4.3.5 **Attitude**

Students working in pairs develop a positive attitude towards programming and show positive responses to working with a learning partner in collaborative programming (pair-programming) (Nagappan 2003; Howard 2006). Howard (2006) found that as the programs become more complex during the course, the more the students' attitudes and appreciation of pair-programming increased.
2.4.3.6 Communication

Students in pairs display a high level of interaction with each other. They discuss aspects related to the programming project; they direct and guide each other to solve the problem (Williams et al. 2002). The pair learns to discuss their ideas and work together, which improves communication, teamwork and effectiveness (Williams & Kessler 2001).

2.5 CONCLUSION

Programmers have generally been used to work alone to develop code due to the educational system of individual performance. However, pair-programming breaks down some personal barriers, such as developing code for software applications alone by introducing collaborative learning through working in a team. This new method of programming creates intercommunication between programmers by sharing their work and accepting recommendations made by a teammate to improve their own skills and produce high quality work.

2.6 SUMMARY

This chapter addressed the primary research objective through four secondary research objectives.

The primary research objective of this literature research is to confirm from the current body of knowledge whether pair-programming plays a significant role towards improving the academic performance of IT students at HEIs in South Africa.

Secondary objectives

The theoretical outcomes (answers) to each of the four secondary research objectives, based on the primary research objective, are stated below.

Objective i): To determine how prominent eXtreme Programming (XP), specifically pair-programming, is as educational tool at Higher Education Institutions (HEIs) in general
Outcome 1: Pair-programming is a systems development methodology created by Kent Beck in 1996 (see Section 2.4).

Outcome 2: Pair-programming is a technique used in eXtreme Programming where two students or two developers program at one workstation (see Section 2.3.4).

Outcome 3: In a study conducted by Chigona and Pollock (2008), students in pairs submitted higher quality work and received higher marks than solo-programming students (see Section 2.4).

Outcome 4: With pair-programming, both programmers share the entire application, both contribute to the success of the application through switching roles periodically, and both should be focused on the task without causing unnecessary difficulties (see Section 2.4.1).

Outcome 5: For effective team work in pair-programming, it is important to recognise that there are specific steps to follow, and that the tasks and interpersonal behaviours of the team might change over time (See Section 2.4.2).

Objective ii): To determine the impact of pair-programming on IT students’ enjoyment level of software development

Outcome 6: Paired students enjoy programming more than solo students and paired students are more confident and less frustrated (see Section 2.4 and table 2.3).

Outcome 7: Pair-programming develops communication skills between both learning partners and enables students to socialise (see Sections 2.4.3 and 2.4.1).

Objective iii): To determine if there is a significant distinction between the academic progress of students who use pair-programming and those who do not
Outcome 8: On average, paired students achieve significantly higher results than those working alone (solo-programming) (see Section 2.4.3.4).

Outcome 9: Paired students develop programs with high functionality and more readable programming code (see Section 2.4 and figure 2.5).

Outcome 10: Programs or code compiled by paired students are significantly more descriptive than programs compiled by solo students for the same projects, and the paired students received higher marks than solo students (see Sections 2.4; 2.4.3.3 and figure 2.4).

Outcome 11: The quality of the applications delivered by paired students is distinctively higher than the applications delivered by solo-programmers (See Section 2.4.3.3).

Objective iv): To establish whether pair-programming positively influences the attitude of software development students, thereby influencing a future career path in programming

Outcome 12: Paired students develop a positive attitude towards programming and working in a team (See Section 2.4.3.5).

Outcome 13: Paired students are more confident in programming than solo students (see Section 2.4.3.2).
CHAPTER 3: RESEARCH METHODOLOGY

Figure 3.1: Graphical representation of Chapter 3
3.1 INTRODUCTION

This chapter discusses the research philosophy, paradigm, approach and strategy which form the core for selecting an appropriate research design and methodology to explore the effect of pair-programming as educational tool on the academic performance of Information Technology students at Higher Education Institutions (HEIs) in South Africa, and whether this tool is able to contribute towards enhancing the academic performance of these students.

The methods used to collect data as well as the sampling techniques used to select the participants, are elaborated on. The methods used in analysing and presenting the findings, are also discussed.

3.2 MEANING OF RESEARCH

Defining the term research is not a clear-cut task. Definitions range from the simple to the complex and it varies according to the authors. According to Singh (2006), the term “research” is composed of two words: Re + Search. “Re” means repetitively (again and again) and “Search” means to find out something.

![Figure 3.2: Meaning of research](source: Singh 2006)

By applying common sense, research can simply be defined as the search for answers to certain questions or problems or phenomena through a planned process which includes the collection, analysis and interpretation of data and drawing conclusions. Definitions of research include:

“Research is the systematic and scholarly application of the scientific method interpreted in its broader sense, to the solution of social studies problems; conversely, any systematic study designed to promote the development of social studies as a science can be considered research” (Mouly 1970).
“Research is considered to be the more formal, systematic, intensive process of carrying on the scientific methods of analysis. It involves a more systematic structure of investigation, usually resulting in some sort of formal record of procedures and a report of results or conclusions” (Best 1977).

“Research comprises defining and redefining problems, formulating hypotheses or suggested solutions; collecting, organising and evaluating data; making deductions and reaching conclusions; and at last carefully testing the conclusions to determine whether they fit the formulating hypothesis” (Best 1977).

Thus, research is the logical process which assists the researcher to answer to the following questions: what, why, how, who and when:

- What = Research problem
- What, Why = Research questions and objectives
- Who, When = Sample of the study and duration of the study
- How = Methodology, collection of data
- Why = Interpretation of the results

3.3 RESEARCH PROBLEM

Students tend to be frustrated when they are introduced to programming courses due to lack of knowledge in solving problems involving various programming languages and technologies (Henson 2002; McMahon 2009). This often results in students dropping out of the course because they are struggling on their own without any individual attention or guidance (University World News 2015).

One proposed solution to address this problem in South African higher education institutes is the introduction of a collaborative pedagogical agile approach in the form of a pair-programming model, in which students develop software in teams in a controlled learning environment according to a structured format.

The study endeavours to establish how pair-programming can shape the experience of tertiary level IT students in South Africa with regard to their academic performance in developing software.
3.4 RESEARCH DESIGN

Research design is the logical map of the research process or an architectural framework that outlines how the study is to be carried out (Mouton 1996). It provides a plan of how all of the main aspects of the research such as research philosophy, paradigm, methodology, strategy, technique, participants and data analysis work together to respond to the research questions or to a research problem. Figure 3.3 shows the layers of research design as indicated by Saunders et al. (2009).

![Research Onion Diagram](image)

**Figure 3.3: Research onion**  
(Source: Saunders et al. 2009)

According to Mouton (1996), the purpose of research design is to plan, construct and conduct the research so that the validity of the findings is maximised. Yin (2003) adds to this by stating that:

“…colloquially a research design is an action plan for getting from here to there, where ‘here’ may be defined as the initial set of questions to be answered and ‘there’ is some set of answers”.

Research design gives direction from the underlying philosophical assumptions throughout data collection to results.
3.4.1 Research philosophy

Research is underpinned by the philosophical assumptions which show the particular way in which the world is viewed and understood (Saunders et al. 2009). When conducting research, it is imperative to reflect on the two parameters that constitute the research philosophy, namely ontology and epistemology. These parameters shape the manner in which the research is conducted, from design to conclusion. It is therefore important to understand the parameters in order to select the correct approach and to ensure that the researcher adopts a suitable method to conduct the research. Figure 3.4 describes the framework of the research philosophy as indicated by Slife and Williams (1995).

![Figure 3.4: A framework for research philosophy](Source: Slife & Williams 1995)

Raddon (2015) agrees with Saunders et al. (2009) that, when conducting research, it is important to consider the two components that constitute the research philosophy, namely ontology (“what constitutes valid knowledge and how can we obtain it?”) and epistemology (“what constitutes reality and how can we understand existence?”).

3.4.1.1 Ontology

According to Blaikie (1993), the core definition of ontology is:
“...the science or study of being and develops this description for the social sciences to encompass claims about what exists, what it looks like, what units make it up and how these units interact with each other”.

In short, “ontology is concerned with the nature of reality” (Saunders et al. 2009). It questions the assumptions the researcher makes regarding the manner in which the world operates. Ontology is influenced by two stances, namely objectivism and subjectivism.

“Objectivism portrays the position that social entities exist in reality external to social actors concerned with their existence, while subjectivism holds that social phenomena are created from the perceptions and consequent actions of those social actors concerned with their existence” (Blaikie 1993).

This research study is aligned with a subjectivist ontological stance which implies that a situation observed can only come into existence through the action of humans in creating and recreating the phenomena observed (Orlikowski & Baroudi 1991).

3.4.1.2 Epistemology

Epistemology is concerned with views on the most apposite ways of “probing into the nature of the world” (Easterby-Smith, Thorpe & Jackson 2008) as well as “what is knowledge and what the sources and limits of knowledge are” (Eriksson & Kovalainen 2008). Chia (2002) describes epistemology as “what to know and how it is possible to know, and the need to reflect on methods and standards through which reliable and verifiable knowledge is produced”. Hatch and Cunliffe (2006) defines epistemology as “knowing how you can know”. They expand this by asking: “How is knowledge generated, what criteria discriminate good knowledge from bad knowledge, and how should reality be represented or described?”

According to Orlikowski and Baroudi (1991), epistemology can be aligned with three possible research philosophies, namely positivism, interpretivism and critical realism.

i) Positivism

Positivists believe the validity of the knowledge is formed by empirical and verifiable proof. With positivism the researcher is not included in the process of the research (Burrell & Morgan 1979).
ii) **Interpretivism**

“Interpretive methods of research start from the position that our knowledge of reality, including the domain of human action, is a social construction by human actors and that this applies equally to researchers” (Walsham 1993).

The interpretivists support the idea that the reality is constructed by social actors. Interpretivism promotes the idea that the researcher needs to understand variances between humans in our role as social actors. This highlights the difference between performing research among people and conducting research on objects such as cars and computers. The aim of interpretivism is not to generalise the population, but to provide a better understanding of how people obtain knowledge in a particular social setting (Neuman 2011).

Pizam and Mansfeld (2009:1, as cited by Dudovskiy 2015) differentiate between positivism and interpretivism in Table 3.1.

**Table 3.1: Positivism and Interpretivism**

<table>
<thead>
<tr>
<th>Assumptions</th>
<th>Positivism</th>
<th>Interpretivism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nature of reality</td>
<td>Objective, tangible, single</td>
<td>Socially constructed, multiple</td>
</tr>
<tr>
<td>Goal of research</td>
<td>Explanation, strong prediction</td>
<td>Understanding, weak prediction</td>
</tr>
<tr>
<td>Focus of interest</td>
<td>What is general, average and representative</td>
<td>What is specific, unique, and deviant</td>
</tr>
<tr>
<td>Knowledge generated</td>
<td>Laws Absolute (time, context, and value free)</td>
<td>Meanings Relative (time, context, culture, value bound)</td>
</tr>
<tr>
<td>Subject/Researcher relationship</td>
<td>Rigid separation</td>
<td>Interactive, cooperative, participative</td>
</tr>
<tr>
<td>Desired information</td>
<td>How many people think and do a specific thing, or have a specific problem</td>
<td>What some people think and do, what kind of problems they are confronted with, and how they deal with them</td>
</tr>
</tbody>
</table>
iii) Critical Realism

The essence of realism is captured in the philosophy of “what the senses show us as reality, is the truth; that objects have an existence independent of the human mind. The theory of realism is that there is a reality quite independent of the mind” (Orlikowski & Baroudi 1991). Critical realists debate that we are all actually experiencing sensations, images of objects in the real world, rather than experiencing the objects directly (Neuman 2011).

It is clear that the pair-programming study is not based on critical realism because, according to Neuman (2011), critical researchers focus on the basis of disagreements, conflicts and paradoxes occurring in society to seek emancipation of the people in society. This research study is aligned with an interpretivist epistemological stance where the researcher acknowledges the different views of interviewees in a social setting.

3.4.2 Research paradigm

A research paradigm is often applied in the social sciences. The term paradigm can cause misunderstandings because it has several meanings. According to Burrell and Morgan (1979), “a paradigm is a way of examining social phenomena from which particular understandings of these phenomena can be gained and explanations attempted”.

![Social theory analysis using four paradigms](Source: Burrell & Morgan 1979)
Burrell and Morgan (1979) divide a research paradigm into four views: *interpretive*, *functionalist*, *radical humanist* and *radical structuralist*, which are illustrated in figure 3.5.

- **Radical Humanism**: This paradigm visualises the current reality as separating people from their truth, and is concerned with emancipating the social reality from social constraints by using radical change. It is aligned with a subjective ontological stance with radical change (Burrell & Morgan 1979).

- **Interpretivism**: This paradigm seeks to explain the nature of behaviour as it occurs in the individual's point of view. It is aligned with a subjective, regulatory ontological stance (Burrell & Morgan 1979).

- **Radical Structuralism**: This paradigm recognises intrinsic structural differences within a society that causes constant change through economic and political crises. It is aligned with an objective ontological stance with radical change (Burrell & Morgan 1979).

- **Functionalism**: This paradigm assumes rational human action on the premise that behaviours can be understood by the use of hypotheses and testing. It is aligned with an objective, regulatory ontological stance (Burrell & Morgan 1979).

The four paradigms serve the following purposes: (i) to assist researchers in explaining their assumptions of their interpretation of the nature of science and society; (ii) to provide an understandable manner in which other researchers consider their work; and to assist researchers in plotting their own path through their research and to comprehend the possibilities of where they are going (Burrell & Morgan 1979).

The primary research paradigm for this study is interpretive as the research is on how knowledge can be obtained based on earlier conceived assumptions such as: (i) There are sufficient workstations to implement pair-programming; (ii) the time slots allocated for pair-programming are the same for all groups taking part in the research; (iii) a controlled environment (computer laboratory) is available for each group; and there is sufficient time for the students to consult with the lecturer.
3.4.3 Research approach

According to Beiske (2007), there are two types of approaches in research which indicate the route to follow when conducting scientific research, namely deductive and inductive. A deductive approach involves developing a hypothesis based on existing theory and validity testing, while an inductive approach is concerned with collecting empirical evidence and building a theory (Beiske 2007).

![Deductive Vs. Inductive](image)

*Figure 3.6: Deductive approach versus Inductive research approach (Source: Beiske 2007)*

According to Burney and Mahmood (2006) “a deductive approach works from the more general to the more specific” while an inductive approach moves from specific observations to broader generalisations and theories.

This study adopts an inductive research approach. The aim of the researcher is to observe patterns derived from empirical evidence and infer the findings to the theory which is called theory building research (Bhattacherjee 2012). In the study, the researcher uses the analysis of the data collected through interviews, observations and assessments to confirm the theory that “pair-programming positively enhances the academic performance of tertiary level IT students”.

3.4.4 Research strategy

The research strategy selected for this pair-programming study is action research.
Action research is an important option for lecturers to consider in a higher education environment as it links both “action” and “research” (Mills, 2011). Action research in higher education is defined as a systematic process of analysing a tertiary situation to understand and improve the quality of teaching and learning processes (Johnson, 2012). Action research provides researchers (lecturers and stakeholders) with new knowledge and understanding on how to improve educational environment (Mills, 2011; Stringer, 2008).

According to Stringer (2008), the action research process is a cycle of five stages: designing the study, collecting data, analysing data, communicating outcomes, and taking actions as shown in figure 3.7.

![Figure 3.7: Action research cycle](Source: Stringer 2008)

During the first stage (designing the study) the researcher carefully points out the problems to be investigated by following the ethics and validity of the study. During the second stage (collecting data), the researcher collects information from a variety of sources related to the study. The information collected in stage two is analysed in stage three (analysing data) to determine key features of the problems under investigation. During the fourth stage (communicating outcomes), the results and/or outcomes of the study are known and communicated to relevant audiences. The
researcher takes action toward finding solutions of the problems investigated during the final and most critical stage (taking action).

In this study, action research—single iteration—assists the researcher to understand the problems faced by students in programming, while pair-programming is the action taken to solve the problems. The research study is based on a pair-programming intervention. Before the intervention, assessment is conducted on both the solo- and pair-programming groups. Next the intervention is actioned. Finally both the solo- and pair-programming groups are assessed again. The data collected during this single iteration of the action research processed is analysed.

3.4.5 Research methodology

Research methodologies are mainly categorised into three types, namely quantitative, qualitative and mixed methods (Saunders et al. 2009).

3.4.5.1 Qualitative research methodology

Different researchers assign different definitions to qualitative research methodology. Definitions range from the simple to the complex and include:

“Qualitative research methodology is the research using methods such as participant observation or case studies which result in a narrative, descriptive account of a setting or practice” (Parkinson & Drislane 2011).

“Qualitative researchers are interested in understanding the meaning people have constructed, that is, how people make sense of their world and the experiences they have in the world” (Merriam 2009).

“Qualitative researchers study things in their natural settings, attempting to make sense of, or to interpret, phenomena in terms of the meanings people bring to them” (Denzin & Lincoln 2005).

Thus, a qualitative research methodology involves a direct experience and feelings to identify the phenomena and proposes possible relationships between causes and effects.
3.4.5.2 Quantitative research methodology

Different researchers assign different definitions to quantitative research methodology. Some of the definitions are:

“Quantitative research is the numerical representation and manipulation of observations for the purpose of describing and explaining the phenomena that those observations reflect” (Creswell 2009).

“Quantitative research is defined as social research that employs empirical methods and empirical statements. An empirical statement is defined as a descriptive statement about what ‘is’ the case in the ‘real world’ rather than what ‘ought’ to be the case” (Cohen & Manion 1980).

Therefore, a quantitative research methodology deals with numbers to measure the reliability and validity of the data. Quantitative research elucidates phenomena through the collection of numerical data which are then analysed by means of mathematically based techniques (Creswell 1994). Table 3.2 illustrates the differences between the qualitative and quantitative research methods.

Table 3.2: Qualitative vs Quantitative research methods
(Source: Genise 2002)

<table>
<thead>
<tr>
<th>Orientation</th>
<th>Quantitative</th>
<th>Qualitative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assumption about the world</td>
<td>A single reality can be measured by an instrument</td>
<td>Multiple realities</td>
</tr>
<tr>
<td>Research Purpose</td>
<td>Establish relationships between measured variables</td>
<td>Understanding a social situation from participants’ perspectives</td>
</tr>
<tr>
<td>Research methods and Processes</td>
<td>• Processes are established before study commences</td>
<td>• Flexible, changing strategies</td>
</tr>
<tr>
<td></td>
<td>• A hypothesis is formulated before study commences</td>
<td>• Design materialised as data are gathered</td>
</tr>
<tr>
<td></td>
<td>• Deductive in nature</td>
<td>• A hypothesis is not needed to commence with study</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Inductive in nature</td>
</tr>
<tr>
<td>Researcher’s role</td>
<td>The researcher is ideally an objective observer who does not take part or effects what is being explored.</td>
<td>The researcher takes part and becomes absorbed in the research/social setting.</td>
</tr>
<tr>
<td>Generalisability</td>
<td>Worldwide context-free generalisations</td>
<td>In-depth context-based generalisations</td>
</tr>
</tbody>
</table>
3.4.5.3 Mixed methods research methodology

A mixed methods research methodology is adopted when either a qualitative or quantitative approach is selected as the original method of research, and where the second approach is adopted during the course of the research due to shortcomings in the initial approach (Creswell 2009). Saunders et al. (2009) argue that a research method can use both a quantitative and qualitative approach for secondary and primary data to source for appropriate information concerning a mutual theme in the same study. Creswell (2009) further affirms that research can be planned in such a way that the results and interpretation of the initial quantitative phase lead to the emergence of the qualitative phase.

This pair-programming study warrants the use of a mixed-methods research methodology with both quantitative and qualitative data collection and analysis on the effect of pair-programming on the academic progress and learning of software development students at HEIs.

3.4.6 Data collection

Multi-methods data collection has been adopted for this study. The three methods used were semi-structured interviews, observations, and assessments.

3.4.6.1 Interviews

Interviews are techniques of data collection through verbal questioning using a set of prepared questions.

“Interviews can be very productive since the interviewer can pursue specific issues of concern that may lead to focused and constructive suggestions” (Shneiderman & Plaisant 2005).

The main advantages of the interview method are: (i) direct contact with the users often leads to specific, constructive suggestions; (ii) interviews are good at obtaining detailed information; and (iii) only a few participants are needed to gather rich and detailed data (Genise 2002; Shneiderman & Plaisant 2005).

The different types of interviews are unstructured, structured and semi-structured.
i) Unstructured interviews

The unstructured interview enables the interviewer to ask open-ended questions and the interviewee to express his or her own opinion freely. According to Preece, Rogers and Sharp (2002), it is not easy to standardise the interview across different interviewees as each interview has a unique structure. It is however possible to generate rich data, information and ideas during interview sessions because the level of questioning can be varied to suit the context, but it is time consuming and difficult to analyse the data.

ii) Structured interviews

With structured interviews, the interviewer uses a set of prepared questions which are short and clearly worded and require precise answers. The structured interview is easy to conduct and it is easily standardised because the same questions are asked to all participants. According to Preece et al. (2002), “structured interviews are most appropriate when the goals of the study are clearly understood and specific questions can be identified”.

iii) Semi-structured interviews

May (2001) is of the view that a semi-structured interview is an ideal technique to collect data because it contributes to ‘easy’ analysis and comparison of data. Burns (2000) describes a semi-structured interview as taking the form of a conversation between the participant and the researcher. Semi-structured interviews combine the advantages of both structured and unstructured interviews (Preece et al. 2002). In order to be consistent, the interviewer has a set of prepared questions to guide him during the interview. As the interview progresses, the interviewee can be given an opportunity to provide more relevant information if he or she wishes to.

For the pair-programming research study, a semi-structured interview approach to collect data from the participants is selected, where data is collected from participants before and after the introduction of pair-programming.
Before commencement of pair-programming, at the beginning of semester 1, the participants are interviewed to determine (in their view):

- Whether they possess any agile programming skills and are knowledgeable in pair-programming
- If they enjoy programming
- If they enjoy traditional programming (solo-programming)
- Their academic performance with solo-programming

Upon completion of the pair-programming intervention, at the end of semester 2, the participants are interviewed to:

- Verify whether they understand agile programming, especially pair-programming
- Determine whether they enjoy agile programming over solo-programming
- Determine their view on whether their academic performance has improved after implementation of pair-programming

Between 10 and 15 minutes per interview provides sufficient time for discussing any issues raised. Tape-recordings (with the approval of the participants) for later use and referral form an integral part of qualitative data collection. It offers the researcher the opportunity time to engage in discussions with the participants without having to take notes.

3.4.6.2 Observations and assessments

The main purpose of selecting assessments for this research is to measure the progress (if any) of the students after implementation of pair-programming.

Students who are registered for the Information Systems module are randomly divided into two groups. The students in group A are introduced to pair-programming, while the students in group B continue with the normal single-student programming approach (solo-programming or traditional programming) over two semesters. For both groups, the same materials are covered and activities conducted. The researcher uses two activities to measure the progress of the students in both groups.
Observations: The researcher assigns different software development projects, all on the same standard, to the students. In group A, the students work in teams (pair-programming) to implement the database while in group B, the students carry out the projects individually. Throughout the sessions, across two semesters, the lecturer (researcher) closely observes the paired and solo students in their respective classes in terms of behaviour, group work (where applicable), timely submission of completed projects, and quality of completed projects.

Assessments: The researcher assesses both groups to measure the academic performance of each student. The assessment consists of three practical tests per semester, thus six practical tests in total. The tests are similar, but not identical, to the projects assigned to the students. During these tests, all the students work individually. Both groups (paired and solo) receive the same tests to complete each time.

3.4.7 Unit of analysis

According to Saunders et al. (2009), a unit of analysis is representative of the targeted population. For this research study, the unit of analysis is the pair-programming technique administered to students during their practical software development sessions.

3.4.8 Unit of observation

The research participants, also known as the units of observation, are the students who were selected through non-random purposive sampling to partake in this study.

3.4.8.1 Population

Population is regarded as any complete group of people and communities where they share mutual characteristics (Zikmund, Babin, Carr & Griffin 2010). In this research, the population consists of all the students registered for the Information Technology qualification, specifically those enrolled for the Information systems module, at the selected HEI in Gauteng. Of the 23 HEIs in South Africa (at the time of this research project) it was convenient to select the specific HEI in Gauteng as the population since the researcher had also been the lecturer there for the units of observation, i.e. convenience sampling was applied.
3.4.8.2 Sample techniques

Sampling is a crucial technique of behavioural research; a research study cannot be undertaken without the use of sampling (Singh 2006). In any research study, it is usually impossible and impractical to collect data from the total population. The collection of data will be made very difficult by factors such as high cost and too much time needed. The main purpose of sampling is to make the research findings cost-effective and precise (Singh 2006).

Cochran (1963) justifies the use of sampling by the following statement:

“In every branch of science we lack the resources to study more than a fragment of the phenomena that might advance our knowledge”.

In his definition, a “fragment” is the sample and “phenomena” is the population. Population means “the entire mass of observations, which is the parent group from which a sample is to be formed. The sample observations provide only an estimate of the population characteristics” (Singh 2006) as shown in figure 3.7.

Figure 3.8 represents the different types of sampling techniques under the probability and non-probability sampling methods.

![Figure 3.8: Example of sample](Source: Singh 2006)
Any sampling method where some elements of the population have no chance of selection, or where the probability of selection cannot be accurately determined, is called non-random (UBOS, 2015).

Purposive sampling is a process of selecting a sample based on the researcher’s knowledge of the population and the nature of the objectives of the study (Babbie 2007).

The sampling technique selected for this study is **non-random purposive sampling**. The participants were divided into two groups (A and B). The students in group A
were introduced to pair-programming while the students in group B continued with the normal solo-programming approach. For both groups, the same materials were covered. Both groups had the same number practical sessions per week, and the lecturer gave the same attention to both. Both groups were observed in the same manner by the lecturer.

3.4.8.3 Sample size

As indicated in section 3.4.8.1, the population in this research are students registered for the Information Systems module, which forms part of the Information Technology qualification at the identified HEI in South Africa. The Information Systems module requires the students to develop software. The number of students selected from the population is 50 (fifty), thus the selected sample size is 50.

3.4.9 Data analysis

Data analysis begins by going back to the aim of study (Greeff 2002) which, in this research, is to investigate the perceptions of students regarding the use of pair-programming in a software development module at HEIs in South Africa. The analysis of raw data is defined as the application of reasoning to understand the data that had been gathered (Zikmund, Babin, Carr & Griffin 2010).

To align the analysis and interpretation with the various types of data, both quantitative and qualitative data analysis techniques are required.

3.4.9.1 Quantitative data analysis

The quantitative data is constituted of the assessment results of the students from both groups, six assessments in total, three per semester. The analysis method selected for this empirical data is twofold:

- A \textit{t-test} to measure the difference, if any, between the semester 1 and semester 2 results of the paired students. The pair-programming intervention is measured on the impact it has on the students’ academic performance over two semesters.
- A second *t-test* to measure the difference, if any, between the semester 1 and semester 2 results of the solo students. This *t-test* measures the change (if any) of the students' academic performance over two semesters.

Graphs and tables are used to draw relevant conclusions from the analysed data.

### 3.4.9.2 Qualitative data analysis

According to Flick (2010), qualitative data analysis searches for the meaningful content of data among the vast amounts of qualitative data collected. To analyse data using a qualitative method, the data first needs to be represented in written format (Saunders *et al.* 2009). Saldana (2009) developed a coding framework to assist researchers in understanding the coding process and concepts, and how they are transformed into themes and categories (figure 3.9). Quinlan (2011) supports the use of a coding framework by stating that a qualitative thematic method will assist any researcher to classify themes into codes. For the pair-programming research study, the qualitative thematic coding framework of Saldana (2009) (figure 3.9) and the principles of hermeneutics (figure 3.10) are combined as interpretive, descriptive tool for analysing data collected from the semi-structured interviews.

![Figure 3.10: Streamlined Codes-to-Theory model for qualitative inquiry](Source: Saldana 2009)
Myers (1997) defines hermeneutics as:

“Interpretation, in the sense relevant to hermeneutics, is an attempt to make clear, to make sense of an object of study. This object must, therefore, be a text, or a text analogue, which in some way is confused, incomplete. The interpretation aims to bring to light an underlying coherence or sense”.

Hermeneutics are concerned with the meaning of text from the interviews which has to be linked to a code through an analysis process (Flick, 2010). Hermeneutic units are utilised in qualitative data analysis software to group sentences of data which have parallel meanings.

The combined thematic coding framework and principles of hermeneutics as method of analysis selected for the pair-programming study (without the use of qualitative data analysis software), is summarised as follows:

- Read through all the transcripts of the recorded interviews
- Summarise the data
- Identify all existing similarities
- Group the data according to a coding structure
- From the meaningful relationships detected, identify patterns and concepts
- Transform these patterns and concepts into a theme (figure 3.10)
3.5 APPOSITE RESEARCH DESIGN SUMMARISED

Table 3.3 contains a summary of an apposite research methodology that can be used to explore the effect of pair-programming on the academic progress of software development students at HEIs in South Africa.

The research design components have been identified as the philosophy, paradigm, approach, strategy, methodology, data collection, unit of analysis, unit of observation, and data analysis.

For each of the components, a proposed action is recommended.
Table 3.3: Apposite research design for pair-programming research at HEIs

<table>
<thead>
<tr>
<th>Research design components</th>
<th>Proposed method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research Philosophy</td>
<td></td>
</tr>
<tr>
<td>Ontology</td>
<td>Subjectivism</td>
</tr>
<tr>
<td>Epistemology</td>
<td>Interpretivism</td>
</tr>
<tr>
<td>Research Paradigm</td>
<td>Interpretive</td>
</tr>
<tr>
<td>Research Approach</td>
<td>Inductive</td>
</tr>
<tr>
<td>Research Strategy</td>
<td>Action Research</td>
</tr>
<tr>
<td>Research Methodology</td>
<td>Mixed Methods (both qualitative and quantitative)</td>
</tr>
<tr>
<td>Data Collection</td>
<td>Multi-Methods</td>
</tr>
<tr>
<td></td>
<td>Semi-structured Interviews</td>
</tr>
<tr>
<td></td>
<td>Observations</td>
</tr>
<tr>
<td></td>
<td>Assessments</td>
</tr>
<tr>
<td>Unit of Analysis</td>
<td>Pair-programming</td>
</tr>
<tr>
<td>Unit of Observation</td>
<td></td>
</tr>
<tr>
<td>Students</td>
<td>Students registered for the Information Systems module at specified HEI in Gauteng</td>
</tr>
<tr>
<td>Sample technique</td>
<td>Purposive (non-random)</td>
</tr>
<tr>
<td>Sample size</td>
<td>50 students</td>
</tr>
<tr>
<td>Data analysis</td>
<td></td>
</tr>
<tr>
<td>Quantitative</td>
<td>T-tests</td>
</tr>
<tr>
<td></td>
<td>Graphs</td>
</tr>
<tr>
<td>Qualitative</td>
<td>Qualitative Thematic Coding</td>
</tr>
<tr>
<td></td>
<td>Hermeneutics</td>
</tr>
</tbody>
</table>

3.8 CONCLUSION

This chapter outlined the research philosophy, paradigm, approach, strategy, methodology, participants, data collection and analysis methods of a study that explores the effect of pair-programming on the academic progress of software development students at HEIs in South Africa.

The ontological stance of the research philosophy is based on subjectivism, and the epistemological stance is aligned with an interpretivist philosophy.

The primary research paradigm for this study is interpretive and the research approach is inductive. This study warrants the use of a mixed methods research methodology and a multi-method data collection strategy. The methods of data collection include a semi-structured interview, observations and assessments.
The unit of observation is pair programming, and the units of analysis are the students registered for the Information Systems module in IT at a specified HEI in South Africa, and the sample size is fifty (50).

Given the types of data collected, both qualitative and quantitative data analysis methods are needed. For the quantitative data, empirical analysis methods in the form of *t*-tests and graphs are selected. The qualitative data are analysed using a thematic coding framework and the hermeneutics principle.

A table which contains a summary of an apposite research design that can be used to explore the effect of pair-programming on the academic performance of software development students at HEIs in South Africa has been proposed.
CHAPTER 4: ANALYSIS AND DISCUSSION

4.1 INTRODUCTION

Williams and Upchurch (2001), Chigona and Pollock (2008) and Williams (1999) are of the opinion that an agile systems approach towards education shows several promising properties for educational purposes and could lead to retention in the number of students enrolled for programming modules at higher educational institutions (HEIs).

For the purpose of this research, pair-programming as agile approach is defined as grouping students in teams whereby collaborative or cooperative learning is implemented. Pair-programming is an approach to group work that maximises learning and satisfaction resulting from working as part of an effective team. The research is guided by determining how pair-programming, which is an agile approach, shapes the experience of tertiary level IT students with regard to their academic performance in developing software. Secondary to academic performance, the research also looks at the attitude and software development enjoyment level of students.
One of the learning outcomes of IT “…focuses on the design and development of appropriate computer-based solutions to specific problems using programming” (Conradie 2013). However, most of the students are not able to develop appropriate computer-based solutions due to a lack of programming skills. Students do not develop strong programming skills because they are taught traditionally with individual programming assignments and competitive grading rather than in-depth learning in teams in order to master programming languages and develop software development skills (Williams & Upchurch 2001). Students face many obstacles when attempting to develop computer-based solutions to specific problems through developing software individually rather than in a team. These obstacles contribute to a low pass rate of students enrolled for computer programming modules at universities in South Africa at first year level. Research indicates that, in general, teams have the capability to make more efficient decisions than individuals because teams can combine knowledge and information, which assists in good decision making (Russo & Schoemaker, 1989; Schmidt, Montoya-Weiss & Masse 2001; Wheeler & Valacich 1996).

The main focus of this chapter is on the data analysis phase of the research study. The research design provides summarised, core information relevant to the research philosophy, paradigm, approach, strategy and data collection methods. The remainder of the chapter concentrates on the data analysis.

4.2 RESEARCH DESIGN

Research design gives direction to the research process, from the underlying philosophical assumptions throughout data collection to the analysis, findings and recommendations. For this study, the ‘Research Onion’ design of Saunders et al. (2009) has been adopted. The ontological stance of the research philosophy is based on subjectivism and the epistemological stance is aligned with an interpretivist philosophy (the quest for subjective knowledge through qualitative data) (Raddon 2015). The research paradigm is interpretive and the research approach inductive in nature. An inductive research approach is concerned with the generation of new theory emerging from the data (Gabriel 2015).
The study warrants the use of a mixed methods research methodology, action research strategy and a multi-methods data collection technique. The methods of data collection include a semi-structured interview, observations and assessments.

A non-random sample size of 50 students was purposively selected from a population of IT students registered for the Information Systems module over two semesters in 2013 at a university in Gauteng. The participants were divided into two groups (A and B). The students in group A were introduced to pair-programming while the students in group B continued with the normal solo-programming approach. For both groups, the same materials were covered.

Data was collected from participants through a semi-structured interview before commencing with pair-programming at the beginning of semester 1, and after completion of the pair-programming intervention at the end of semester 2.

Throughout the sessions, across two semesters, the researcher closely observed the paired and solo students in their respective classes in terms of behaviour, group work (where applicable), timely submission of completed projects, and quality of completed projects.

The researcher also assessed both groups during the intervention to measure the academic performance of each student. The assessment consisted of three tests per semester, thus six tests in total. The tests were similar, but not identical, to the projects assigned to the students. During these assessments, all the students worked individually. Both groups received the same tests to complete each time.

4.3 DATA ANALYSIS

Both qualitative and quantitative data analysis methods were applied to the data collected. Empirical analysis methods in the form of t-tests and graphs were performed on the quantitative data collected. The qualitative data were analysed using a thematic coding framework and hermeneutics principles.
4.3.1 Quantitative data analysis

The quantitative data collected constitutes the six assessment results of the students from both the pair-programming and solo-programming groups. Two $t$-tests were performed.

The first $t$-test measured the difference between the semester 1 and semester 2 results of the paired students to determine the impact of the pair-programming intervention on the students’ academic performance over two semesters.

The second $t$-test measured the difference between the semester 1 and semester 2 results of the solo-programming students to determine the impact of not administering pair-programming on the students’ academic performance over two semesters.

4.3.2 Qualitative data analysis

The qualitative data collected constitutes tape-recordings and the subsequent transcripts of the semi-structured interviews conducted with the 50 participants before commencement and after completion of the pair-programming intervention. As data analysis method, a combined thematic coding framework and the principles of hermeneutics were selected. The analysis steps included reading through all the transcripts of the recorded interviews, summarising the data, identifying all existing similarities, and grouping the data according to a coding structure. From the meaningful relationships detected, patterns and concepts we identified and transform into themes (Strauss & Corbin 1990).

4.4 DISCUSSION OF RESULTS

The secondary research objectives of the study have been stated in section 1.5. The outcomes of secondary research objectives ii), iii) and iv) are discussed below.

4.4.1 Enjoyment level

| Secondary research objective ii): To determine the impact of pair-programming on IT students’ enjoyment level of software development |
a) Interview analysis

Would you enjoy pair-programming more than solo-programming?

This question, derived as a theme (using thematic analysis) from Questions 5, 11, 15 and 16 in the Interview Guide (see Annexure A), assisted the researcher in establishing the enjoyment level of students towards pair-programming. Some of the participants’ answers are indicated below.

Table 4.1 indicates students’ answers before the pair-programming intervention.

<table>
<thead>
<tr>
<th>Questions</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Q5:</strong> Do you enjoy programming? Why?</td>
<td>Student 1: No, I don’t enjoy programming because it is very difficult for me and I don’t understand it.</td>
</tr>
<tr>
<td><strong>Q11:</strong> Are you going to further your career in IT? Why?</td>
<td>Student 1: Yes but in networks.</td>
</tr>
<tr>
<td><strong>Q15:</strong> Do you think pair-programming will help students enjoy programming?</td>
<td>Student 1: No idea.</td>
</tr>
<tr>
<td><strong>Q16:</strong> Do you think you will enjoy a pair-programming experience more than programming alone? Why?</td>
<td>Student 1: I don’t know yet.</td>
</tr>
</tbody>
</table>

- **Question 5: Do you enjoy programming? Why?**

  80% of the students did not enjoy programming because they found it difficult and 20% of the students indicated that they enjoy programming.
• **Question 11: Are you going to further your career in IT? Why?**

84% of the students said no and 16% of the students said yes because they enjoy IT.

• **Question 15: Do you think pair-programming will help students enjoy programming?**

92% of the students said they don’t know and 8% agreed that pair-programming can help students to enjoy programming.

• **Question 16: Do you think you will enjoy pair-programming experience more than programming alone? Why?**

95% of the student said they don’t know because they have never done it before and 5% of students thought that pair-programming could be enjoyable.

Table 4.2 below indicates students’ answers after the pair-programming intervention.

**Table 4.2: Enjoyment level: After pair-programming**

<table>
<thead>
<tr>
<th>Questions</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Q5: Do you enjoy programming? Why?</strong></td>
<td>Yes, I do because programming is challenging and it is funny. Yes, it is a bit challenging. Yes, programming increases my skills of thinking and solving problems. Kind of. Now, I start understanding it.</td>
</tr>
<tr>
<td><strong>Q11: Are you going to further your career in IT? Why?</strong></td>
<td>Yes, I want to go as far as possible in order to learn as much as possible. I will think about it. Yes, I'm enjoying programming. Yes, I love computers and programming. Yes, I enjoy IT.</td>
</tr>
<tr>
<td><strong>Q15: Do you think pair-programming will help students enjoy programming?</strong></td>
<td>Yes Yes definitively. Yes because two heads are better than one head. Yes. Obvious.</td>
</tr>
<tr>
<td><strong>Q16: Do you think you will enjoy a pair-programming experience more than programming alone? Why?</strong></td>
<td>Yes, it helps me to understand programming. Yes, I do enjoy pair-programming experience. No, I want to work alone. Yes, my partner helped me a lot and together programming becomes so easy. Yes, it was a good experience.</td>
</tr>
</tbody>
</table>
- **Question 5: Do you enjoy programming? Why?**
  89% of the students enjoyed programming because they found it becomes easy to find solutions with the help of a programming partner. 11% of the participants indicated that they do not enjoy programming.

- **Question 11: Are you going to further your career in IT? Why?**
  94% of the students indicated yes because they enjoy programming, and 6% of the students indicated no.

- **Question 15: Do you think pair-programming will help students enjoy programming?**
  98% of the students were of the opinion that pair-programming can help students enjoy programming and 2% were not sure.

- **Question 16: Do you think you will enjoy pair-programming experience more than programming alone? Why?**
  89% of students enjoyed pair-programming experience, 7% enjoyed working alone and 4% did not enjoy the experience.

The qualitative thematic analysis and subsequent results signify that an agile approach to programming has a positive impact on the students’ enjoyment level of developing software.

**b) Observation analysis**

The researcher observed the following:

- **Group A:** The paired students enjoyed software development, were confident in programming more than the solo group of students, and seemed less frustrated. In addition, the paired students developed problem solving and communication skills. The majority of paired students were eager and/or willing to pursue a career in programming.

- **Group B:** The majority of solo-programming students were frustrated and found programming to be very difficult. They did not seem to enjoy programming at all.
4.4.2 Academic progress

**Secondary research objective iii):** To determine how pair-programming impacts the academic progress of IT students at HEIs in South Africa with regard to software development

a) Assessment analysis

Figure 4.3 shows the academic performance of the students introduced to pair-programming over two semesters. In the first semester, the students started using pair-programming without any experience of the technique and the overall average of the three assessments for the group was 63.8%.

In the second semester, the same students continued using pair-programming to cultivate their software development skills, which resulted in a 7.8% point increase in the overall average of the group.

![Average Results](image)

**Figure 4.2: Group A: Students using pair-programming**

Figure 4.4 shows the academic performance of the solo-programming students during both semesters of the research study. In the first semester, the academic performance of the students was poor with an average of 46.5% for the three assessments. In the second semester, there was a 5.5% point increase.

The average of the academic performance of the individual (solo) programming students increased to 52%, but it was still less than 71.6% of students using pair-programming. This indicates the need for an intervention to improve the academic performance of software development students.
The overall average of both groups increased with 15 points from semester 1 to semester 2; however, Group A displayed higher average marks than Group B throughout the intervention across both semesters. This indicates that the intervention seems to have a positive impact on the academic performance of IT students.

Analysis methodology followed on the empirical assessment data:

The researcher assessed the students introduced to pair-programming by marking each student's assessments, three per semester, and calculating the average. The total group average was then calculated from the individual average mark of each student in the group. This was done for both groups over two semesters.

Next, the researcher compared the results of the two groups—pair-programming vs. solo-programming. The results obtained from the assessments were used to determine the impact of pair-programming on the academic performance of the students.

To perform a paired t-test in terms of statistical significance, it was decided to set the value at a 95% confidence interval level (p < 0.05). Effect sizes served to decide on the practical significance of the findings. A cut-off point of 0.30 (medium effect) was set for the practical significance of correlation coefficients. A paired t-test was performed to measure whether there was a significant difference between the results of semester 1 and semester 2 for the solo-programming group (Table 4.3).
### Table 4.3: Paired t-test results for solo-programming

<table>
<thead>
<tr>
<th></th>
<th>Variable 1</th>
<th>Variable 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>0.465</td>
<td>0.52</td>
</tr>
<tr>
<td>Variance</td>
<td>0.00239</td>
<td>0.0072</td>
</tr>
<tr>
<td>Observations</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Pearson Correlation</td>
<td>0.824443517</td>
<td></td>
</tr>
<tr>
<td>Hypothesized Mean Diff.</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Df</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>t Stat</td>
<td>2.569046516</td>
<td></td>
</tr>
<tr>
<td>P(T&lt;=t) one-tail</td>
<td>0.025046626</td>
<td></td>
</tr>
<tr>
<td>t Critical one-tail</td>
<td>2.015048373</td>
<td></td>
</tr>
<tr>
<td>P(T&lt;=t) two-tail</td>
<td>0.050093252</td>
<td></td>
</tr>
<tr>
<td>t Critical two-tail</td>
<td>2.570581836</td>
<td></td>
</tr>
</tbody>
</table>

The p-value is 0.05 and $t_s=2.56 < t_C=2.57$, indicating no statistical significant difference between semester 1 and semester 2. Thus, no change occurred with regard to student performance from semester 1 to semester 2 for the solo-programming students.

A paired t-test was also performed to measure whether there was a significant difference between the results of semester 1 and semester 2 for the pair-programming group (Table 4.4).

### Table 4.4: Paired t-test results for agile programming (pair-programming)

<table>
<thead>
<tr>
<th></th>
<th>Variable 1</th>
<th>Variable 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>0.638333333</td>
<td>0.716666667</td>
</tr>
<tr>
<td>Variance</td>
<td>0.010616667</td>
<td>0.003666667</td>
</tr>
<tr>
<td>Observations</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Pearson Correlation</td>
<td>0.93494699</td>
<td></td>
</tr>
<tr>
<td>Hypothesized Mean Diff.</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Df</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>t Stat</td>
<td>3.751008357</td>
<td></td>
</tr>
<tr>
<td>P(T&lt;=t) one-tail</td>
<td>0.006639958</td>
<td></td>
</tr>
<tr>
<td>t Critical one-tail</td>
<td>2.015048373</td>
<td></td>
</tr>
<tr>
<td>P(T&lt;=t) two-tail</td>
<td>0.013279917</td>
<td></td>
</tr>
<tr>
<td>t Critical two-tail</td>
<td>2.570581836</td>
<td></td>
</tr>
</tbody>
</table>
The p-value is 0.01 < 0.05 and \( t_s = 3.75 > t_c = 2.57 \), indicating a significant statistical difference between semester 1 and semester 2. The pair-programming intervention had a positive impact on the academic performance of the students.

Analysis of the results indicates that the students who followed a pair-programming approach to developing software obtained significantly higher results and mastered programming skills on a higher level than those who followed a solo-programming approach.

### 4.4.3 Attitude

**Secondary research objective iv):** To determine the impact of pair-programming on the attitude of IT students towards software development

#### a) Interview analysis

*What can be done to attract more students to programming?*

This question, derived as a **theme** (using thematic analysis) from Question 12 in the Interview Guide (see Annexure A), assisted the researcher in establishing the attitude of students towards programming. Some of the participants’ answers are indicated below.

Table 4.5 below indicates students’ answers before the pair-programming intervention.

<table>
<thead>
<tr>
<th>Question 12</th>
<th>Student 1</th>
<th>Student 2</th>
<th>Student 3</th>
<th>Student 4</th>
<th>Student 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>What can be done to attract more students to programming?</td>
<td>Working in group.</td>
<td>By teaching and making students understood programming.</td>
<td>Grouping students.</td>
<td>Workshop.</td>
<td>Write programs for money.</td>
</tr>
</tbody>
</table>

Table 4.6 below indicates students’ answers after the pair-programming intervention.
Table 4.6: Attitude: After pair-programming

<table>
<thead>
<tr>
<th>Question 12</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Student 1</td>
</tr>
<tr>
<td>What can be done to attract more students to programming?</td>
<td>Introduction of pair-programming.</td>
</tr>
</tbody>
</table>

From the answers of students after the introduction of pair-programming, it is clear that they are in agreement that working in a team will attract more students to programming. Pair-programming made the students more positive, confident and learning was made easier through sharing ideas.

The data analysis indicates that 98% of the students agreed that pair-programming had positively influenced their attitude towards software development in their studies and a future career (figure 4.2).

![Figure 4.4: Students’ attitude towards programming](image)

b) Observation analysis

The researcher observed the following:

- **Group A**: The paired students became increasingly familiar with coding as they shared their thoughts, ideas and source code with one another. All worked efficiently on their assignments. They were preparing well for work in the software industry where employees develop software in teams.
- **Group B**: The majority of the solo-programming students displayed little or no interest in programming. They worked alone on the assignments and seemed to struggle.

### 4.5 CONCLUSION

The primary objective of this study has been formulated as exploring the impact of pair-programming on the academic performance and perception of tertiary level students regarding the use of an innovative agile approach in developing software.

After having analysed the data collected through interviews, observations and assessments, it is clear that an innovative agile programming approach has a positive effect on the academic progress of students, including increased high-level reasoning and critical thinking skills; deeper understanding of learned material; increased quality in time on task; lower levels of anxiety and stress; greater intrinsic motivation to learn and achieve; a greater ability to view situations from others’ perspective; more positive and supportive relationships with peers; more positive attitudes toward programming modules; and a higher self-esteem in terms of creativeness, innovativeness and collaboration; and mastery of learning.

With an agile programming approach, positive interdependence is structured into the team’s tasks and activities, and students are responsible for each other’s success. Communication skills are taught and expected to be used by all team members. The instructor observes and intervenes if necessary to ensure that the process is followed.

From the pair-programming research conducted, results signify that an agile approach will contribute towards assisting students in mastering software development concepts, logic and code, among others, and enhancing students’ programming skills, academic performance, enjoyment level and attitude towards software development.
5.1 INTRODUCTION

This chapter presents the findings derived from the objectives of the study. The viability of using pair-programming in the teaching and training of software development students at HEIs in South Africa is compared to that of using the solo-programming approach. Conclusions are drawn on whether pair-programming has a positive effect on the attitude and enjoyment level of developing software and an improvement in the academic performance of students. A model for applying pair-programming at HEIs in SA is proposed and recommendations for optimal implementation of pair-programming in a controlled setting are made.
5.2 RESEARCH QUESTIONS

This study endeavoured to investigate whether an agile systems approach has a positive impact on IT students registered at HEIs in South Africa by answering the primary research question through six secondary research questions.

The primary research question of this study is:

PRQ: How does pair-programming as agile software development method shape the experience of tertiary level IT students with regard to their academic performance in developing software?

The findings for each of the six secondary research questions are stated and discussed below.

5.2.1 Secondary Research Question 1

SRQ1: How prominent is eXtreme Programming, specifically pair-programming, as educational tool at Higher Education Institutions (HEIs) in general?

Finding 1: Pair-programming is a systems development methodology created by Kent Beck in 1996 (see Section 2.4)

Finding 2: Pair-programming is a technique used in eXtreme Programming where two students or two developers program at one workstation (see Section 2.4)

Finding 3: In a study conducted by Chigona and Pollock (2008), students in pairs submitted higher quality work and received higher marks than solo-programming students (see Section 2.4)

Finding 4: With pair-programming, both programmers share the entire application, both contribute to the success of the application through switching roles periodically, and both should be focused on the task without causing unnecessary difficulties (see Section 2.4.1)
Finding 5: For effective team work in pair-programming, it is important to recognise that there are specific steps to follow, and that the tasks and interpersonal behaviours of the team might change over time (see Section 2.4.2)

5.2.2 Secondary Research Question 2

SRQ2: What is the impact of pair-programming on IT students’ enjoyment level of software development?

Pair-programming positively affects the enjoyment level of programming students. It helps the students to interact, have team discussions and actively participate in team work.

Finding 6: Paired students enjoy software development more than solo students and paired students are more confident and less frustrated (see Section 2.4.3.1 and table 2.3)

Finding 7: Pair-programming improves learning and comprehension of software development because each learning partner has knowledge and skills to offer and both partners then learn from each other (see Section 2.4.1)

Finding 8: Students love pair-programming because it makes software development easier (see Annexure C)

Finding 9: Paired students find that there are different methods to developing a software development project (see Section 2.4.1)

Finding 10: Pair-programming develops communication skills between both learning partners and enables students to socialise (see Sections 2.4.3.6; 2.4.1)

Finding 11: With pair-programming, students communicate and share the entire application (i.e. programming code) (see Annexure C)
5.2.3 Secondary Research Question 3

SRQ3: How does pair-programming impact the academic progress of IT students at HEIs in South Africa with regard to software development?

Pair-programming positively affects the academic performance of IT students by helping them to interact and discuss possible solutions among themselves. With pair-programming, both learning partners share the entire application and both are responsible for the success of the application. The paired students combine their unique skills to have a large subset of knowledge and skills to solve the problem.

Finding 12: On average, the academic performance of students introduced to pair-programming, improved significantly. In semester 2, the same students continued using pair-programming and this resulted in a 15% point increase in terms of academic achievement (see Sections 2.4.3.4; 4.4 and figure 4.3)

Finding 13: Paired students passed the software development module after having failed it twice before (see Annexure C)

Finding 14: Pair-programming assists students in understanding software development, therefore students managed to pass the subject (see Annexure C)

Finding 15: On average, paired students achieve significantly higher results than those working alone (solo-programming) (see Section 2.4.3.4)

Finding 16: Pair-programming is indicated as the reason for a significant increase in the software development marks of students and for passing the subject with high marks (see Annexure C)

Finding 17: Paired students develop time management skills because both learning partners feel responsible for and accountable to each other, therefore they plan ahead to avoid letting each other down (see Section 2.4)

Finding 18: Paired students submitted all their projects on time (see Annexure C)
**Finding 19:** Paired students develop programs with high functionality and more readable programming code (see Sections 2.4; 2.4.1 and figure 2.5)

**Finding 20:** The quality of the applications delivered by paired students is distinctively higher than the applications delivered by solo-programmers (see Section 2.4.3.3)

**Finding 21:** Pair-programming develops trust between learning partners (see Section 2.4.1)

**Finding 22:** Programs or code developed by paired students are significantly more descriptive than programs developed by solo students for the same projects, and paired students received higher marks than solo students for these programs (see Sections 2.4; 2.4.3.3 and figure 2.4)

**Finding 23:** Pair-programming increases the motivation among the paired students because each learning partner wants to contribute to the success of the joint programming project (see Section 4.1)

**5.2.4 Secondary Research Question 4**

**SRQ4:** How does pair-programming impact the attitude of IT students towards software development?

Paired students submitted higher quality work which increased their confidence and positively changed their attitude towards software development and a future career.

**Finding 24:** Paired students are more confident in programming than solo students (see Section 2.4.3.2)

**Finding 25:** Paired students become more accustomed to programming concepts than solo students (see Section 2.4.3.2)

**Finding 26:** Paired students work more efficiently on their projects because pair-programming reduces debugging time (see Sections 2.4.1 and 2.4.3.3)
Finding 27: Pair-programming develops the problem solving skills of students (see Annexure C)

Finding 28: Pair-programming students are more willing to pursue a career in programming than solo-programming students because they enjoy software development in teams more than working on their own (see Annexure C)

Finding 29: Pair-programming prepares students for the world of work where employees often work in teams (see Sections 2.4.2; 2.4.3.5 and 2.4.3.6)

Finding 30: Paired students act positively towards working with their programming partners using a collaborative programming approach (see Section 2.4 and table 2.3)

Finding 31: Programming code developed by paired students is simple, shorter, and easier to understand (see Section 2.4 and figure 2.5)

Finding 32: Paired students develop a positive attitude towards programming and working in a team (see Section 2.4.3.5)

5.2.5 Secondary Research Question 5

SRQ5: What model can be proposed to shape the experience of tertiary level IT students with regard to their academic performance in developing software?

This study has shown that pair-programming contributes significantly towards improving the academic performance of IT students in terms of software development. However, the implementation of pair-programming in the classroom can seem challenging due to the structure of the classroom or the lack of knowledge of the lecturer/instructor on how to apply pair-programming. There are no conventional guidelines to follow to implement pair-programming in the classroom. The researcher proposes the following guidelines which a prospective pair-programming lecturer/instructor can use as framework to implement pair-programming in a practical classroom setting (see figure 5.2 for a graphical representation of the proposed pair-programming framework).
i) Instructor training

Lecturers (also referred to as instructors or facilitators) play a vital role in the successful implementation of pair-programming in the classroom. They need to be...
trained on how to use pair-programming in a controlled environment. It is important to familiarise themselves with pair-programming to be able to accurately explain and apply the principles of this agile programming method to students.

The instructors should frequently explain and reinforce the principles of pair-programming to their students to clarify the roles and keep the students involved during the entire process. Without this necessary reinforcement from the instructor, students would be tempted to revert to solo-programming, a technique they are familiar with (Williams et al. 2002). The instructor therefore has to ensure that the principles of pair-programming are followed; for instance, the pairs must switch roles periodically, and all the learning partners in a group have to be equally involved.

ii) Teaching students the principles of successful pair-programming

Instructors have to teach the students how to use pair-programming. It is a major misconception to assume that students understand pair-programming. The students may erroneously think pair-programming is simply about dividing the work into equal parts and each partner has to concentrate on their section without taking into account the work of the other learning partner(s) in the group. To avoid incorrect implementation of pair-programming, students need to be taught that:

- They have to work together at the same workstation as a (program) coder, controller or advisor
- They need to switch roles periodically
- Each student in the paired group needs to actively participate in the project
- Discussions and collaboration between coder, controller and advisor are encouraged

When a paired group consists of two learning partners (the most preferable group pairing for student training), they switch between coder and advisor.

For three learning partners in a paired group, the roles assigned are coder, advisor, and controller (i.e. project manager).

A discussion on paired groups with four or five learning partners does not fall within the parameters of this study and will therefore not be elaborated on.
iii) Pairing by skills level

The main objective of pair-programming is to ensure that all learning partners in a paired group share the entire application and are equally responsible for the success of the application. It is therefore advised to group the students according to their programming skills level so that they can assist each other in a meaningful way. One option is to pair the highly skilled student to a student with a lower programming skillset. Both learning partners will then have to combine their unique skills to tap into a larger subset of skills and knowledge.

iv) Strict attendance

The instructor has to ensure that the paired students attend class regularly. A student who stays away from class, negatively impacts the quality of the paired group’s project. A student is allowed to change his/her learning partner with the permission of the instructor if the student feels the partner does not contribute positively and equally to the project in terms of regular attendance.

v) Peer evaluation

The Instructor has to provide a mechanism to obtain feedback from the paired students on their learning partners and must act immediately if any problem arises. This will encourage the partners to work hard and contribute to the success of the project. The learning partners are aware that they will evaluate each other and submit the evaluation report to the instructor. Peer evaluation influences the performance of the students and compels them to participate actively and fairly in the project.

By using the following evaluation framework developed by Kaufman, Felder and Fuller (1999), the student can easily rate the contribution of his/her learning partner(s):

- **“Excellent:*** Consistently displayed robust knowledge; tutored the learning partner; well prepared and cooperative.
- **Very good:** Consistently did what s(he) was supposed to do; very well prepared and cooperative.
- **Satisfactory**: Usually did what s(he) was supposed to do; acceptably prepared and cooperative.
- **Ordinary**: Often did what s(he) was supposed to do; minimally prepared and cooperative.
- **Marginal**: Sometimes failed to show up or complete the project; rarely prepared.
- **Deficient**: Often failed to show up or complete the project; rarely prepared.
- **Unsatisfactory**: Consistently failed to show up or complete the project; unprepared.
- **Superficial**: Practically no participation.
- **No show**: The learning partner never participated in the pair-programming sessions.

**vi) Individual evaluation**

The instructor has to ensure that learning takes place and that each student in the group does not rely on his/her learning partner(s) to do all the work. Consequently, the students should be evaluated individually to measure their performance. Individual evaluation assists the instructor in verifying whether the students have learned the course material and assisted each other within their respective groups.

**vii) Composition of the group**

The size of the group depends on the size of the class. Normally with pair-programming, two students are paired into a group; however, if the number of students is high, the instructor can pair up to four, maximum five, students into a group. In cases when there is a student without a group (no one left to pair the student with), the instructor can assign that student to any of the existing groups.

In addition to the framework, the following guidelines are proposed for the paired students to optimise the pair-programming technique (Williams & Kessler 2001; Werner, Denner & Bean 2004):
• The paired students are equal participants and all of them own and share the project.

• The advisor in the paired group must always be actively guiding the coder. It is therefore important that the learning partners switch roles so that each learning partner has the opportunity to code and to advise respectively.

• ‘Ego-less programming’ is essential for establishing effective communication between the learning partners. It is advised to use the words "we" and "us" when groups present their work to the instructor.

• Taking a break periodically is important for maintaining the stamina and refreshing the mind.

• The paired students should view and respect each other as co-learners, colleagues or friends who assist one another to enhance their skills in order to deliver quality work. They should not view one another as someone who forces them into compromise.

5.2.6 Secondary Research Question 6

**SRQ6:** How can pair-programming be implemented optimally in a controlled learning environment?

The researcher observed that a significant number of Information Systems students do not have an adequate understanding of agile systems, especially pair-programming. This is confirmed by Salo and Abrahamsson (2004) who state that a tutorial on pair-programming must be held before the students start using an agile method to familiarise themselves with the principles of pair-programming.

**Recommendation 1:** Higher education institutions should encourage the use of an agile approach to programming, especially in software development where students need to work in teams (collaborative learning) to develop applications.

**Recommendation 2:** Pair-programming should be correctly implemented from the start of a programming module (see Section 5.2.5). Instructors and students will only benefit from pair-programming if it is correctly implemented.
**Recommendation 3**: Instructors need to be trained on how to use pair-programming in order to familiarise themselves with the principles of pair-programming.

**Recommendation 4**: Students must be taught on how to apply the principles of pair-programming to avoid implementing this technique incorrectly, which will result in ineffective learning.

**Recommendation 5**: Pair-programming does not simply mean grouping students in pairs to write programs. However, the instructors need to facilitate and manage the pair-programming experience and continuously apply the guidelines proposed in the framework as well as the principles of collaborative learning.

**Recommendation 6**: Students must switch roles periodically, and discussion and collaboration between the coder and advisor are encouraged.

**Recommendation 7**: The principles of collaborative learning must be followed and respected during the pair-programming sessions.

**Recommendation 8**: The institution should have a controlled environment (computer laboratory) with sufficient workstations available to students in order to implement pair-programming.

**Recommendation 9**: The use of the guidelines proposed in the pair-programming framework (see Section 5.2.5) will assist an instructor to successfully implement pair-programming in a classroom environment.

**Recommendation 10**: A survey on the experiences of instructors concerning the use and implementation of pair-programming in a software development module can be conducted to further optimise the pair-programming model and guidelines.

**Recommendation 11**: Higher education institutions could consider developing applications which provide support by means of a virtual community or e-learning, where students interact and share knowledge with their lecturers and learning partners through ‘virtual communication’ in a setting that simulates a class contact session.
## 5.3 SUMMARY OF FINDINGS

<table>
<thead>
<tr>
<th>Finding No.</th>
<th>Findings</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finding 1</td>
<td>Pair-programming is a systems development methodology created by Kent Beck in 1996.</td>
<td>Chapter 2, Section 2.4</td>
</tr>
<tr>
<td>Finding 2</td>
<td>Pair-programming is a technique used in eXtreme Programming where two students or two developers program at one workstation.</td>
<td>Chapter 2, Section 2.4</td>
</tr>
<tr>
<td>Finding 3</td>
<td>In a study conducted by Chigona and Pollock (2008), students in pairs submitted higher quality work and received higher marks than solo-programming students.</td>
<td>Chapter 2, Section 2.4</td>
</tr>
<tr>
<td>Finding 4</td>
<td>With pair-programming, both programmers share the entire application, both contribute to the success of the application through switching roles periodically, and both should be focused on the task without causing unnecessary difficulties.</td>
<td>Chapter 2, Section 2.4.1</td>
</tr>
<tr>
<td>Finding 5</td>
<td>For effective team work in pair-programming, it is important to recognise that there are specific steps to follow, and that the tasks and interpersonal behaviours of the team might change over time.</td>
<td>Chapter 2, Section 2.4.2</td>
</tr>
<tr>
<td>Finding 6</td>
<td>Paired students enjoy software development more than solo students and paired students are more confident and less frustrated.</td>
<td>Chapter 2, Section 2.4.3.1</td>
</tr>
<tr>
<td>Finding 7</td>
<td>Pair-programming improves learning and comprehension of software development because each learning partner has knowledge and skills to offer and both partners then learn from each other.</td>
<td>Chapter 2, Section 2.4.1</td>
</tr>
<tr>
<td>Finding 8</td>
<td>Students love pair-programming because it makes software development easier.</td>
<td>Annexure C: Feedback from students on enjoyment of pair-programming</td>
</tr>
<tr>
<td>Finding 9</td>
<td>Paired students find that there are different methods to developing a software development project.</td>
<td>Chapter 2, Section 2.4.1</td>
</tr>
<tr>
<td>Finding 10</td>
<td>Pair-programming develops communication skills between both learning partners and enables students to socialise.</td>
<td>Chapter 2, Sections 2.4.3.6 and 2.4.1</td>
</tr>
<tr>
<td>Finding No.</td>
<td>Findings</td>
<td>References</td>
</tr>
<tr>
<td>------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Finding 11</td>
<td>With pair-programming, students communicate and share the entire application (i.e. programming code).</td>
<td>Annexure C: Feedback from students on enjoyment of pair-programming</td>
</tr>
<tr>
<td>Finding 12</td>
<td>On average, the academic performance of students introduced to pair-programming, improved significantly. In semester 2, the same students continued using pair-programming and this resulted in a 15% point increase in terms of academic achievement.</td>
<td>Chapter 4, Section 4.4: Figure 4.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chapter 2, Section 2.4.3.4</td>
</tr>
<tr>
<td>Finding 13</td>
<td>Paired students passed the software development module after having failed it twice before.</td>
<td>Annexure C: Feedback from students on academic performance</td>
</tr>
<tr>
<td>Finding 14</td>
<td>Pair-programming assists students in understanding software development, therefore students managed to pass the subject.</td>
<td>Annexure C: Feedback from students on academic performance</td>
</tr>
<tr>
<td>Finding 15</td>
<td>On average, paired students achieve significantly higher results than those working alone (solo-programming).</td>
<td>Chapter 2, Section 2.4.3.4</td>
</tr>
<tr>
<td>Finding 16</td>
<td>Pair-programming is indicated as the reason for a significant increase in the software development marks of students and for passing the subject with high marks.</td>
<td>Annexure C: Feedback from students on academic performance</td>
</tr>
<tr>
<td>Finding 17</td>
<td>Paired students develop time management skills because both learning partners feel responsible for and accountable to each other, therefore they plan ahead to avoid letting each other down.</td>
<td>Chapter 2, Section 2.4</td>
</tr>
<tr>
<td>Finding 18</td>
<td>Paired students submitted all their projects on time.</td>
<td>Annexure C: Feedback from students on academic performance</td>
</tr>
<tr>
<td>Finding 19</td>
<td>Paired students develop programs with high functionality and more readable programming code.</td>
<td>Chapter 2, Sections 2.4, 2.4.1 and Figure 2.5</td>
</tr>
<tr>
<td>Finding 20</td>
<td>The quality of the applications delivered by paired students is distinctively higher than the applications delivered by solo-programmers.</td>
<td>Chapter 2, Section 2.4.3.3</td>
</tr>
<tr>
<td>Finding No.</td>
<td>Findings</td>
<td>References</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Finding 21</td>
<td>Pair-programming develops trust between learning partners.</td>
<td>Chapter 2, Section 2.4.1</td>
</tr>
<tr>
<td>Finding 22</td>
<td>Programs or code developed by paired students are significantly more descriptive than programs developed by solo students for the same projects, and the paired students received higher marks than solo students for these programs.</td>
<td>Chapter 2, Sections 2.4.3.3, and 2.4: Figure 2.4</td>
</tr>
<tr>
<td>Finding 23</td>
<td>Pair-programming increases the motivation among the paired students because each learning partner wants to contribute to the success of the joint programming project.</td>
<td>Chapter 2, Section 4.1</td>
</tr>
<tr>
<td>Finding 24</td>
<td>Paired students are more confident in programming than solo students.</td>
<td>Chapter 2, Section 2.4.3.2</td>
</tr>
<tr>
<td>Finding 25</td>
<td>Paired students become more accustomed to programming concepts than solo students.</td>
<td>Chapter 2, Section 2.4.3.2</td>
</tr>
<tr>
<td>Finding 26</td>
<td>Paired students work more efficiently on their projects because pair-programming reduces debugging time.</td>
<td>Chapter 2, Section 2.4.3.3, Chapter 2, Section 2.4.1</td>
</tr>
<tr>
<td>Finding 27</td>
<td>Pair-programming develops the problem solving skills of students.</td>
<td>Annexure C: Feedback from students on attitude towards software development</td>
</tr>
<tr>
<td>Finding 28</td>
<td>Pair-programming students are more willing to pursue a career in programming than solo-programming students because they enjoy software development in teams more than working on their own.</td>
<td>Annexure C: Feedback from students on attitude towards software development</td>
</tr>
<tr>
<td>Finding 29</td>
<td>Pair-programming prepares students for the world of work where employees often work in teams.</td>
<td>Chapter 2, Sections 2.4.3.5, 2.4.3.6 and 2.4.2</td>
</tr>
<tr>
<td>Finding 30</td>
<td>Paired students act positively towards working with their programming partners using a collaborative programming approach.</td>
<td>Chapter 2, Sections 2.4.3.5 and 2.4: Table 2.3</td>
</tr>
<tr>
<td>Finding 31</td>
<td>Programming code developed by paired students is simple, shorter, and easier to understand.</td>
<td>Chapter 2, Section 2.4: Figure 2.4</td>
</tr>
<tr>
<td>Finding 32</td>
<td>Paired students develop a positive attitude towards programming and working in a team.</td>
<td>Chapter 2, Section 2.4.3.5</td>
</tr>
</tbody>
</table>
5.4 PRACTICAL CONTRIBUTION

The main contribution of this study is collaborative learning as a pedagogical approach in which cooperation between students who develop software, is encouraged. The performance of students is improved when they work together to obtain module outcomes. In this study, module outcomes relate to specific practical computer programming outcomes students must obtain in order to pass.

The researcher assessed both groups (pair-programming and solo-programming) to measure the academic performance of the students. The assessment consisted of three practical tests. During the assessments, all the students worked individually. The results obtained from the empirical analysis of the test marks indicate that the students introduced to pair-programming obtained higher marks than the solo-programming students.

5.5 LIMITATIONS

The focus of this research was on HEIs in South Africa, more specifically, the main campus of a university in Gauteng, and did not include any satellite campuses.

The researcher acknowledges that the findings may not be generalised as the population is relatively small (students registered for the Information Systems module) and might differ from other higher education institutions in South Africa.

5.6 FUTURE RESEARCH

This research was delimited to HEI students in South Africa registered for the Information Systems module of the Information Technology qualification. Follow-up research on pair-programming could be conducted with programming students across a number of universities and/or a number of programming modules.

A study which focuses on a larger sample size than 50 students could be conducted to confirm the improvement of the academic performance, attitude and enjoyment level of students taking part in pair-programming.
5.7 SUMMARY

The main objective of this study was to establish whether the use of pair-programming would contribute significantly towards improving the academic performance of tertiary level IT students in South Africa. An interpretive paradigm with an inductive approach has been identified as a feasible research design for the study.

Students registered for the Information Systems module were randomly divided into two groups. The students in group A were introduced to an agile programming approach called pair-programming, while the students in group B continued with the traditional solo-programming approach.

The study warranted the use of both qualitative and quantitative research methodologies, with a mixed-methods survey data collection strategy. Data was collected through semi-structured interviews, observations and assessments over a period of two consecutive semesters.

Both qualitative and quantitative data analysis methods were applied to the data collected. Empirical analysis methods in the form of $t$-tests and graphs were performed on the quantitative data collected. The qualitative data were analysed using a thematic coding framework and hermeneutics principles.

The qualitative analysis results indicated that pair-programming has a significant positive impact on the attitude and software development enjoyment level of students. The statistical analysis results confirmed that the academic performance of students introduced to pair-programming was significantly higher than students who continued with a solo-programming approach.

The paired students displayed a higher level of reasoning and analytical skills, better understanding of learned material, and a higher quality of time on task than the solo-programming group. Paired students displayed lower levels of anxiety and stress, higher motivation to learn and achieve, more supportive and positive relationships with classmates, more positive attitudes toward programming, and higher self-esteem expressed through creativeness, innovativeness, collaboration and mastery of learning.
From the findings the researcher developed a Pair-programming Framework to provide guidelines on how to implement pair-programming efficiently in a controlled learning environment.

5.8 Conclusion

From the results of this study, pair-programming as an agile systems development methodology contributes significantly towards enhancing the academic performance and enjoyment level of software development students registered for the Information Technology qualification at Higher Education Institutions in South Africa, and changes their attitudes to be more positive towards software development.
REFERENCES


102


Annexure A: Interview Guide

- The purpose of the interview is to obtain data on pair-programming.
- Interviews were held in a quiet environment at the university.
- Each participant was given 10 min to 15 min.
- All information provided is confidential.

<table>
<thead>
<tr>
<th>Name:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Student Number:</td>
<td></td>
</tr>
<tr>
<td>Date of interview:</td>
<td></td>
</tr>
</tbody>
</table>

- Do you have your own computer?
- How did you become interested in computers?
- Why did you decide to study IT?
- Do you enjoy IT?
- Do you enjoy programming? And Why?
- What don’t you like in programming?
- What programming language do you prefer? And Why?
- What skills do you need to be strong in programming?
- Do you have those skills?
- How many times do you practice programming by yourself per week?
- Are you going to further your career in IT? And Why?
- What can be done to attract more students to programming?
- Do you have any idea about solo-programming? If yes, what is solo-programming?
- Do you have any idea about pair-programming? If yes, what is pair-programming?
- Do you think pair-programming will help students enjoy programming?
- Will you enjoy pair-programming experience more than programming alone? And why?
ANNEXURE B: INTERVIEW RESPONSES FROM PARTICIPANTS (BEFORE AND AFTER)

From the 50 students interviewed, feedback from ten (10) participants who were most representative of the collaborative group opinion, were selected to present in this annexure.

BEFORE INTRODUCING PAIR-PROGRAMMING TO STUDENTS

<table>
<thead>
<tr>
<th>Questions</th>
<th>Answers</th>
<th>Answers</th>
<th>Answers</th>
<th>Answers</th>
<th>Answers</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Do you have your own computer?</td>
<td>Yes, I do.</td>
<td>Yes, I have.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>2. How did you become interested in computers?</td>
<td>I got interested in computers because I like playing games on my computer.</td>
<td>By going to the Internet café.</td>
<td>I was so obsessed with computers.</td>
<td>Mentors informed me about IT.</td>
<td>I wanted to know how to fix computers.</td>
<td></td>
</tr>
<tr>
<td>3. Why did you decide to study IT?</td>
<td>I decided to study programming because I love programming.</td>
<td>I’m so interested in technology.</td>
<td>I love computers.</td>
<td>I saw a lot of people I know succeeding.</td>
<td>I love computers.</td>
<td></td>
</tr>
<tr>
<td>4. Do you enjoy IT?</td>
<td>No, I don’t enjoy at all.</td>
<td>Yes</td>
<td>Sometimes</td>
<td>Yes</td>
<td>Very much.</td>
<td></td>
</tr>
<tr>
<td>5. Do you enjoy programming? Why?</td>
<td>No, I don’t enjoy programming because it is very difficult for me and I don’t understand it.</td>
<td>Yes, it is a bit challenging.</td>
<td>No, it is boring and very confusing.</td>
<td>Yes, It challenges me and I find it very interesting</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>7. What programming language do you prefer? Why?</td>
<td>I don’t prefer any programming language because as I said I don’t understand programming.</td>
<td>Visual Basic because it is easy.</td>
<td>Don’t like programming.</td>
<td>Java</td>
<td>Java, I want to create mobile applications.</td>
<td></td>
</tr>
<tr>
<td>Questions</td>
<td>Participant 1</td>
<td>Participant 2</td>
<td>Participant 3</td>
<td>Participant 4</td>
<td>Participant 5</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------------------------------------</td>
<td>---------------------------------------------------</td>
<td>---------------------------------------------------</td>
<td>---------------------------------------------------</td>
<td>---------------------------------------------------</td>
<td>---------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>8. What skills do you need to be strong in programming?</td>
<td>Problem solving in programming.</td>
<td>Coding skills.</td>
<td>Problem solving.</td>
<td>Creativeness</td>
<td>Faith</td>
<td></td>
</tr>
<tr>
<td>9. Do you have those skills?</td>
<td>No, I don’t have it.</td>
<td>No.</td>
<td>No.</td>
<td>Not yet.</td>
<td>Not yet.</td>
<td></td>
</tr>
<tr>
<td>10. How many times do you practice programming by yourself per week?</td>
<td>Four times.</td>
<td>Two times.</td>
<td>Not sure.</td>
<td>3 to 4 times a week.</td>
<td>2 times a week.</td>
<td></td>
</tr>
<tr>
<td>11. Are you going to further your career in IT? Why?</td>
<td>Yes, I can’t change the course.</td>
<td>Yes, to know more about technology.</td>
<td>No, IT is difficult because of programming.</td>
<td>Yes, it is very broad and one can choose anything within IT.</td>
<td>Yes, I love computers more than anything.</td>
<td></td>
</tr>
<tr>
<td>12. What can be done to attract more students to programming?</td>
<td>Working in group.</td>
<td>By teaching and making students understood programming.</td>
<td>Grouping students.</td>
<td>Workshop</td>
<td>Write programs for money.</td>
<td></td>
</tr>
<tr>
<td>13. Do you have any idea about solo-programming? If yes, what is solo-programming?</td>
<td>Yes, solo-programming means working alone.</td>
<td>No.</td>
<td>Yes, programming alone.</td>
<td>Yes, a sort of programming where one can work individually.</td>
<td>Writing program by yourself without any help from anyone.</td>
<td></td>
</tr>
<tr>
<td>14. Do you have any idea about pair-programming? If yes, what is pair-programming?</td>
<td>No, I don’t have any idea. But I can guess it is working in group.</td>
<td>No.</td>
<td>No idea about pair-programming.</td>
<td>Yes, two or more work together.</td>
<td>Sharing the program with someone.</td>
<td></td>
</tr>
<tr>
<td>15. Do you think pair-programming will help students enjoy programming?</td>
<td>I don’t know yet.</td>
<td>I don’t know anything about pair-programming.</td>
<td>Don’t know.</td>
<td>Yes</td>
<td>Yes, programming works better with two people.</td>
<td></td>
</tr>
<tr>
<td>16. Do you think you will enjoy pair-programming experience more than programming alone? Why?</td>
<td>I don’t know yet.</td>
<td>Don’t know yet.</td>
<td>Don’t know.</td>
<td>I don’t know yet because I usually work alone.</td>
<td>No, I like programming on my own.</td>
<td></td>
</tr>
<tr>
<td>Questions</td>
<td>Answers</td>
<td>Answers</td>
<td>Answers</td>
<td>Answers</td>
<td>Answers</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>1. Do you have your own computer?</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td><strong>2. How did you become interested in computers?</strong></td>
<td>I got interested when my sister had one.</td>
<td>Playing games on computer.</td>
<td>Through playing videos games.</td>
<td>By doing my assignments.</td>
<td>I want to learn more about technology.</td>
<td></td>
</tr>
<tr>
<td><strong>3. Why did you decide to study IT?</strong></td>
<td>I want to be a programmer.</td>
<td>I want to know more in IT field.</td>
<td>To become a qualified software developer.</td>
<td>IT has a wide knowledge.</td>
<td>I want to learn about technology.</td>
<td></td>
</tr>
<tr>
<td><strong>4. Do you enjoy IT?</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, I do enjoy IT but not the programming part.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td><strong>5. Do you enjoy programming? Why?</strong></td>
<td>Yes, because programming is challenging and I like challenges.</td>
<td>No, it is confusing for me.</td>
<td>Yes, programming is what I want to do in the industry.</td>
<td>No, it is very difficult.</td>
<td>Yes, it taught me how to run codes.</td>
<td></td>
</tr>
<tr>
<td><strong>6. What don’t you like in programming?</strong></td>
<td>Complex questions.</td>
<td>Everything, as I said it is confusing.</td>
<td>I like programming.</td>
<td>I don’t like everything which deals with programming.</td>
<td>Nothing</td>
<td></td>
</tr>
<tr>
<td><strong>7. What programming language do you prefer? Why?</strong></td>
<td>VB</td>
<td>I don’t like any programming language.</td>
<td>Java</td>
<td>Maybe VB because is easy.</td>
<td>C# and Java because I find them simple and understandable.</td>
<td></td>
</tr>
<tr>
<td><strong>8. What skills do you need to be strong in programming?</strong></td>
<td>Understanding the programming logic.</td>
<td>I don’t know.</td>
<td>Logic design and analytical skills.</td>
<td>No idea.</td>
<td>Practicing</td>
<td></td>
</tr>
<tr>
<td><strong>9. Do you have those skills?</strong></td>
<td>Not really.</td>
<td>I don’t know.</td>
<td>Yes</td>
<td>Don’t like programming, as I said.</td>
<td>Yes, I practice a lot.</td>
<td></td>
</tr>
<tr>
<td>Questions</td>
<td>Answers</td>
<td>Answers</td>
<td>Answers</td>
<td>Answers</td>
<td>Answers</td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>10. How many times do you practice programming by yourself per week?</td>
<td>Every day (2 hours per day).</td>
<td>I don’t practice at all.</td>
<td>5 times a week.</td>
<td>I practice only before the test or exam.</td>
<td>Twice a week.</td>
<td></td>
</tr>
<tr>
<td>11. Are you going to further your career in IT? Why?</td>
<td>Yes, because I want to be very good in computers.</td>
<td>Yes but in networks.</td>
<td>Yes because I want to become a better software developer with a degree in IT.</td>
<td>Yes because it is too late for me to change the course.</td>
<td>Yes, because I love and enjoy doing it.</td>
<td></td>
</tr>
<tr>
<td>12. What can be done to attract more students to programming?</td>
<td>Student must be taught programming from high school.</td>
<td>Enough time for practicing.</td>
<td>Exposing students to great opportunities that are offered by programming.</td>
<td>Start coding from scratch and the lecturer should share codes where and when to place a specific piece of code, not just opening a folder with the codes already done.</td>
<td>Tell the students more about programming, also how does it work.</td>
<td></td>
</tr>
<tr>
<td>13. Do you have any idea about solo-programming? If yes, what is solo-programming?</td>
<td>No</td>
<td>No</td>
<td>Yes, when one programmer builds an application on his own.</td>
<td>No</td>
<td>Yes, a program that is made by one person.</td>
<td></td>
</tr>
<tr>
<td>14. Do you have any idea about pair-programming? If yes, what is pair-programming?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes, a program that is made by two people.</td>
<td></td>
</tr>
<tr>
<td>15. Do you think pair-programming will help students enjoy programming?</td>
<td>No idea.</td>
<td>I don’t know.</td>
<td>Don’t know yet.</td>
<td>I don’t have any information on pair-programming.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>16. Do you think you will enjoy pair-programming experience more than programming alone? Why?</td>
<td>I don’t know.</td>
<td>I don’t know.</td>
<td>To be honest, I don’t know.</td>
<td>I don’t have any information on pair-programming, as I said previously.</td>
<td>Yes, I will need the help somewhere.</td>
<td></td>
</tr>
</tbody>
</table>
### Questions

<table>
<thead>
<tr>
<th>Questions</th>
<th>Answers</th>
<th>Answers</th>
<th>Answers</th>
<th>Answers</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Do you have your own computer?</td>
<td>Yes, I do.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>2. How did you get interested in computers?</td>
<td>I got interested in computers because I like playing games on my computer.</td>
<td>By going to the Internet café.</td>
<td>I was so obsessed with computers.</td>
<td>Mentors informed me about IT.</td>
<td>I wanted to know how to fix computers.</td>
</tr>
<tr>
<td>3. Why did you decide to study IT?</td>
<td>I decided to study programming because I love programming.</td>
<td>I’m so interested in technology.</td>
<td>I love computers.</td>
<td>Many people I know succeeding in IT.</td>
<td>I love programming.</td>
</tr>
<tr>
<td>4. Do you enjoy IT</td>
<td>Yes, I do.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Very much.</td>
</tr>
<tr>
<td>5. Do you enjoy programming? Why?</td>
<td>Yes, I do because programming is challenging and it is funny.</td>
<td>Yes, it is a bit challenging.</td>
<td>Kind of. Now, I start understanding it.</td>
<td>Programming is very interesting and I like it.</td>
<td>It is challenging and I love challenges.</td>
</tr>
<tr>
<td>6. What don’t you like in programming?</td>
<td>I don’t like when a lecturer spends more time in theory.</td>
<td>I don’t like when we don’t practice.</td>
<td>I don’t like too much theory.</td>
<td>Lot of theory.</td>
<td>Nothing</td>
</tr>
<tr>
<td>8. What skills do you need to be strong in programming?</td>
<td>Problem solving, critical thinking, fast thinking and logical thinking.</td>
<td>To be creative.</td>
<td>Coding.</td>
<td>Creativity</td>
<td>Problem solving skills.</td>
</tr>
<tr>
<td>9. Do you have those skills?</td>
<td>Yes, I do have them after being introduced to pair-programming.</td>
<td>Now I can say I’m getting them.</td>
<td>Not sure but right now I’m able to code in Visual Basic.</td>
<td>Yes, I do.</td>
<td>Yes, I think I do.</td>
</tr>
</tbody>
</table>

---

**AFTER INTRODUCING PAIR-PROGRAMMING TO STUDENTS**
<table>
<thead>
<tr>
<th>Questions</th>
<th>Answers</th>
<th>Answers</th>
<th>Answers</th>
<th>Answers</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>10. How many times do you practice programming by yourself per week?</td>
<td>Many times because now I enjoy programming.</td>
<td>Many times because my partner assists me.</td>
<td>Many times.</td>
<td>Every day, I can’t sleep without doing some exercises.</td>
<td>6 times a week.</td>
</tr>
<tr>
<td>11. Are you going to further your career in IT? Why?</td>
<td>Yes, I want to go as far as possible in order to learn as much as possible.</td>
<td>I will think about it.</td>
<td>Yes, I’m enjoying programming.</td>
<td>Yes, it is very broad and gives many options in the industry.</td>
<td>Yes, I love computers and programming.</td>
</tr>
<tr>
<td>14. Do you have any idea about pair-programming? If yes, what is pair-programming?</td>
<td>Yes, I do. Pair-programming means working in group of two students in which both students share everything and switch roles and both must be active.</td>
<td>Yes, programming in team of two partners.</td>
<td>Yes of course, coding in a group of two partners where you share the entire project.</td>
<td>Yes, sharing codes.</td>
<td>Typing codes collaboratively with the partners.</td>
</tr>
<tr>
<td>15. Do you think pair-programming will help students enjoy programming?</td>
<td>Yes because two heads are better than one head.</td>
<td>Yes, it is the best.</td>
<td>Obvious</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>16. Will you enjoy pair-programming experience more than programming alone? Why?</td>
<td>Yes, I do. Pair-programming helps me to improve my programming skills. In our team we help each other to find solution.</td>
<td>Yes, it helps me to understand programming.</td>
<td>Yes, it helps me to pass Visual Basic.</td>
<td>Yes, I do enjoy pair-programming experience.</td>
<td>Yes, It was amazing.</td>
</tr>
<tr>
<td>Questions</td>
<td>Answers</td>
<td>Answers</td>
<td>Answers</td>
<td>Answers</td>
<td>Answers</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1. Do you have your own computer?</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>2. How did you get interested in computers?</td>
<td>My sister introduced me to computers.</td>
<td>Playing games and doing my assignment on my computer.</td>
<td>Playing video games.</td>
<td>Doing my assignments on the computer.</td>
<td>To know more about IT.</td>
</tr>
<tr>
<td>3. Why did you decide to study IT?</td>
<td>I want to be a good programmer and develop some application</td>
<td>To know more about IT.</td>
<td>I want to be a qualified programmer.</td>
<td>IT Gives me many options to do. For examples: Networks, Programming, Database, etc.</td>
<td>To have more knowledge in IT.</td>
</tr>
<tr>
<td>4. Do you enjoy IT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>5. Do you enjoy programming? Why?</td>
<td>Yes, programming increases my skills of thinking and solving problems.</td>
<td>Yes, I do. Programming helps me to solve problems.</td>
<td>Yes, I want to be a programmer in my life.</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>8. What skills do you need to be strong in programming?</td>
<td>Programming logic skills.</td>
<td>Problem solving.</td>
<td>Logical and analytical skills.</td>
<td>Problem solving.</td>
<td>Logical thinking.</td>
</tr>
<tr>
<td>9. Do you have those skills?</td>
<td>Yea, I’m going there.</td>
<td>Yes I do have it.</td>
<td>I’m getting there with the help of my partner.</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>10. How many times do you practice programming by yourself per week?</td>
<td>Everyday</td>
<td>5 times a week with or without my partner.</td>
<td>5 times.</td>
<td>Almost every day.</td>
<td>6 times a week.</td>
</tr>
<tr>
<td>Questions</td>
<td>Answers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. Are you going to further your career in IT? Why?</td>
<td>Participant 6: Yes, I want to be a good programmer.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 7: Yes, programming is so interesting.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 8: Yes, I want to be a better software developer.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 9: Yes, now I love programming.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 10: Yes, I enjoy IT.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. What can be done to attract more students to programming?</td>
<td>Participant 6: Applying pair-programming.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 7: Enough time for practicing (practical classes) and let the</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>students to do the assignments in team (Pair-programming).</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 8: Exposing students to great opportunities that are offered</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>by programming.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 9: Teaching theory to students and give them enough time to</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>practice in team.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 10: Grouping the students in teams and give them many</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>exercises to do.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13. Do you have any idea about solo-programming? If yes, what is solo-</td>
<td>Participant 6: Working alone.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>programming?</td>
<td>Participant 7: Yes, coding alone.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 8: Yes, coding individually.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 9: Yes, coding alone.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 10: Yes, doing the whole coding alone.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14. Do you have any idea about pair-programming? If yes, what is pair-</td>
<td>Participant 6: Coding the entire project in team by sharing ideas.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>programming?</td>
<td>Participant 7: Yes, coding with your partner.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 8: Yes, coding with your partner.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 9: Yes, now I know what is pair-programming. Sharing code</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>with your partner and both of you must be active throughout coding.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 10: Yes, coding with your partner.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15. Do you think pair-programming will help students enjoy programming?</td>
<td>Participant 6: Yes, it is a strong tool.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 7: Yes, because students are able to get help from their</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>partners but not for me because I want to get the solution by my own.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 8: Yes, based on my experience pair-programming helps me to</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>understand programming. So yes it helps students to enjoy programming.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 9: Yes definitively.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16. Will you enjoy pair-programming experience more than programming</td>
<td>Participant 6: Yes, it was a wonderful experience.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>alone? Why?</td>
<td>Participant 7: Yes, it was a good experience.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 8: No, I want to work alone.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 9: Yes, I enjoyed pair-programming. It helps me to</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>understand some coding by sharing ideas with my partner.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Participant 10: Yes, my partner helped me a lot and together</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>programming becomes so easy.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ANNEXURE C: FEEDBACK FROM STUDENTS - OBJECTIVES

From the 50 students interviewed, feedback from ten (10) participants who were most representative of the collaborative group opinion, were selected to present in this annexure.

ENJOYMENT OF PAIR-PROGRAMMING

Student 1: I enjoy pair-programming.
Student 2: I love pair-programming because it makes programming easy.
Student 3: Everyone enjoys pair-programming.
Student 4: With pair-programming, I communicate and share ideas with my partner.
Student 5: Pair-programming is much better than working on your own.
Student 6: I enjoy Pair-programming because it enhances my ability to think.
Student 7: Pair-programming helps me to be able to run some applications alone what I couldn’t do before.
Student 8: Pair-programming assists us to increase our logical thinking to solve programming problems which is wonderful.
Student 9: I enjoy pair-programming because I can discuss with my partner. But sometimes I feel like pair-programming is time consuming and I like to work alone.
Student 10: Pair-programming is awesome. My partner assisted me to develop a positive attitude towards programming and stop thinking that programming is difficult.

ACADEMIC PERFORMANCE

Student 1: I have managed to increase my marks and passed the subject with good marks.
Student 2: My partner and I passed the subject after failing it twice before.
Student 3: It helps to understand the subject therefore I managed to pass the subject.

Student 4: Pair-programming is useful. I submitted all the projects on time because we were assisting each other.

Student 5: My partner and I are able to develop good projects.

Student 6: I am happy. Now I managed to pass programming and understand programming.

Student 7: I passed programming with good results which was impossible before being introduced to pair-programming. My partner really helped me a lot to understand programming.

Student 8: I passed all my programming subjects and now I am able to develop good applications alone.

Student 9: I and my teammate passed programming with distinction. So I am very happy.

Student 10: I passed some programming subjects and I am still struggling with other. But I am sure at the end of the day I will pass.

ATTITUDE TOWARDS SOFTWARE DEVELOPMENT AND A CAREER IN PROGRAMMING

Student 1: Now I understand programming and I am confident. So I think I will continue with programming to develop more software.

Student 2: I have developed problem solving skills. So I will be a good programmer to contribute to the success of the community.

Student 3: Yes of course, I will continue with programming.

Student 4: So far pair-programming helps me to understand programming. But I am still thinking of carrying on with programming.

Student 5: Pair-programming makes me enjoy programming, so I will try my luck in programming.

Student 6: pair-programming makes programming easy but still I will go to networks because I love networks.
Student 7: I will towards programming because it brightens my future and I hope one day I will design my own program.

Student 8: Yes of course, I will continue with programming in the industry.

Student 9: I enjoy programming so I will continue with it.

Student 10: Pair-programming is awesome. My partner assisted me to develop a positive attitude towards programming and stop thinking that programming is difficult.
ANNEXURE D: PUBLISHED JOURNAL ARTICLE

SA Innovation Summit Journal

ON

RESEARCH, SCIENCE, TECHNOLOGY AND INNOVATION
2014

LOCAL INNOVATION
GLOBAL CONVERSATION

VOLUME 2, ISSUE 1

Selected papers presented at Cape Town Stadium, Western Cape on 16–17 September 2014
# TABLE OF CONTENTS

1. PREFACE ........................................................................................................ iii
2. FOREWORD ..................................................................................................... v
3. COMMITTEES ................................................................................................. vii
4. INTERNATIONAL PANEL OF REVIEWERS .................................................. viii
5. KEYNOTE SPEAKER: PATRICIA GOUWS ................................................... 1
   5.1 Biography ................................................................................................... 1
   5.2 Keynote Address: Innovation through Community Engagement — Inspired towards Science, Engineering and Technology (I-SET) – Gouws, PM ................................................... 2
6. FULL PAPERS ................................................................................................. 15
   6.1 Schooling Teachers and Learners by Design – Enslin, C (Dr).......................... 15
   6.3 Agile Programming: An Innovative Approach to Software Development In Higher Education – Kasilongo, KWM ......................................................................................... 49
   6.4 Degradation of Pre-treated Abattoir Wastewater for Optimisation of OLR and HRT in a UASB Reactor – Ondari, JM ................................................................................................................ 60
7. DISCUSSION PAPERS .................................................................................... 72
   7.1 The Rationale and Trajectory of ‘Re-Blocking’: A Case of Formalising the Informal and Informalising the Formal – Bolnick, A ........................................................................ 72
   7.2 Innovation and Enterprise Development: The Case of eThekwini Municipality – Chetty, A (Dr) ... 83
   7.3 Design Thinking as a Catalyst for a Culture of Innovation in Organisations: A Case of Project Management – Makhoaibe, P .................................................................................. 95
   7.4 The Benefits of Structured Innovation – Markoulides, D .................................. 111
   7.5 A Novel Engineering Design for early diagnosis of Breast Cancer – Vaughan, K (Dr) ............ 118
8. SHORT BIOGRAPHIES ................................................................................... 127
   8.1 Authors ....................................................................................................... 127
   8.2 Journal editors and Academic Conference Stream Organisers ............................ 129
# 4. INTERNATIONAL PANEL OF REVIEWERS

<table>
<thead>
<tr>
<th>Name</th>
<th>Affiliation and Email</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Beyers, R (Dr)</strong></td>
<td>Young Engineers and Scientists of Africa, South Africa, Email: <a href="mailto:ron@yesa.org.za">ron@yesa.org.za</a></td>
</tr>
<tr>
<td><strong>Du Pré, RH (Prof)</strong></td>
<td>Chairperson: South African Technology and Training Platform (SATTP), South Africa, Email: <a href="mailto:rhdupre@sattp.net">rhdupre@sattp.net</a></td>
</tr>
<tr>
<td><strong>De la Harpe, AC (Dr)</strong></td>
<td>Cape Peninsula University of Technology, South Africa, Email: <a href="mailto:andre@cencra.com">andre@cencra.com</a></td>
</tr>
<tr>
<td><strong>Farquharson, F</strong></td>
<td>National Skills research Agency (NASRA), South Africa, Email: <a href="mailto:fionaf@vitalitybroadband.co.za">fionaf@vitalitybroadband.co.za</a></td>
</tr>
<tr>
<td><strong>Grové, W</strong></td>
<td>University of the Western Cape, South Africa, Email: <a href="mailto:wgrove@uwc.ac.za">wgrove@uwc.ac.za</a></td>
</tr>
<tr>
<td><strong>Lombard, A</strong></td>
<td>Vaal University of Technology, South Africa, Email: <a href="mailto:alombard@vut.ac.za">alombard@vut.ac.za</a></td>
</tr>
<tr>
<td><strong>Meyer, ME</strong></td>
<td>South African Technology and Training Platform (SATTP), Australia, Email: <a href="mailto:memeyer2265@gmail.com">memeyer2265@gmail.com</a></td>
</tr>
<tr>
<td><strong>Simpson, A (Dr)</strong></td>
<td>Medico-legal Consultant, South Africa, Email: <a href="mailto:andresimson10@gmail.com">andresimson10@gmail.com</a></td>
</tr>
<tr>
<td><strong>Vlok, A</strong></td>
<td>University of Stellenbosch, Business School, South Africa, Email: <a href="mailto:awie.vlok@telkomsa.net">awie.vlok@telkomsa.net</a></td>
</tr>
<tr>
<td><strong>Warden, SC (Dr)</strong></td>
<td>Cape Peninsula University of Technology, South Africa, Email: <a href="mailto:wardens@cput.ac.za">wardens@cput.ac.za</a></td>
</tr>
</tbody>
</table>
6.3 Agile Programming: An Innovative Approach to Software Development in Higher Education

KWM Kafilongo  
Vaal University of Technology  
Email: kindukaf@gmail.com

A Jordaan  
Vaal University of Technology  
Email: annjor@yebo.co.za

P Conradie  
Vaal University of Technology  
Email: piertc@vut.ac.za

ABSTRACT

The number of students passing computer programming modules at universities in South Africa at first year level is low. Only with the second attempt do most students pass. This delay results in most students completing their National Diplomas in four or even five years. One possible contribution towards a solution for this problem is the introduction of an innovative agile approach to Information and Communications Technology (ICT) education. The agile approach, based on the principles of agile systems development methodologies, is especially relevant to ICT education. This study endeavoured to investigate the perceptions of fifty students regarding the use of an innovative agile approach in a computer programming module. Interviews were used as a data collection method. It was established that students have a positive opinion regarding the use of an agile approach in innovative computer programming, with students reporting that they found the class more engaging, that cooperation assisted in developing a shared understanding, and that they were more motivated. This improved motivation links to enhanced self-efficacy, identified as a separate theme during content analysis. The use of an agile approach in the classroom can thus support praxis, introducing a teaching strategy with positive results, not only regarding the final marks, but also the perceptions of students.

Keywords: Agile Approach, Collaborative Learning, Engagement, Innovation, Motivation.

1. INTRODUCTION

An agile systems approach seems advantageous to education in general and could lead to retention in the number of students enrolled in Information Technology (IT) courses at Higher Education Institutions (HEIs) in South Africa. With an agile approach, the students use collaborative learning (also known as cooperative learning) which maximises the learning and satisfaction that result from working as part of a high-performance team. This research was guided by the question whether an agile approach could have a positive effect on the perception of students regarding the use of such an approach in an innovative computer programming module, and whether it could result in an increase in the student retention rate in Information and Communications Technology (ICT) at higher education (HE) level.

One of the learning outcomes of IT focuses on the design and development of appropriate computer-based solutions to specific problems using programming (i.e. software development). However, most of the students are not able to develop an appropriate computer-based solution using programming due to a lack of innovative programming skills. They do not develop strong programming skills because they are taught traditionally with
individual programming assignments and competitive grading rather than deeper learning in
teams in order to master programming languages and develop innovative programming skills.
Students face many obstacles when attempting to develop computer-based solutions to
specific problems using programming or developing software individually rather than in a
team. This contributes to the low pass rate of students enrolled for computer programming
modules at universities in South Africa at first year level. Research concluded that teams in
general have the potential to make more effective decisions than individuals as teams can
**pool** knowledge and information, which assists in good decision making (Russo &

2. LITERATURE REVIEW

According to Beck (2000), the term Agile Software Development Methodology (ASDM)
refers to specific methodologies that share the principles and values as stated in the *Agile
Manifesto*, highlighting twelve principles through which a methodology can be identified to
be agile. In order for a methodology to be deemed agile, the most important characteristic is
the ability to adapt quickly to change. This adaptability is achieved through the techniques
and tools of the particular methodology (Beck, 2000). One innovative tool, referred to as agile
programming, is based on the premise that better software can be developed by iterative and
incremental software development methodologies which include Extreme Programming (XP),
Dynamic Systems Development Method (DSDM) and Feature-Driven Development (FDD).
According to Beck (2000), the use of XP in Industry has been claimed to provide significant
benefits and there seems to be potential in the use of this methodology for student projects. In
addition, Adams *et al*. (2003) states that the use of XP is common in most fields of innovative
software development. In a study done by Zhang (2010), it was found for example that agility
is widely accepted in the manufacturing industry as a new competitive concept.

Beck (2000) identified twelve principles through which the agile methodology can be
recognised:

a) The highest priority is to satisfy the customer through early and continuous delivery of
valuable software.

b) Welcome changing requirements, even late in development. Agile processes harness
change for the competitive advantage of the customer.

c) Deliver working software frequently, from a couple of weeks to a couple of months,
with a preference to a shorter timescale.

d) Business people and developers must work together daily throughout the project.

e) Build projects around motivated individuals. Provide them with the environment and
support they need, and trust them to get the job done.

f) The most efficient and effective methodology of conveying information to and within
a development team is face-to-face conversation.

g) Working software is the primary measure of progress.

h) Agile processes promote sustainable innovative development. The sponsors,
developers and users should be able to maintain a constant pace indefinitely.
i) Continuous attention to technical excellence and good design enhances agility.

j) Simplicity, which is the art of maximising the amount of work not done, is essential.

k) The best architectures, requirements and designs emerge from self-organisation.

l) At regular intervals, the team reflects on how to become more effective and then adjusts its behaviour accordingly.

One of the most important reasons for using an agile approach in innovative programming is the use of grouping students in teams whereby collaborative or cooperative learning is implemented. Collaborative learning used in agile programming is an approach to group work that maximises the learning and satisfaction which results from working as part of a high-performance innovative team. Relative to students being taught traditionally with instructor-centred lectures, individual assignments focus on concepts which have limited opportunities for students to practice programming skills.

Moreover, for many students, especially novices and those without the relevant background, it is not easy to learn programming concepts and languages whereas students who are taught to use agile programming tend to learn collaboratively because they are working in a collaborative group. Thus, collaboratively taught students tend to exhibit higher academic achievement; advanced high-level reasoning and critical thinking skills; a deeper understanding of learned material; greater time on task; lower levels of anxiety and stress; greater intrinsic motivation to learn and achieve; greater ability to view situations from others’ perspectives; more positive and supportive relationships with peers; more positive attitudes toward programming subjects; and a higher self-esteem (Johnson et al., 1998).

Johnson et al. (1998) further state that cooperative learning is instruction that involves students working in teams to accomplish a common goal, under conditions that include the following elements:

Positive interdependence—team members are obliged to rely on one another to achieve the goal. If any team members fail to do their part, everyone suffers consequences.

Individual accountability—all students in a group are held accountable for doing their share of the work and for mastery of all of the material to be learned.

Face-to-face promotive interaction—which enables some of the group work to be parcelled out and done individually and some to be done interactively, with group members providing one another with feedback, challenging reasoning and conclusions, and perhaps most importantly, teaching and encouraging one another.

Appropriate use of collaborative skills—thereby encouraging and helping students to develop and practice trust-building, leadership, decision-making, communication and conflict management skills.

Group processing—team members set group goals, periodically assess what they are doing well as a team, and identify changes they need to make in order to function more effectively in the future.
3. RESEARCH PROBLEM

ICT students in general display a lack of knowledge in solving problems that involve different programming languages and technologies. Most of the time students do not have experience in innovative software development. This results in students becoming discouraged and feeling that they are struggling alone. One possible solution for this problem is the introduction of an agile approach (i.e. a collaborative pedagogical approach), whereby the students develop software in collaborative teams. The problems are encapsulated in the following research questions.

4. RESEARCH QUESTIONS

The research questions for this study are:

- What is the current status of the academic performance of ICT students in Higher Education?
- How does agile programming affect the academic performance of ICT students?
- How does pair-programming change the attitudes of students to be more positive towards innovative software development and a career in ICT?
- What recommendations can be made to advance agile programming?

5. AIMS OF THE STUDY

The aims of this study are to:

- To determine the current status of the academic performance of ICT students in Higher Education.
- To determine how agile programming affects the academic performance of ICT students.
- To examine how pair-programming changes the attitudes of students to be more positive towards innovative software development and a career in ICT.
- To recommend strategies that can be used to advance agile programming.

6. RESEARCH METHODOLOGY

This study is aligned with the positivist and post-positivist research paradigm. Therefore, the nature of the study warranted the use of a quantitative statistical analysis on the effect of agile programming in Higher Education (HE). A quasi-experimental study and survey (i.e. two research methods) have been conducted over a period of one year with students who registered for the module Information systems. Firstly a quasi-experiment was performed and secondly a survey was conducted before and after the implementation of the selected agile approach. The survey was conducted by interviewing each participant individually (i.e. students in the Information Systems class). The results obtained, in combination with the
results of the experimental study which related to assessment marks, were used to draw a relevant conclusion.

6.1 Population and sampling method

Population is regarded as any complete group of people and communities where they share mutual characteristics (Zikmund, 2010). The participants in this research study were students registered for the module *Information Systems* which is a module in Information Technology (IT) where the students are introduced to the concepts of databases and required to develop a working database by using a programming language. From this population a random sample of fifty students were selected.

6.2 Ethical considerations

For both methods, the purpose of the study was explained to the participants and anonymity guaranteed. In addition, the participants were informed that the information gathered is for research purposes only and cannot be used against them. The researcher ensured that no names were mentioned and recorded during the semi-structured interviews.

6.3 Data Collection

The data collection for this study was done by means of both a quasi-experimental study and a semi-structured interview. There were no challenges from participants to collect data for this study as these students attended the module offered by the researcher at a University of Technology in South Africa.

6.3.1 Quasi-experimental study

A quasi-experiment was conducted with a group of students being introduced to an agile approach, while another group continued with the normal single-student programming approach (solo-programming or traditional programming). A survey was conducted before and after the implementation of the selected agile approach to investigate the perceptions of students regarding the use of an agile approach in a computer programming module. The researchers were also interested in measuring the improvement (if any) on the academic performance of participants (students) after implementing innovative agile programming.

For the purpose of this study an agile approach is defined as grouping students in teams whereby collaborative or cooperative learning is implemented. Collaborative learning as used in agile programming is an approach to group work that maximises the learning and satisfaction which results from working as part of a high-performance team.

The researcher divided students registered for the *Information Systems* module into two groups. The students in Group A were introduced to an agile approach while the students in group B continued with the normal single-student programming approach (solo-programming or traditional programming). The researcher lectured both groups by covering the same materials and using the same activities. The researcher used two activities to measure the
academic performance of the students in both groups.

Activity 1: Project

The researcher assigned different projects to the students. The projects were to develop a database. In group A, the students worked in teams (agile programming) to implement the database while in group B the students conducted the projects individually. The researcher observed that the projects done in teams were of high quality and submitted in time while the projects done individually were not of the same high standard and most were submitted late.

The students presented their projects to the researcher for marking. During the presentation of the projects, the students who used the agile programming showed the enjoyment and mastering of programming and delivered high quality databases while the students who worked individually displayed a lack of programming skills and most of the databases were of a poor quality.

Activity 2: Assessment

The researcher assessed both groups to measure the academic performance of each student. The assessment consisted of three tests. During the assessments, all the students worked individually. After marking all three assessments, the researcher observed that the students who were introduced to innovative agile programming performed better that the students who were not introduced to agile programming.

6.3.2 Semi-structured interview

According to May (2001), the semi-structured interview is an ideal technique to collect data because it assists to easily analyse and compare data. According to Burns (2000), a semi-structured interview takes the form of a conversation between the participant and the researcher. The researcher conducted semi-structured interviews to obtain information from participants before and after the quasi-experimental study.

The objectives of conducting the interview twice were:

i) Before quasi-experimental study

To determine whether the participants have any agile programming skills and are knowledgeable on pair-programming.

To determine if participants enjoy programming.

To determine if participants enjoy traditional programming (solo-programming).

To measure the academic performance by using solo-programming.
ii) After quasi-experimental study

To verify whether the participants understand agile programming, especially pair-programming.

To determine whether the students enjoyed agile programming or solo-programming.

To measure the improvement (if any) on the academic performance of participants (students) after implementing agile programming.

The participants were granted between 10 and 20 minutes for the interview to provide sufficient time for discussions about issues raised. The researcher used tape-recordings (with the approval of the participants) for later use and referral. Tape-recordings offered the researcher sufficient time to engage in discussions with the participants without having to take notes.

6.4 Data Analysis

According to Strydom and Delport (2002), data analysis in a qualitative inquiry involves a dual approach. The analysis begins by referring back to the purpose of study (Greeff, 2002) which, in this research, was to investigate the perceptions of students regarding the use of an agile approach in a computer programming module at a HEI in South Africa. The analysis of raw data is defined as the application of reasoning to understand the data that has been gathered (Zikmund et al., 2010). The researcher used graphs and tables in an effort to draw relevant conclusions from the recorded data.

The statistical analysis was carried out with the aid of the SPSS version 17 application. Descriptive statistics (e.g. means, standard deviation, skewness and kurtosis) have been used to analyse the data (Steyn, 2000). Cronbach alpha coefficients were employed to determine the internal consistency, homogeneity and unidimensionality of the measuring instruments. Coefficient alphas contain important information regarding the proportion of variance of the items of a scale in terms of the total variance explained by that particular scale. Pearson product-moment correlation coefficients were used to specify the relationship between the variables. In terms of statistical significance, it was decided to set the value at a 95% confidence interval level (p < 0,05). Effect sizes served to decide on the practical significance of the findings. A cut-off point of 0,30 (medium effect) was set for the practical significance of correlation coefficients.

7. DISCUSSION OF RESULTS

The researcher focused on the students’ perceptions of an innovative agile approach in programming. The researcher posed the following questions to participants:

Would you enjoy an agile approach experience more than individual programming outside a team environment? Why?
This was to assist the researcher in establishing the impact of an agile approach. The results pointed to a positive impact on the students’ perception of an agile approach in programming. The data analysis and graph below indicate that 98% of the students interviewed were of the opinion that their programming skills improved by working in a team, and that their level of enjoyment towards programming also increased.

The researcher focused on the average results of both group A (students using an agile approach) and group B (individual programming) over two semesters. In our study, the assessment consisted of developing a database in the practical classes.

Figure 1 below indicates the students’ perception of an innovative agile approach towards programming.

![Figure 1: Students’ perception of innovative agile approach](image)

As the results indicate in Figure 2 and Figure 3 below, group A obtained an average of 60% in semester 1 and it increased by 15% in semester 2 while group B obtained less than 50% in both semesters. This means group A was performing very well.

Figure 2 shows the academic performance of the students introduced to agile programming in both semester 1 and semester 2. In semester 1, the students started using agile programming without any experience and the average was 60%. In semester 2, the same students continued using agile programming and gained experience. This resulted in an increase of 15% of the academic performance of the students. It can thus be concluded that agile programming has a positive impact on the academic performance of ICT students.
Figure 2: Group A: Students using an agile approach

Figure 3 below shows the academic performance of the students conducting individual programming in both semester 1 and semester 2. In semester 1, the academic performance of the students was poor with an average of 30%. In semester 2, there was a little improvement of the same the students using individual programming. The average of the academic performance went to 45% but still it was under 50%. This shows that there is a need for an intervention to improve the academic performance of the students.

Figure 3: Group B: Individual programming

The researcher assessed the students who were introduced to an agile approach individually to determine their performance. Next, the researcher compared the results with those students who were not introduced to agile programming. The results obtained from the assessments were used to determine the impact of an agile approach on student academic performance.
Analysis of the results indicates that the students who followed an agile approach obtained significantly higher results and mastered programming skills better than those who were not introduced to the agile approach (individual programming).

A paired t-test was performed to measure if there was a significant difference between the results of semester 1 and semester 2 for agile programming. The p-value was 0.01 (p<0.05) and t=3.75, indicating a statistical significant difference between semester 1 and semester 2. The intervention, i.e. agile programming, had a positive impact on student academic performance. Again, a paired t-test was performed to measure if there was a significant difference between the results of semester 1 and semester 2 for individual programming. The p-value was 0.05 and t=2.56, indicating no statistical significant difference between semester 1 and semester 2. Thus, no change occurred with regard to student performance between semester 1 and semester 2.

8. CONCLUSION

The aim of this study was to explore the impact of an agile programming approach on the perception of students regarding the use of an innovative agile approach in a computer programming module at HEIs in South Africa. After analysing the data gathered through surveys, it is clear that an innovative agile programming approach had a positive effect on the academic achievement of students, i.e. an increased high-level reasoning and critical thinking skills; deeper understanding of learned material; greater time on task; lower levels of anxiety and stress; greater intrinsic motivation to learn and achieve; a greater ability to view situations from others’ perspectives; more positive and supportive relationships with peers; more positive attitudes toward programming subjects; and a higher self-esteem by providing creativeness, innovativeness, collaboration and mastery of learning.

With an innovative agile programming approach, positive interdependence is structured into the team task activities and students are responsible for each other’s success. Communication skills are taught and expected to be used by all team members. The instructor observes and intervenes if necessary to ensure that the process is followed. From our research conducted, the results proved to be positive that an agile approach will assist the students in mastering IT subjects such as programming concepts, programming logics and programming code among others, and enhancing their skills and academic performance.

REFERENCES


ANNEXURE E: LANGUAGE EDITING CERTIFICATE
TO WHOM IT MAY CONCERN

FROM: CS van Wyk

SUBJECT: Language Editing

DATE: 15 June 2015

The following dissertation was received for final language editing on the above date:

Candidate : KWM Kafilongo
Student number : 210122587
Qualification : MTech: ICT
Dissertation title : The use of pair-programming to enhance the academic performance of tertiary level software development students

The dissertation is ready for submission.

CS van Wyk
016 950 6644
083 264 3185
SATI 1002439