Role of ADRI Model in Teaching and Assessing Novice Programmers

by

Sohail Iqbal Malik
B.Eng, MSc

Submitted in fulfillment of the requirements for the degree of
Doctor of Philosophy

Deakin University

January, 2016
I am the author of the thesis entitled

Role of ADRI model in teaching and assessing novice programmers

submitted for the degree of Doctor of Philosophy (Information Technology)

This thesis may be made available for consultation, loan and limited copying in accordance with the Copyright Act 1968.

'I certify that I am the student named below and that the information provided in the form is correct'

Full Name: .........Sohail Iqbal Malik.................................................................
(Please Print)

Signed: .................................................................

Date: .........................June 01, 2016.................................................................
DEAKIN UNIVERSITY

CANDIDATE DECLARATION

I certify the following about the thesis entitled (10 word maximum)

Role of ADRI Model in Teaching and Assessing Novice Programmers

Submitted for the degree of Doctor of Philosophy (Information Technology)

a) I am the creator of all or part of the whole work(s) (including content and layout) and that where reference is made to the work of others, due acknowledgment is given.

b) The work(s) are not in any way a violation or infringement of any copyright, trademark, patent, or other rights whatsoever of any person.

c) That if the work(s) have been commissioned, sponsored or supported by any organization, I have fulfilled all of the obligations required by such contract or agreement.

d) That any material in the thesis which has been accepted for a degree or diploma by any university or institution is identified in the text.

e) All research integrity requirements have been complied with.

'I certify that I am the student named below and that the information provided in the form is correct'

Full Name: ..............Mr Sohail Iqbal Malik..................................................

(Please Print)

Signed: .......................................................... Signature Redacted by Library

Date: ............................December 28, 2015..................................................
Acknowledgements

I thank to my Lord (Allah), who gives me the strength to complete this long journey.

I would like to give my sincere and special thanks to my Principal Supervisor Dr. Jo Coldwell-Neilson. Her support, genuine care, patience and concern enabled me to reach this stage of my PhD. Her brilliant ideas and informative suggestions played a very significant role in sharpening my research work.

I would like to thank my employer Buraimi University College, Oman for supporting me in executing my research. Special thanks to H.E Sheikh Ahmad bin Nasser Al-Nuaimi, Dr. Mohammed Farhan Al-Majali, Dr Mohammed Jahangir, Mr. Osama Al-Khasawneh, Mr. Ijaz Khan, Mr. Naushad, and Ms. Raghad. Thanks also to Dr. Anwer, Dr. Guru, Mr. Zaheer Khan and Ms. Elena Gorbacheva for their opinions regarding statistical analysis.

I would like to thank my wife, Saadia Saleem, my twins, Aleena and Abdullah, in supporting me and showing their patience in completing my long journey. A special thanks to my parents, my wonderful siblings, my nephews and nieces and all my friends and colleagues for their support. I thank particularly my father, who always encourages me to finish this task.

This thesis is dedicated to my late mother, who was a great role model of persistence, character and strength.
Relevant Publications


## Contents

Abstract ................................................................................................................................ 1

1  Introduction .................................................................................................................. 4

1.1  Background ........................................................................................................... 4

1.2  Research problem ................................................................................................. 5

1.3  Purpose of the research ......................................................................................... 7

1.4  Significance of the research ................................................................................... 9

1.5  Research approach ............................................................................................... 10

1.6  Organization of the thesis ..................................................................................... 12

2  Literature Review ....................................................................................................... 15

2.1  Introduction ......................................................................................................... 15

2.2  Learning to Program ............................................................................................. 17

2.2.1  Novice Programmers ..................................................................................... 17

2.2.2  Programming knowledge and strategies ....................................................... 18

2.3  Introductory Programming Course design & Teaching Model ......................... 19

2.3.1  Characteristics of novice programmers ........................................................ 25

2.3.2  Teaching Methodologies .............................................................................. 26

2.3.3  Curriculum and content delivery ................................................................. 32

2.4  Mobile device supported learning ........................................................................ 35

2.5  Introductory Programming Supporting Software Tools .................................... 36

2.6  Problem Solving Techniques .............................................................................. 37

2.6.1  Program development by stepwise refinement ............................................. 41

2.7  Educational Taxonomies .................................................................................... 42

2.7.1  Bloom’s Taxonomy ....................................................................................... 42

2.7.2  Revised Bloom’s Taxonomy ......................................................................... 45

2.7.3  Comparison between original and revised Bloom’s taxonomies ................. 46

2.7.4  SOLO Taxonomy ............................................................................................ 47

2.7.5  Two Dimensional Adaptation of Bloom’s Taxonomy – The Matrix Taxonomy ................................................................. 49

2.8  Summary of factors influencing introductory programming learning ............ 51

2.9  Conclusion .......................................................................................................... 52
# CONTENTS

3 Research Methodology and Design .................................................................54
   3.1 Introduction ...............................................................................................54
   3.2 Learning theories ......................................................................................57
   3.3 Defining and Designing Action Research ..................................................59
   3.4 Critiques of Action Research .......................................................................63
   3.5 Case Study ................................................................................................64
   3.6 Research Design .........................................................................................64
      3.6.1 Novices ..............................................................................................65
      3.6.2 Instructor ............................................................................................65
      3.6.3 Details of introductory programming course ........................................66
      3.6.4 Context of learning .............................................................................67
   3.7 Action Research themes and goals ..............................................................67
      3.7.1 Action Research: Cycle 1 ....................................................................70
      3.7.2 Action Research: Cycle 2 ....................................................................73
      3.7.3 Action Research: Cycle 3 ....................................................................75
   3.8 Paper-based Survey ....................................................................................78
      3.8.1 Addressing critiques of surveys ..........................................................79
   3.9 Focus Group ................................................................................................80
      3.9.1 Addressing critiques of focus groups ..................................................81
   3.10 Methods of Data Analysis .........................................................................81
   3.11 Ethical Considerations .............................................................................83
   3.12 Delimitations ............................................................................................84
   3.13 Limitations ...............................................................................................85
   3.14 Conclusion ................................................................................................86
4 Analysis of Current Teaching and Assessment materials of Introductory
   Programming Course .......................................................................................88
   4.1 Introduction ...............................................................................................88
   4.2 The ADRI Model .......................................................................................90
   4.3 Methodology for Cycle 1 ..........................................................................93
      4.3.1 Population and Ethical Consideration ................................................94
      4.3.2 Preparation of the First Survey ...........................................................95
CONTENTS

4.4 Results .....................................................................................................................96
  4.4.1 Analysis of student first survey ......................................................................97
  4.4.2 Analysis of introductory programming course parameters .........................102
  4.4.3 Analysis of Assessment tools .........................................................................111
4.5 Discussion .............................................................................................................113
  4.5.1 ADRI model to handle programming knowledge and problem solving strategies .................................................117
  4.5.2 Analysis of current teaching materials and assessment tools of introductory programming .......................................119
4.6 Conclusion ..........................................................................................................121

5 ADRI based Introductory Programming Course .....................................................123
  5.1 Introduction ......................................................................................................123
  5.2 ADRI model four stages for novices ..................................................................124
    5.2.1 Approach Stage .......................................................................................126
    5.2.2 Deployment Stage ...................................................................................128
    5.2.3 Result Stage ............................................................................................128
    5.2.4 Improvement Stage ..................................................................................129
  5.3 ADRI based Introductory Programming Course Materials ...............................129
    5.3.1 ADRI based Programming Examples or Problems ..................................130
    5.3.2 ADRI based Editor ..................................................................................133
  5.4 Implementation of ADRI based approach ..........................................................138
  5.5 Evaluation of ADRI based approach .................................................................138
    5.5.1 Second Survey with treatment group (cycle 2 students) .............................139
    5.5.2 Focus Group with Introductory Programming Instructors ......................153
  5.6 Discussion .......................................................................................................163
    5.6.1 Survey outcomes ....................................................................................164
    5.6.2 Focus group ............................................................................................169
  5.7 Conclusion .......................................................................................................172

6 Improved ADRI based Introductory Programming Course ..................................174
  6.1 Introduction ......................................................................................................174
  6.2 Suggestions for the ADRI based Introductory Programming Course ...............177
CONTENTS

6.2.1 Common Syntax and Semantic Errors Section in the Result Stage ..............177
6.2.2 A glossary of general terms in ADRI based teaching materials ................183
6.2.3 Supporting features in the ADRI editor ..................................................184

6.3 Impact of the ADRI approach on the teaching materials .............................187
6.4 Impact of the ADRI approach on the students’ performance .......................190
   6.4.1 Comparison of the students’ final grades ..............................................191
   6.4.2 Impact of the ADRI approach on the students who passed the course .....192

6.5 Comparison of the traditional and ADRI based introductory programming  
teaching materials ..............................................................................................194

6.6 Discussion .....................................................................................................197
   6.6.1 Extra section in the Result stage ..........................................................198
   6.6.2 Glossary document for general terms ..................................................199
   6.6.3 New features in the ADRI editor ..........................................................199
   6.6.4 Analysis of final grades of last four semesters ......................................200
   6.6.5 Comparison of the traditional and ADRI approaches in the teaching  
        materials ....................................................................................................201

6.7 Conclusion ....................................................................................................202

7 Conclusions .......................................................................................................204
   7.1 Summary of the research and its contribution ..........................................204
   7.2 Contribution to practice ............................................................................220
   7.3 Implications for future research .................................................................222
   7.4 Concluding Remarks ...................................................................................224

Reference List ......................................................................................................226
Appendix A Ethics Approval .............................................................................244
Appendix B - First Survey .................................................................................245
Appendix C – Second Survey .............................................................................250
Appendix D – Focus Group ...............................................................................255
Appendix E - ADRI based Examples, Exercises and Syntax Errors questions ....256
Lists of Figures

Figure 1.1: Programming shortcut? Adapted from Webster (1994) – Page 7 ............... 6
Figure 2.1: The didactic triangle (adapted from Kansanen, 1999; Kinnunen, 2009) .......... 24
Figure 2.2: The didactic triangle (in the context of this study) .................................... 24
Figure 2.3: Problem Solving Process (PSP) Hyde et al. (1979) .................................... 40
Figure 2.4: Bloom’s Taxonomy: Cognitive Domain (Bloom and Krathwohl, 1956) ...... 42
Figure 2.5: Revised Bloom’s Taxonomy (Anderson et al., 2001) ............................... 45
Figure 2.6: Matrix Taxonomy (Fuller et al., 2007) ....................................................... 50
Figure 3.1: Five steps of Action Research Cycle (Adapted from Shuhidan, 2012; p. 49) . 62
Figure 3.2: Three cycles of action research ................................................................. 68
Figure 3.3: Participants groups of students ................................................................. 85
Figure 4.1: Three cycles of the action research with emphasis on the cycle 1 .......... 89
Figure 4.2: Four Stages of the ADRI Model (Iqbal and Harsh, 2013) .......................... 91
Figure 4.3: First survey ................................................................................................. 96
Figure 4.4: Analysis of introductory programming materials ..................................... 102
Figure 5.1: Three cycles of the action research with emphasis on the cycle 2 .......... 123
Figure 5.2: Interface of ADRI based Editor ............................................................... 134
Figure 5.3: Edraw interface for drawing flowcharts ...................................................... 136
Figure 5.4: Question menu with Next and Previous sub-menus ................................. 136
Figure 5.5: Topic menu with introductory programming topics as sub-menu .......... 137
Figure 5.6: Mean comparison of programming concepts questions ......................... 142
Figure 5.7: Mean comparison of teaching topics ........................................................ 146
Figure 5.8: Mean comparison of learning situations ................................................... 149
LIST OF FIGURES

Figure 5.9: Mean comparison of resources ................................................................. 152
Figure 6.1: Three cycles of the action research with emphasis on the cycle 3 .......... 175
Figure 6.2: Help menu in the ADRI editor ............................................................... 185
Figure 6.3: Tool tip feature for the ADRI editor ...................................................... 185
Figure 6.4: Mnemonic and accelerator features for the ADRI editor .................... 186
Figure 6.5: Impact of the ADRI approach on the students who passed the course .... 193
Figure 7.1: Four stages of the ADRI model ............................................................. 205
Figure 7.2: Failure rates in the IP course ................................................................. 216
Figure 7.3: Dropout rates in the introductory programming course ....................... 217
Figure 7.4: Attrition rate in the introductory programming course ......................... 218
List of Tables

Table 2.1: Polya four principles of problem solving..........................................................38
Table 2.2: SOLO Taxonomy (Shuhidan, 2012).................................................................47
Table 2.3: Summary of research results on factors influencing introductory programming learning outcomes ..............................................................................................................51
Table 3.1: Themes and goals of three cycles of action research ........................................69
Table 3.2: first survey with novices ...................................................................................71
Table 3.3: analysis of introductory programming course materials ..................................72
Table 3.4: preparation of ADRI based teaching materials .................................................73
Table 3.5: second survey with novices ..............................................................................74
Table 3.6: focus group with introductory programming instructors ..................................75
Table 3.7: improved ADRI based teaching materials .......................................................76
Table 3.8: analysis of the ADRI based teaching materials .................................................76
Table 3.9: analysis of introductory programming final grades ........................................77
Table 3.10: comparison of the traditional and ADRI based teaching materials ...............78
Table 4.1: Programming Concepts .....................................................................................98
Table 4.2: Teaching Topics ...............................................................................................99
Table 4.3: Learning Situations .........................................................................................100
Table 4.4: Teaching and Learning Resources ..................................................................101
Table 4.5: Analysis of Course Parameters .......................................................................103
Table 4.6: Types of Problems and Examples ....................................................................106
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.7</td>
<td>Presentation Styles</td>
<td>108</td>
</tr>
<tr>
<td>4.8</td>
<td>Four Problem Solving Steps</td>
<td>110</td>
</tr>
<tr>
<td>4.9</td>
<td>Analysis of Assessment Tools</td>
<td>112</td>
</tr>
<tr>
<td>4.10</td>
<td>Proposed ADRI model four approaches for novice programmers</td>
<td>115</td>
</tr>
<tr>
<td>5.1</td>
<td>ADRI model four approaches for novice programmers</td>
<td>125</td>
</tr>
<tr>
<td>5.2</td>
<td>Programming example based on four stages of ADRI approach</td>
<td>131</td>
</tr>
<tr>
<td>5.3</td>
<td>Programming Concepts (treatment group)</td>
<td>141</td>
</tr>
<tr>
<td>5.4</td>
<td>Mann-Whitney U test for programming concepts</td>
<td>143</td>
</tr>
<tr>
<td>5.5</td>
<td>Teaching Topics (treatment group)</td>
<td>145</td>
</tr>
<tr>
<td>5.6</td>
<td>Mann-Whitney U test for teaching topics</td>
<td>147</td>
</tr>
<tr>
<td>5.7</td>
<td>Learning Situations responses for treatment group</td>
<td>148</td>
</tr>
<tr>
<td>5.8</td>
<td>Mann-Whitney U test for learning situations</td>
<td>150</td>
</tr>
<tr>
<td>5.9</td>
<td>Resources responses for treatment group</td>
<td>151</td>
</tr>
<tr>
<td>5.10</td>
<td>Mann-Whitney U test for resources</td>
<td>153</td>
</tr>
<tr>
<td>6.1</td>
<td>Programming example incorporated with syntax error example</td>
<td>180</td>
</tr>
<tr>
<td>6.2</td>
<td>Example of Semantic error</td>
<td>181</td>
</tr>
<tr>
<td>6.3</td>
<td>Example of syntax warning</td>
<td>182</td>
</tr>
<tr>
<td>6.4</td>
<td>Example of glossary term</td>
<td>184</td>
</tr>
<tr>
<td>6.5</td>
<td>analysis of the ADRI based teaching materials against five categories</td>
<td>188</td>
</tr>
<tr>
<td>6.6</td>
<td>Analysis of the students’ grades</td>
<td>190</td>
</tr>
<tr>
<td>6.7</td>
<td>Grading scheme for the Introductory programming course</td>
<td>192</td>
</tr>
<tr>
<td>6.8</td>
<td>Three categories of students</td>
<td>193</td>
</tr>
<tr>
<td>6.9</td>
<td>Comparison of the traditional and ADRI based teaching materials</td>
<td>196</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 7.1: ADRI approach addresses factors influencing an introductory programming course ...........................................................................................................................................219
Abstract

Programming is a rewarding and yet demanding field in the ICT labour market. It is considered a challenging and difficult area of learning for significant numbers of novice programmers. As a result, high failure and drop-out rates from introductory programming (IP) courses are reported despite extensive research which attempts to address the issue. The key challenge in learning programming is that a number of different sets of skills have to be acquired at the same time. Novices have to not only learn the syntax and semantics of a programming language but also have to develop appropriate problem solving skills. Traditional approaches to teaching programming place more emphasis on the syntax and semantics of the language rather than problem solving strategies to address programming problems. A new approach is required in the learning process of introductory programming which emphasizes all the required skills that novices need to develop. In this research the principles embodied in the ADRI technique (Approach, Deployment, Result, Improvement) were incorporated to redesign the teaching approach and learning materials in an introductory programming course.

An action research methodology was used with three cycles to investigate whether this new approach is effective in improving student programming skills during the course. There were nine activities performed during the span of these three cycles. All three entities of the didactic triangle (student, instructor and content) together with learning context were incorporated in the research design to better understand the problem and execute the proposed solution. The first cycle focused on analysing the teaching materials based on the traditional approach. The second cycle described the process of preparing and then evaluating the teaching materials based on the ADRI approach. The third cycle
introduced improvements in the ADRI approach (as an outcome of the second cycle) and the impact of the ADRI approach was determined compared to the traditional approach to learning introductory programming. The learning context of this research was an introductory programming course offered at Buraimi University College, Oman.

The outcomes of cycle one showed that the current teaching materials used in the introductory programming course were designed using a traditional approach with the primary focus on the syntax and semantics of a programming language. The ADRI approach was incorporated into the teaching materials with a focus on paying equal attention to programming skills and problem solving strategies. The four stages of the ADRI technique were incorporated into the examples and exercises presented in the lectures and laboratory sessions. An ADRI editor was developed to facilitate the new approach and students’ learning process. A glossary of general terms used in programming was prepared to support students.

The four stages of the ADRI approach focused on different sets of skills in the learning process. The first stage (Approach) emphasised problem solving strategies such as pseudo-code and flowchart; the second stage (Deployment) focused on syntax and semantics; the third stage (Result) addressed inputs, outputs and processes used in solving a given problem statement; and the fourth stage (Improvement) emphasised different programming constructs to solve the same given problem statement.

This research gathered data from students enrolled in an introductory programming course in four consecutive semesters. The students who enrolled in the first two semesters where taught using the traditional approach (the control group) and those enrolled in the
final two semesters used the ADRI approach (the treatment group). Surveys comparing the control and treatment groups showed that the ADRI approach was perceived by students as impacting positively on their learning process. The students’ perceptions were supported by their final results which showed not only improved outcomes but also a positive improvement in students’ retention after introducing the ADRI approach.

This study delivers a new teaching approach which promotes deep learning promoting a programming paradigm that starts with the *Problem Statement*, then moves to *Solution Planning* before generating *Code*, rather than the traditional style of starting with the *Problem Statement* and moving straight to *Coding*, which promotes a programming shortcut. The ADRI approach also encourages practice through the requirement of having to solve the same problem using different techniques.
1 Introduction

1.1 Background

Programming is an important and fundamental skill that needs to be developed to meet the requirements of an information society. Programmers are considered to be builders of the information society. They develop software applications consumers’ use every day, multimedia applications which provide constant streaming and entertainment information, and complex systems which perform essential tasks.

The Bureau of Labor Statistics (2015) predicts high demand for programmers in the employment market in coming years. To meet the requirements of the market, computer science (and related disciplines) degree programs emphasis the programming field by including programming studies in the first semester of the degree programs. To become a programmer requires the development of multiple skills including critical thinking, problem solving, being detailed-oriented, as well as learning the syntax and semantics of the programming language. For novices, to acquire all these skills is considered a challenging and difficult task. They have to focus on learning both problem solving strategies and the syntax and semantics of the programming language to acquire these skills.

An approach to teaching programming is required which emphasises all the required skills for novices to become programmers and strengthens their capabilities to support our information society. Moreover, it also helps computer science and related disciplines in fulfilling the growing demands of the labour market for programmers with these multi-skill attributes.
1 INTRODUCTION

The following section introduces the research problem, focusing on the traditional approach to teaching programming. The purpose of the research and the research questions are discussed briefly in section 1.3. The significance of the research is discussed in section 1.4. The research approach and how the cycles of research address the research questions are introduced in section 1.5 and finally the organization of the thesis presented in section 1.6.

1.2 Research problem

High failure and drop-out rates from introductory programming (IP) courses continue to be of significant concern to computer science disciplines despite extensive research attempting to address the issue (Guzdial & Soloway, 2002; Lahtinen et al., 2005; Sykes, 2007; Yadin, A. 2011; Watson & Li, 2014; Zingaro, 2015). Researchers have identified that novice programmers spend more time trying to understand programming language syntax and semantics (programming knowledge) (De Raadt, 2008) rather than paying attention to planning the problem (problem solving strategies) (Webster, 1994; Kölling & Rosenberg, 1996; Ala-Mutka, 2004; De Raadt, 2008). Koulouriet al. (2015) argued that ‘teaching problem solving before programming yielded significant improvements in student performance’ (p. 1). Therefore, a new approach is required which emphasises both programming knowledge and problem solving strategies equally.

The traditional approaches to teaching introductory programming promotes programming shortcuts as students try to develop code directly from the problem statement (Webster, 1994) as depicted in figure 1.1. Students attempt to convert the given problem statement directly into a computer program by using a programming language without going
through the problem solving phase. The more correct way, as suggested in figure 1.1, is to convert the problem statement into an algorithm by using problem solving strategies such as pseudo-code or flowchart and then write a code by using the developed algorithm.

De Raadt (2008) pointed out that most of the text books used for teaching programming languages emphasise the syntax of programming languages rather than seeking to develop problem solving skills. In some cases, problem solving topics are discussed only in the first chapter. Normally, only a small number of text books discuss problem solving topics and they are integrated throughout the book in different topics. A better approach, as suggested by the three step (3-step) approach in figure 1.1 (Problem statement $\rightarrow$ Problem solving phase $\rightarrow$ Implementation phase), should be used to develop course materials which pay equal attention to problem solving strategies and syntax and semantics.

Winslow (1996) argued that practice is an important aspect of learning to program and if novices want to become expert programmers than they should practice, practice and practice. The traditional approach in many introductory programming courses gives more attention to practicing syntax and semantics of programming languages compared to
problem solving strategies (Ala-Mutka, 2004; De Raadt, 2008). As a result, novices only achieve surface learning of the programming domain. Novices should follow the 3-step pathway suggested in figure 1.1, which promotes deep learning and demands more practice.

However, despite the abundance of research investigating different factors and conditions that may influence the achievements of students’ learning outcomes in IP courses, no definitive solution have been identified. Research is still needed to explore the learning difficulties faced by novices and solutions developed to address these problems.

1.3 Purpose of the research

It is evident from the discussion in the previous section that IP has been a focus of research for a long time and many interventions have been developed to improve student outcomes but no definitive solution has been identified yet. Furthermore it is important to address the problems because it also effects the recruitment and retention of students in computer science and related fields, a discipline area that is struggling to meet the recruitment demands of industry (Microsoft, 2012).

The purpose of the study is to develop an approach which supports the teaching and learning process for novice programmers’ that encourages students to focus on all aspects of learning to program adequately. In other words, the approach needs to focus on problem solving strategies as well as on the syntax and semantics of the language. A novel approach is proposed to address difficulties encountered by students in IP by incorporating the ADRI model (Approach, Deployment, Result, Improvement) into the pedagogy, which is well known within the quality assurance and enhancement discipline
1 INTRODUCTION

for self, as well as for external, review (Carroll and Razvi, 2006) and can be considered in
the context of reviewing and assessing both teaching and learning (Carroll and Palermo,
2006; Abuid, 2010; Oliver, 2010). The ADRI model is discussed in detail in chapter 4.

To achieve the aims of current research, the study will deliver an approach and
curriculum that ensures novice programmers are able to practice programming knowledge
and problem solving strategies concurrently thus improving their programming skills.
This will ultimately help computer science departments and educators to address the key
issue of coping with high failure and dropout rates from IP courses. The ADRI model
will be employed as a theoretical framework for this investigation.

To achieve the purpose of the study, the following overarching research question is
described for this thesis:

*Does the implementation of the ADRI model in an introductory programming
teaching context improve student outcomes?*

The overarching question is broken down into the following four sub-questions:

*RQ1: What are the perceptions of students of the barriers and affordances to
learning programming?*

*RQ2: What are the perceptions of instructors of the barriers and affordances to
teaching programming?*

*RQ3: What is the impact of applying the ADRI approach on introductory
programming course materials?*

*RQ4: What is the impact of applying the ADRI approach on student learning?*
1.4 Significance of the research

Programming is one of the demanding fields in the computer science discipline and ‘employment of computer programmers is projected to grow 8 percent from 2012 to 2022’ (Bureau of Labor Statistics, 2015). In 2012, Microsoft published a report showing that American colleges are not producing enough graduates in computer science to meet expected demand up to 2020 (Microsoft, 2012). On the other hand, coping with high failure and dropout rates in the IP course is another issue for computer science disciplines (Guzdial & Soloway, 2002; Lahtinen et al., 2005; Sykes, 2007; Yadin, A. 2011; Watson & Li, 2014; Zingaro, 2015). Most of the computer science and related disciplines offer IP course at their first level and if students fail or drop out of this course, they are unlikely to enrol in a follow-on course (Wiedenbeck et al., 2004). One of the possible reasons is because the traditional approach used in teaching the IP course does not provide students all the required skills. The study described in this thesis analyses the current issues in the teaching and learning process of the introductory programming and introduces a new approach which addresses these problems. The context of learning for current study was Buraimi University College, Oman and the IP course was offered with the traditional approach (see section 3.6.3). As discussed above, the traditional approach gives more attention to practicing syntax and semantics of programming languages compared to problem solving strategies (Iqbal & Harsh, 2013). The current research aims to help the computer science discipline by introducing the new approach in the teaching and learning process of the IP course.

Learning to program is considered to be a difficult task for significant numbers of novices (Shuhidan, 2012). The traditional approach often used in the teaching and learning
1 INTRODUCTION

processes of introductory programming puts a greater emphasis on syntax and semantics of programming languages which results in students struggling to achieve the learning outcomes set for the course. The approach introduced in this study aims to ensure that equal attention is given to syntax, semantics and problem solving strategies thus improving students’ learning in the course. The current research helps computer science discipline in achieving students learning outcomes set for the course.

Tavares et al. (2001) found that curriculum organisation and teaching methods were the two main factors which had significant impact on high failure rates in IP courses. Meisalo et al. (2002) pointed out that 30% of their programming course students dropped out from the course because they found programming exercises too hard and difficult. This study introduces a new presentation style for the curriculum of the IP course at Buraimi University College. A supporting editor is also developed to encourage students to follow the stages in the new approach.

This research focuses on promoting deep learning for the novices and discouraging programming shortcuts as shown in figure 1.1, which will improve student outcomes in the course and, ultimately, improve participation in computer science.

1.5 Research approach

The goal of this study was to investigate how student learning outcomes may be enhanced in the IP course. To achieve this goal, the overarching question was broken down into four research questions as described in section 1.3 above. An action research methodology was adopted, consisting of three cycles to explore the research questions.
1 INTRODUCTION

In cycle 1, the barriers and affordances to learning programming were explored from the perceptions of students in, and instructors of, the IP course. A survey was deployed to students, to gather their perceptions of learning programming, with the aim of addressing the first research question:

*RQ1: What are the perceptions of students of the barriers and affordances to learning programming?*

The perceptions of the IP instructors (through analysis of teaching and assessment materials and a focus group) were probed in two cycles. In the first cycle, it was collected by analysing the current teaching practices and materials offered with the traditional approach with the aim of answering the following research question:

*RQ2: What are the perceptions of instructors of the barriers and affordances to teaching programming?*

In cycle 2 of the action research, the findings of cycle 1 were used to improve the teaching and learning process of the IP course. The ADRI approach was incorporated into the course and the teaching materials were prepared based on the ADRI approach and offered to the students in semester 1, 2014-15.

The impact of the ADRI approach on the course was explored from the perspective of both students and instructors of the IP course. The students were surveyed using the same survey as in the first cycle. The results of both surveys were compared to determine the impact of the ADRI approach on the course materials compared to the traditional approach. A focus group was conducted with the instructors to obtain their in-depth
1 INTRODUCTION

feedback about the ADRI approach in the course. The outcomes of cycle 2 addressed the following research question:

*RQ3: What is the impact of applying the ADRI approach on introductory programming course materials?*

In cycle 3, the ADRI approach was improved based on the feedback received from staff and students in cycle 2. The teaching materials were incorporated with the improved ADRI approach and offered to the students of semester 2, 2014-15. The grades of the students in the four semesters involved in this study were compared to determine the impact of the ADRI approach on student learning. The teaching materials based on the traditional and ADRI approaches were also compared to determine the impact of the ADRI approach on their design and implementation. Consequently, the outcomes of cycle 3 answer the following research question:

*RQ4: what is the impact of applying the ADRI approach on student learning?*

1.6 Organisation of the thesis

This thesis is presented in 7 chapters. Chapter 1 (this chapter) provided an overview of introductory programming courses, the importance of programming in the computer science discipline, and the issues relating to learning programming identified in the literature. It also describes the purpose and significance of the study and provides an overview of the research approach.

The literature on introductory programming relevant to this thesis is discussed in Chapter 2. The chapter begins with a brief background to learning to program. The following
sections describe the three entities of the didactic triangle (novices, instructor and content) used in the programming context. A brief overview of different programming teaching methods, mobile supported learning, and supporting software tools is discussed. Finally in chapter 2, different problem solving techniques and educational taxonomies used in programming are described.

Chapter 3 focuses on the research methodology and the design followed for this study. The chapter begins with a brief introduction to research methodology, research design, and quantitative and qualitative methods. The next section describes the action research methodology adopted for this research. The remaining sections of this chapter discuss learning theories, the three entities of the didactic triangle, themes and goals of the action research, methods of data analysis, ethical consideration, delimitations and limitations of the study.

Chapter 4 describes the first cycle in the action research used in the study. The chapter starts with a brief introduction to how cycle 1 was conducted together with the research questions being investigated in this cycle. The next section discusses the four stages of the ADRI model. Results of the first survey and analysis of the IP course parameters are also presented. The chapter concludes with a discussion of the results of cycle 1 and how these impacted on the activities in cycle 2.

Chapter 5 presents the second cycle of the action research used in the study. The chapter begins with a brief discussion of the feedback received from cycle 1 along with the research questions being investigated in this chapter. The next section describes the four stages of the ADRI model as implemented to support the learning of programming by
novices. The design of the ADRI based teaching materials and the supporting ADRI editor are discussed. A comparison of the results from the first and second student surveys, findings of the focus group conducted with the instructors and discussion of the outcomes conclude this chapter.

Chapter 6 describes cycle 3 of the action research and starts with a brief discussion of the feedback received from cycle 2 together with the research questions being investigated in this chapter. Then follows a discussion of the recommendations incorporated into the ADRI approach based on the outcomes of cycle 2. Analysis of the ADRI based teaching materials is presented. Analysis of the students’ final grades for the four semesters involved in this study and a comparison of the traditional and ADRI based teaching materials are presented. The chapter ends with a discussion on the findings of cycle 3 and the conclusions drawn.

The concluding chapter, Chapter 7, brings the findings of all three cycles together, summarising the outcomes of this research and exploring their implications. The contribution of the research to the body of knowledge is discussed and some recommendations are made for both future research and practice.
2 Literature Review

2.1 Introduction

Learning to program is considered to be difficult and challenging for a significant number of novice programmers (Reardon & Tangney 2014; Shuhidan, 2012). As a consequence high failure and dropout rates from introductory programming (IP) courses have been reported (Guzdial & Soloway, 2002; Lahtinen et al., 2005; Sykes, 2007; Yadin, A. 2011; Watson & Li, 2014; Zingaro, 2015). Many studies have been conducted to determine the difficulties and challenges faced by novice programmers (section 2.2). Tavares et al. (2001) identified curriculum organisation and teaching methods as the two main factors for high failure rates in IP courses and these are discussed in section 2.3. Meisalo et al. (2002) pointed out that 30% of their programming course students dropped out from the course because they found programming exercises too hard and difficult. Other studies include: Robins et al. (2003) on the educational aspects of novice programmers; Soloway and Spohrer (1989) presented a comprehensive literature review on novice programmers and Ala-Mutka (2004) discussed learning and teaching problems for novice programmers. Wiedenbeck et al. (2004) concluded that:

‘Decisions about majoring in computer science and related fields are often determined by a student’s success or failure in the introductory course. If a student drops out, fails, or passes with a struggle, that student is unlikely to enrol for a follow-on course. In spite of research on factors that influence the enrolment and success of novices in introductory programming, it is still not fully understood what makes an introductory programming course positive and successful for some, but difficult and frustrating for others.’ (p. 97)
This study proposes that learning to program requires students to pay equal attention to programming knowledge (syntax and semantics) and problem solving strategies (Iqbal and Harsh, 2013). Winslow (1996) argued ‘that novice programmers know the syntax and semantics of individual statements, but they do not know how to combine these features into valid programs’ (p. 17). Robins et al. (2003) discussed that ‘typical introductory programming textbooks devote most of their content to presenting knowledge about a particular language’ (p. 141). De Raadt (2008) examined 40 programming textbooks and discovered that only a small proportion of them (6 out of 40) integrated problem solving strategies throughout the book. He further inquired from introductory programming course instructors within universities of Australia and New Zealand in his 2003 census how much time was spent on teaching problem solving strategies. ‘Some participants reported that problem solving strategies were not included in their courses’ (p. 8).

Ala-Mutka (2004) discussed that learning to program consists of several activities like ‘learning the language features, program design and program comprehension’ (p. 3). But the traditional approach used in teaching programming courses and in textbooks is to start with programming knowledge about a particular language. On the other hand, less attention is given to learning strategies to solve programming problems.

To address these issues, the main goal of the current research is to propose a teaching approach for IP courses which incorporates programming knowledge and problem solving strategies by giving all aspects equal emphasis.

Zingaro (2015) conducted a quasi-experimental study in a computer science 1 (CS1) course to explore how the pedagogy approach (traditional lecture vs. Peer Instruction) in
combination with student achievement goals relates to ‘exam grades, interest in the subject matter, and course enjoyment’ (p. 1). He concluded that the pedagogy approach improves the students’ interest in subject matter and it is an important factor in improving students’ outcomes in CS1.

2.2 Learning to Program

Learning to program is one of the challenges for a significant number of novices in computer science and related disciplines. The key challenge in learning programming is to acquire different sets of skills at the same time. Novices have to learn both the syntax and semantics of a programming language while developing problem solving skills. Further, programming courses require students to study theoretical concepts and practice these concepts when designing and developing programs. Therefore, novice programmers need to learn different sets of concepts in parallel, and have to apply these in a practical setting concurrently, pushing the cognitive processes of many into overload. Very simplistically, cognitive processes involve transferring information in the brain and translating it into knowledge through the brain’s capacity to process information (Shuhidan, 2012).

2.2.1 Novice Programmers

The Oxford English Dictionary (2014) defines novice as ‘a person new to and inexperienced in a job or situation’. Researchers defined novice programmers as those who are new, beginner or in the first stages of becoming a programmer (Thomas et al., 2004). In our context, novice is normally a first semester student within the Information Technology (IT) department who has little or no prior experience or knowledge of
programming. Generally, programming is considered an important component of computer science study plans and it can lead to a rewarding career (Robins at el, 2003).

It is generally accepted that novice programmers need 10 years of experience to become an expert programmer (Winslow, 1996). Dreyfus and Dreyfus (1986) proposed five stages for novices to become experts: novice, advanced beginner, competence, proficiency and expert. Winslow (1996) suggested that four year undergraduate degree graduates would rank between competent and proficient.

Some of the characteristics of novices drawn from various studies are:

- They have fragile knowledge which means they know the concepts but they fail to recognise when to use them (Perkins and Martin, 1986).
- They lack a mental model of the area (Kessler and Anderson, 1989).
- They emphasise surface knowledge of the subject (Winslow, 1996).
- They use general problem solving strategies rather than problem domain dependent strategies (Winslow, 1996).

2.2.2 Programming knowledge and strategies
Programming knowledge covers the syntax and semantics of programming languages. On the other hand, programming strategies include problem solving approaches, algorithms and other methodologies to solve a problem (Davies, 1993; De Raadt, 2008). Soloway (1986) stated that their teaching curriculum explicitly covers programming knowledge. By contrast, programming strategies, which are equally important because they provide ways to solve problems by applying programming knowledge, are taught implicitly.
2 LITERATURE REVIEW

Novices are expected to develop their own programming strategies by implicit learning (Rist, 1991).

De Raadt (2008) introduced explicit instruction on programming strategies (Goal/Plan) in an introductory programming course. The results show that explicit incorporation of programming strategies may improve outcomes for novice programmers.

2.3 Introductory Programming Course design & Teaching Model

Mahmoud et al., (2004) argued that course design is considered one of the reasons for high failure and dropout rates in IP courses. So they designed a course with ‘two goals in mind: to improve the students’ experience in their first computer programming; and to achieve retention in the new program’ (p. 120). They designed an introductory programming course featuring HTML, JavaScript and Java. They introduced a ‘programming for fun’ approach, students start programming with HTML and JavaScript, programs are run in browsers, which are more supportive in handling errors compared to conventional programming languages. This approach presented introductory programming concepts to students without having to worry about the compilation overhead. ‘After working with JavaScript for 3-4 weeks, students are well prepared to move on to Java’ (p. 122). Pair programming was used, to share knowledge among students, in both labs and assignments. They achieved very good results in their first offering of the course, with all students passing with a high average mark.

Tirronen et al. (2011) presented a course design for an introductory programming unit which promoted a learning-oriented (Klug, 1976) and learner-directed learning process. Their main goal was to construct the content of the course based on questions raised by
students. The course design was based on two hours per week allocated for supervised sessions and assignment review and ten hours independent study by students in between. They called it ‘test-driven teaching’ because the assignment was given before any teaching had taken place. Before the supervised sessions students attempted to solve the exercises and sent their questions to the teacher which helped the teacher to prepare topics for the session based on the concerns raised. Before the assignment review session, students worked in groups of three and were encouraged to complete weekly assignment work. During the review session, assignments were presented by a randomly selected member of the group so it was the responsibility of all group members to ensure they had a thorough understanding of all the assignments.

Peer-learning (Boud et al., 1999) was promoted during the review session because students presented their solutions and reviewed the solutions of others. The course was evaluated on a pass or fail criteria rather than a grade. They did not include an exam and the course was passed by completing all the assignments’ or exercises. The pedagogic rationale behind the teaching method was to promote self-directed learning (Candy, 1991; Boyer et al., 2008). Candy (1991) argued that self-directed learning moves the control from the teacher to the students and hence promoted increased motivation and novelty. The course was designed as a pass/fail program but in a credit point environment a particular grade (or marks) needs to be awarded rather than just awarding a pass or fail.

Webster (1994) discussed that in most cases novice programmers attempt to write code directly to solve the problem statement (Problem StatementÆCode) as shown before in Figure 1.1. Iqbal and Harsh (2013) discussed how the traditional approach used in the learning process of the introductory programming course also promotes this practice. All
the programming examples discussed during the lectures and exercises solved in the lab sessions are based on this practice. The task in the examples and exercises is to convert the problem statement directly into a computer program which promotes students taking programming shortcuts. Most programming editors encourage this practice and promote programming shortcuts.

Corney et al. (2010) found that there was a correlation between students who failed the IP course and dropped out of the degree program. They argued that lack of motivation is one of the factors for poor performance of novices in IP courses. A new course structure was introduced to address the retention problem and to improve student engagement and progression. They introduced a course which covered some introductory programming, as well as introducing database, web development and networking for first semester students. The basic topics covered in programming in this unit included sequence, selection, iteration and function. Collaborative learning (Wilson et al., 1993) was introduced which provided a conducive environment for learning and the students benefited from peer support and pressure (McKinney & Denton, 2006). Paired students supported each other in completing exercises, assessment tasks, and small projects. Tutors ensured that paired students swapped the roles of driver and navigator (section 2.3.2 on paired programming). A reflective report (Kay et al., 2007) was introduced for the students to compare their initial and final skills in different topics of an IP course. The report covered questions related to acquisition of knowledge, whether students enjoyed their studies and whether they would like to enrol in advanced units related to the topic. ‘Assessment for this unit is a combination of a portfolio of activities undertaken during the semester, a reflective report comparing initial and final skills in the areas taught in the
unit, and weekly online quizzes’ (p. 6). As a result, the failure rate and attrition in the course was reduced to 6%.

Linn and Clancy (1992) introduced case studies of programming problems. Each case study includes: ‘(a) a statement of the programming problem; (b) a narrative description of the process used by an expert to solve the problem, written so that a student can understand the expert’s approach; (c) a listing of the expert’s code; (d) study questions to provide practice in program design, problem solving, and analysis; and (e) test questions to assess student’s understanding of the problem solution’ (p. 121). Their discussion of these case studies with students helped the students learn how to apply general problem-solving skills to specific domains. Further, the case studies implemented eight principles of program design (‘the recycling principle, the multiple representation principle, the alternative paths principle, the reflection principle, the fingerprint principle, the divide and conquer principle, the persecution complex principle, the literacy principle’ p. 128) and offered effective computer programs due to provision of a wide range of instructional techniques.

Linn and Clancy (1992) further suggest that it is better to use formal methods such as pseudo-code or flowcharts rather than narrative descriptions to solve problems so that students can develop the frameworks to use this knowledge in other contexts. Moreover, they further suggest that ‘[a]nalysis of the cognitive demands of the programming design task make it clear that instruction that focuses on the syntax of a programming language or on simplified design techniques such as Top-down design are unlikely to impart the skills that individuals need to be effective programmers. Introductory textbooks, written under page constraints and emphasizing many unrelated programming techniques rather
than a few commonly occurring principles, tend to encourage this kind of instruction. As long as this is the case, only the small number of students who are capable of inferring program design skills via unguided discovery will succeed. The success of these case studies indicates that techniques can be found to teach program design skills and therefore make programming accessible to a wider range of students.’ (p. 130)

Kaasbøll (1998) discussed three didactic models for introductory programming: semiotic ladder, cognitive objectives taxonomy, and problem solving. In the semiotic ladder model, teaching and learning is performed in the sequence of syntax, semantics and pragmatics of the language-like tools. The sequence of instruction in the cognitive objectives taxonomy ‘comprised using an application program, reading the program, changing the program, creating a program may also be added’ (p. 196). He described the problem solving model as ‘Through solving problems, the students should extend their experience and repertoire of practice and the basis for the process is the knowledge structure of the field of programming. The problem solving process is guided by methods and environments’ (p. 196). He conducted five exploratory interviews with instructors of an introductory programming in Australian universities, including a question about their teaching model. Most of the instructors did not give any answers. He presented the above mentioned three models but the responses indicated that most courses had no clear teaching model for an IP course.

The didactic triangle (Kansanen, 1999), shown in figure 2.1, is used to explain the relationship between teacher, content and student in introductory programming courses. These three entities are important in teaching and learning environments. The student is taught by the teacher, who guides the student through the contents. The contents or
teaching materials are available for both student and teacher. The interaction or relationships between these three entities are important to facilitate the learning process.

**Figure 2.1: The didactic triangle (adapted from Kansanen, 1999; Kinnunen, 2009)**

In this study, I use the didactic triangle to discuss typical students (characteristics of novice programmers), teachers (programming teaching methods) and content (learning resources) in the following subsections.

**Figure 2.2: The didactic triangle (in the context of this study)**
2 LITERATURE REVIEW

Kinnunen (2009) argues that the context of the learning situation should also be considered because the learning process does not occur in a vacuum. In this study, the context of learning is Buraimi University College which is discussed in detail in section 3.6.3.

2.3.1 Characteristics of novice programmers
The IP course is one of the first courses novices take in computer science education (Matthiasdottir 2006). Novices have to take it to progress in their study plan. Many face difficulties in the first few weeks of their programming course. They are unsuccessful when attempting to write computer code that meets stated objectives (Carbone et al., 2009). The whole situation de-motivates them and causes frustration which leads to disengagement with the curriculum. As a consequence, high failure and dropout rates are reported (Watson & Li, 2014; Zingaro, 2015).

Novices start programming with limited surface knowledge and skills. As a consequence, they generally approach programs ‘line by line’ rather than understanding the overall structure or ‘chunks’ of programs (Winslow, 1996).

They normally spend little time planning or testing code (Lane et al., 2012) and try to solve problems in the context of coding rather than understanding the broader sense of programs (Kölling and Rosenberg, 1996). Ala-Mutka (2003) mentioned that program planning is one of the important activities in learning programming languages. Further, novices spend more time learning programming language syntax as this is comparatively easy compared to developing the ability to write an algorithm that solves a given problem efficiently (Ansari, 2011).
Novice programmers’ general intelligence or mathematical abilities are also related to their success in learning programming languages (Bruce-Lockhart et al., 2000). Researchers also investigated whether some programming language constructs are more difficult for novice programmers to understand than others. For example, variable initialisation is more difficult to understand than updating or testing variables (Soloway, et al., 1989).

Spohrer et al. (1986) suggested that some novices believe that programming is difficult because they need to grasp a significant amount of knowledge and different skills at the very beginning. Additionally, novices do not pay full attention to understanding tasks given by the instructor. This may lead them to a solution which does not meet the program specification (Ginat et al., 2004).

2.3.2 Teaching Methodologies and Styles
The teaching method is an approach by which educators deliver teaching materials to the students and is one of the crucial factors that help to motivate students master the course content (Mohorovicic and Strcic, 2011). Students’ achievement of the learning outcomes is also affected by their learning style and motivation. De Raadt and Simon (2011) suggested that it is a good idea to design teaching materials which covers all VARK (visual, aural, read/write, and kinesthetic) modalities. They concluded that the learning materials prepared to cover both kinesthetic and read/write preferences suited all their students who participated in a study. Moreover, ‘If learning materials are designed with good textual material to suit the read/write modality and good hands-on experience to suit the kinesthetic, such materials should suit the learning preferences of virtually all of the students’ (p. 111).
Papp-Varga et al. (2008) argue that ICT teaching is a comparatively new ‘problem domain’ as compared to well established subjects like physics or mathematics. So its teaching methodology is not well established or formulated and, as a consequence, different teachers are using their own ‘blend of methods’ (p. 163).

Gomes and Mendes (2014) interviewed 18 instructors of an introductory programming course about their pedagogical and motivational strategies. They concluded that ‘student-teacher relationship, teachers’ proximity with the students, class competitions, challenges and strategies in which the students are engaged actively and continuous assessment’ (p. 7) are considerable motivational strategies. Moreover, most teachers’ do not consider this course different from other courses and use the same traditional pedagogical strategies in it. Some teachers’ emphasised that a good choice of exercises and teaching materials also plays an important role in the teaching and learning process.

Jenkins (2002) classified learning styles into two approaches (surface and deep). Surface learning concentrates on memorising the facts, while deep learning gives in-depth understanding of a topic. In programming, surface learning is required to memorise the syntax of a language while deep learning is needed to understand the programming logic, problem solving strategies and, consequently, build competence in programming.

Winslow (1996) suggested that models of data structure, program design, problem domain and data representation are all important in teaching programming. If the instructor omits any one of these models from the teaching process, ‘the students will make up their own models of dubious quality’ (p. 21). The instructor should teach initial
facts, models and rules at a simple level and ‘only expand and refine them as the student gains experience’ (p. 21).

Furthermore, Winslow (1996) argued that practice is an important part in programming and designing practice questions for programming syntax is easy compared to designing problem solving questions. In related research, Huet et al. (2004) introduced a block of four hours (one hour of lecture followed by three hours of laboratory work) in an IP course to promote practice. The students’ feedback suggested that a block of three hours is very hard for them. On the other hand, the block of three hours helped in conducting small evaluation opportunities throughout the semester.

Schneider (1978) proposed ten principles for the IP course in computer science:

‘Principle 1: Students should immediately be taught that a clear, concise problem statement is always the first step in programming.

Principle 2: The single most important concept in a programming course is the concept of an algorithm.

Principle 3: It is important to introduce the duality of data structures and algorithms in the programming process.

Principle 4: Choose a programming language that enhances the learning process.

Principle 5: The presentation of a computer language should concentrate on semantics and program characteristics not syntax.

Principle 6: The presentation of a computer language must include concerns for programming style from the very beginning.
Principle 7: The subject of debugging should be formally presented.

Principle 8: The subject of program testing and verification should be formally presented.

Principle 9: The subject of documentation should be formally presented.

Principle 10: A student should be introduced to realistic programming applications and realistic programming environments.’ (pp. 107-112)

The current study focuses on achieving some of these principles discussed in section 4.2.

_Programming teaching methods_

A teaching method involves the methods and principles used to instruct students in achieving the desired learning and it is implemented by teachers. An overview of some existing programming teaching methods as follows:

_Problem-based learning_

Nuutila (2005) explained that problem-based learning (PBL) engaged students in problem solving. In PBL, the students will face real world problems which help them to enhance their ‘disciplinary knowledge, higher order thinking and practical skills’ (Mohorovicic and Strcic, 2011; p. 49).

Seven steps(Examination of the case, Identification of the problem, Brainstorming, Sketching of an explanatory model, Establishing the learning goals, Independent studying, and Discussion about learned material) are involved in PBL implementation (Nuutila, 2005) and students work in groups. They analyse the problem and determine what they already know about it and sort out what they need to learn to solve the problem.
After that the students work independently to read all the required materials. The students then sit again and discuss what they have learned in order to solve the problem. Finally, they elaborate on their solutions.

Researchers concluded that PBL enhances retention time of students’ knowledge up to several years and students’ performance in follow up courses is better than those who attended traditional introductory programming courses. It also helps to enhance students’ creative thinking and motivation (Nuutila, 2005).

**Puzzle-based learning**

The main aim of puzzle-based learning (PZBL) is to teach students problem solving and critical thinking techniques (Merrick, 2010). In programming courses, PZBL is taught by dividing the problem solution into a number of puzzle pieces. The student reconstructs the program by putting puzzle pieces in correct order and his or her performance is evaluated by looking at the reconstructed solution (Yoneyama at el., 2008).

A research study conducted by Merrick, 2010 showed that PZBL increased students’ interest and active participation in the programming course.

**Pair Programming**

In pair programming, two programmers work on the same program (code) side by side at the same computer. Both programmers are involved in planning, designing and testing the program (code).

Each programmer has a different role in pair programming. One programmer works as a driver and other as a navigator. The driver is typing actual code and navigator is
observing his work for errors, offering suggestions and alternative solutions. The role of driver and navigator is changed at regular intervals between programmers.

Researchers have suggested that pair programming motivates both programmers. They devised solutions in less time and with fewer errors and they approached collaboration in a positive way (Zacharis, 2011).

Some studies revealed that pair programming can be exhausting and irritating. So appropriate care should be taken while organising the pairs. Different skill level between pair programmers also affects collaboration (Chaparro et al., 2005).

Huet et al. (2004) discussed that some lecturers feel that pair programming is less efficient than individual work for introductory programming courses. They observed that one of the students actually programs while the other student becomes an observer. On the other hand, some lecturers like pair programming because it promotes learning and one of the students can serve as a tutor for the other.

*Game-themed Programming*

Game-themed Programming (GTP) is a teaching approach in which abstract programming concepts are taught to the students by exploring small game applications. The main aim of GTP is not to teach game programming to the students, but to help them understand programming concepts through simple game assignments (Sung, K., et al. 2011).

A study revealed that success rates in GTP classes were higher than in other traditional classes. Students were engaged in devising solutions for game assignments which increased their motivation and enthusiasm (Sung, K., et al. 2011).
2 LITERATURE REVIEW

Miljanovic (2015) used game based learning to teach debugging to novices. He discussed that debugging is a critical skill that novices should acquire early in their programming career. Programmers who lack this ability spend hours attempting to fix the errors in their programs. For novices, it means that very little progress is made over time because most of the time is spent on finding and fixing the errors. He prepared a game (RoboBUG) for the novice to learn debugging techniques. The results show positive impact on the learning process of novices.

Prerecorded lectures

Smith and Fidge (2008) introduced recorded ‘mini’ lectures in an introductory programming course to enhance students’ performance. Prerecorded lectures (PL) are narrated slides from the teaching materials. They provide supplementary support to the students for conventional lectures. PL also helps students in catching up with missing sessions, as they usually available online. It particularly helps students who fail to absorb or understand class lectures by allowing them to revisiting the lectures at their own pace. Unlike text books, PL can separate important topics from less important ones (Smith and Fidge, 2008).

Researchers concluded that students’ feedback about PL was very positive. Most of the students stated that PL helped them in understanding some programming concepts in a better way (Smith & Fidge, 2008).

2.3.3 Curriculum and content delivery

The ACM-IEEE Joint Task Force on Computing Curricula (2013) provides guidelines for computing undergraduate programs. The latest revision, Computer Science Curricula
2 LITERATURE REVIEW

2013 (CS2013), provides comprehensive guidance on curricular structure and development in a variety of educational contexts (ACM-IEEE Joint Task Force on Computing Curricula, 2013).

The CS2013 body of knowledge is organised into 18 knowledge areas (KA). The most important programming related KAs include Programming Languages (PL), Software Development Fundamentals (SDF), Algorithms and Complexity (AL) and Software Engineering (SE). Computer programming concepts and skills topics are introduced in these KAs for introductory and advanced computer science programming courses.

Software Development Fundamentals (SDF) is a newly introduced KA in CS2013. It focuses on the entire software development process in introductory programming. Due to its broad spectrum, it includes KAs which could be included in other software-oriented KAs (e.g. programming constructs and problem solving (PL), development methodologies (SE), and algorithm analysis (AL)). SDF provides basic concepts and those KAs cover advanced topics in the above mentioned areas.

Computer programming knowledge (syntax and semantics) and problem solving strategies are given equal attention in these KAs. Problem solving skills are also included as one of the characteristics of computer science graduates which means that graduates should know how to identify and design solutions to real world problems instead of just writing code (ACM-IEEE Joint Task Force on Computing Curricula, 2013). The Australian Computer Society (2013) also suggested emphasising both problem solving strategies and programming knowledge in the teaching and learning of programming languages.
De Raadt et al. (2005) reviewed forty textbooks prescribed to students in introductory programming. They found that ‘Simple Program Design’ (Robertson, 2004) is the most widely used textbook. It covers problem solving strategies related to programming. It is not targeted to a specific computer language, so this book is used in combination with a language specific text in introductory programming courses.

Problem solving topics were covered in varying degrees in the analysed books. De Raadt et al. (2005) mention that only six books out of forty investigated integrated problem solving aspects throughout, implying that most of the analysed books focused only on language syntax. Some authors provide a large number of examples and put less emphasis on problem solving instructions. Some books cover problem solving topics in early chapters but they are not integrated in the remainder of the book. There are a small number of books where problem solving is integrated throughout the books using case studies and examples. (De Raadt et al., 2005)

Winslow (1996) stated that ‘the old saw that practice makes perfect has solid psychological basis’ (p. 18). Practice is an important component in learning process of programming. If novices want to become expert programmers then they should practice, practice and practice programming problems (Winslow, 1996).

Hook and Eckerdal (2015) analysed the results of the final exam in an introductory programming course and compared it with the study behaviour of the students, explored through a questionnaire. They concluded that those students who had higher marks in the course spent more time on the computer in practicing programming problems. On the other hand, those students who spent more time on reading books and attending lectures
but did not spend time on practicing programming problems gainedless marks. So it is important to encourage students to spend more time on the computer in practicing programming exercises.

2.4 **Mobile device supported learning**

Mobile device supported learning provides additional support for accessing introductory programming teaching materials to students (Iqbal et al., 2013). It provides teaching materials anywhere, anytime and eliminates space and time constraints for learners. It also allows interaction between teachers and learners without physical contact (Tapia-Moreno, 2012). The work of Mephee et al. (2008) suggested that the retention of specific subject knowledge in first year university students for a mobile learning group was better than for those who were not supported in this way.

Surveys conducted with novice programmers concluded that the overwhelming majority of participants agreed that mobile device supported learning provides additional support to access and understand teaching materials. The teaching materials can be saved in mobile memory which can be accessed at the learners’ convenience. Furthermore, the students can access recorded class lectures along with other teaching contents. It can help them to catch-up with missed classes (Iqbal et al., 2013).

Reardon and Tangney (2014) explored ‘how smartphones, studio-based learning and extensive scaffolding could be used in combination in teaching of a freshman introduction to programming course’ (p. 13). They concluded that studio-based learning is an effective pedagogy for an introductory programming course when used in conjunction with application development with smartphones. Moreover, ‘the findings support the view that
developing applications for smartphones leverages off the motivational, authenticity, and contextualized affordance of mobile learning’ (p. 13).

2.5 Introductory Programming Supporting Software Tools

Gomez-Albarran (2005) reviewed supporting software tools used in teaching programming. He categorised supporting tools into four groups as follows:

- Tools with a reduced development environment
- Example-based environments
- Tools based on visualisation
- Simulation environments

*Tools with a reduced development environment* help novices to overcome the obstacles of understanding the commercial development environments, which are developed for professional programmers. Examples of reduced development environments include BlueJ (Barnes and Kölling, 2003), DrJava (Allen et al., 2002), AnimPascal (Dagdilelis et al., 2002), THETIS (Freund and Roberts, 1996) and Racket (Flatt & Findler, 2011).

*Example-based environment* helps programmers to use previously written or learned programs to solve new problems. This approach is also reflected in the software reuse area. The four examples of example-based environment are ELM-PE (Episodic Learner Model—the programming environment) (Weber, G. 1996), ELM-ART (Episodic Learner Model—the Adaptive Remote Tutor) (Weber and Brusilovsky, 2001), WebEx (short for Web Examples) (Brusilovsky, 2001) and Javy (Gómez-Martínet al., 2003).
2 LITERATURE REVIEW

Tools based on visualisation help students in achieving learning because ‘human beings are good in processing visual information’ (Gomez-Albarran, 2005; p.135). Visualisations have been used in learning for a long time, and help students to learn abstract and complex concepts of programming (Ala-Mutka, 2004). ANIMAL (A New Interactive Modeler for Animations in Lectures) (Rößling and Freisleben, 2002) and LEONARDO (Crescenzi et al., 2000) are two examples of visualisation tools used for teaching programming.

Simulation environments help students to understand programs and their effects by means of visualizations in an imaginary world. ‘Students observe (and sometimes take active part in) this imaginary world, whose inhabitants behave according to the execution of program instructions. So program actions are reflected in inhabitants’ actions’ (Gomez-Albarran, 2005; pp. 138-139). Alice (Cooper et al., 2003) and Karel the Robot (Pattis, 1981) are well known examples of simulation environments.

2.6 Problem Solving Techniques

De Raadt (2008) defines problem solving as ‘A mechanism for achieving a solution to a programming problem’. It is commonly accepted that teaching problem solving techniques is not an easy task (Allan & Kolesar, 1997). Additionally, different researchers have shown that students in IP courses donot concentrate on developing problem solving skills and instead spend more time on coding the programming problems (Yoo et al., 2012).

In 1945, George Polya identified four basic principles of problem solving in his book ‘How to Solve It’ (Polya, 1945). Table 2.1 shows his four principles of problem solving.
Table 2.1: Polya four principles of problem solving

<table>
<thead>
<tr>
<th>First principle:</th>
<th>Understand the problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second principle:</td>
<td>Devise a plan</td>
</tr>
<tr>
<td>Third principle:</td>
<td>Carry out the plan</td>
</tr>
<tr>
<td>Fourth principle:</td>
<td>Look back</td>
</tr>
</tbody>
</table>

Thompson (1996) proposed four stages of ‘How to program it’ based on Polya’s principles. He identified the four stages as follows:

1) Understanding the problem
2) Designing the program
3) Writing the program
4) Looking back

*Understanding the problem:*

The first important step is to understand the problem. What are the inputs (arguments) and outputs (result)? Is the specification of the problem complete and understandable? Is it possible to break down the problem into parts?

*Designing the program:*

Think about the connections between the inputs and the outputs of the program. ‘If there is no immediate connection, you might have to think of auxiliary problems which would help in the solution’ (Thompson, 1996; Para. 2).
Has a similar type of problem been encountered before? The main purpose is to develop a plan of how to write the program.

*Writing the program:*

In writing the program, a particular programming language is used to start implementing the design. Develop the solution by dividing the program into smaller parts. Use parts of other programs to carry out the solution.

*Looking back:*

Test the program with different test cases. How can this program be reused to devise other programs?

Winslow (1996) discussed problem solving learning processes for students. He emphasised that, for novices, understanding programming language syntax, semantics and problem solving skills are equally important. He divided program problem solving into four steps:

1) ‘Understand the problem
2) Determine how to solve the problem:
   a) in some form, and
   b) in computer compatible form
      Note that novices have trouble going from ‘a’ to ‘b’)
3) Translate the solution into computer language program, and
4) Test and debug the program’ (p. 19)

Soloway (1986) suggested *Goals* and *Plans* as a problem solving strategy for analysing problems and constructing programs. ‘With a given problem, the process begins with the
instructor determining the goals that need to be achieved to solve the problem. These goals are then mapped to plan’ (De Raadt, 2008; p. 26).

Hyde et al. (1979) developed the Problem Solving Process (PSP) for an introductory programming course which promoted problem solving skills in students. PSP consists of the steps shown in Figure 2.3. It is clear that PSP requires students to finish four steps before typing the program on computer. The PSP promotes problem solving skills in students.

Figure 2.3: Problem Solving Process (PSP) Hyde et al. (1979)
Koulouri et al. (2015) conducted a study in an introductory programming course to determine the benefits of problem solving training before programming. They compared the performance of two groups of students with and without problem solving training. They revealed that students provided with problem solving training performed better than those without problem solving training. They discussed a possible explanation was that novice programmers start working with syntax and semantics of the language without considering or analysing the problem statement. The problem solving training helps students to understand and interpret the problem in terms of devising a programming solution, which they can then translate into lines of code.

### 2.6.1 Program development by stepwise refinement

Wirth (1971) discussed the program development process as a gradual stepwise refinement process. He said that it is common practice to teach programming by examples. Furthermore, these examples play a crucial role in the success of a programming course. Normally, examples of programs are shown as a finished product and their syntax explanations are presented in detail. On the other hand, active programming requires new programs to be designed. Therefore it is important that teaching methods for programming courses should concentrate equally on design and construction of programs. He demonstrated one programming example which was gradually developed in a sequence of refinement steps. In each refinement step, program instructions are decomposed into more detailed instructions and some design decisions are taken.
2.7 Educational Taxonomies

Educational taxonomies are effective tools in writing learning objectives and assessing students’ attainment. They can also be used to classify test items (Fullers et al., 2007). Two taxonomies that are widely used in higher education are Bloom’s and SOLO taxonomies. In this study, I also discuss the revised Bloom’s and matrix taxonomies.

2.7.1 Bloom’s Taxonomy

Bloom’s taxonomy was proposed by a committee of educators chaired by Benjamin Bloom in 1956 (Bloom and Krathwohl, 1956). It divides educational objectives into three domains: cognitive, affective, and psychomotor. The affective domain covers emotions, attitudes and feelings towards the learning process. The psychomotor domain focuses on physical movements and uses the motor-skills areas. The cognitive domain revolves around comprehension and critical thinking to attain knowledge. A goal of Bloom’s taxonomy is to create a holistic form of education by motivating educators to focus on all three domains.

![Figure 2.4: Bloom’s Taxonomy: Cognitive Domain (Bloom and Krathwohl, 1956)](image-url)
The cognitive domain consists of six levels as shown in Figure 2.4. In this domain, knowledge is the lowest level and deals with memorising and recalling information. The second level, comprehension, demonstrates understanding of facts and makes use of memorised information. The third level, application, involves solving problems in new situations by applying acquired knowledge and techniques. The fourth level, analysis, breaks information into parts and determines how the parts relate to each other. The fifth level, synthesis, puts the parts together to form a new pattern or propose alternative solutions. The sixth level, evaluation, is the highest level of the cognitive domain, and relates to presenting judgments about information, ideas or materials based on a set of criteria (Isaacs, 1996).

Johnson and Fuller (2006) conducted a study which examined where Bloom’s taxonomy is appropriate for computer science. A group of academics examined 54 assessments that were given to first year students studying computer science. The aim of the study was to determine the level of Bloom’s taxonomy achieved by these assessments. The results presented showed disagreement between the academics who designed and delivered the modules and the group who analysed all the assessment tasks. The assessors realised that most of the assessments were at the application level while the academics who designed and delivered the modules felt that they were also assessing analysis. They presented two reasons for this disagreement. Firstly, it is difficult to determine the ‘taxonomic level of the assessment without having an intimate knowledge of the way in which the material being assessed was taught’ (p. 121). Secondly, the assessors and academics had different understandings of the levels of Bloom’s taxonomy. Another notable finding is that some
academics realised that the highest levels of Bloom’s taxonomy (synthesis and evaluation) were not suitable for first year computer science courses.

‘In some cases it was clear that this was because the convener subscribed to the view that these levels would not be addressed until the final year of the degree programme. In others it seemed that it was because they felt that application was the ‘core’ of what computing is about and so it is appropriate to concentrate on its development in teaching and assessment.’ (Johnson and Fuller, 2006, p. 121)

Gluga et al. (2012) conducted a study on a web-based interactive tutorial (ProGoSs) that helped computer science educators in understanding and practicing Bloom’s taxonomy to classify programming exam questions. The interactive tutorial was tested and evaluated with ten educators. The results showed that the interactive tutorial was effective in developing participants’ understanding of Bloom’s taxonomy in relation to programming exam questions. Furthermore, participants’ self-explanation and reflection revealed that ‘pre-conceived misunderstandings of the categories, or different interpretations about the complexity of tasks and sophistication required to solve them’ (p. 150) were the main reasons why different people choose different categories for the same programming exam questions. It implies that one participant views one programming task at the Synthesis level and another participant views the same task at Application or Knowledge level.

Shuhidan et al. (2009) analysed summative assessment of novice programmers by using Bloom’s and SOLO taxonomies. The short answer questions consisted mainly of multiple-choice questions were categorised with Bloom’s taxonomy. They found that it was difficult to classify questions using Bloom’s taxonomy particularly between
Comprehension and Application levels. All the questions came under three lower levels of Bloom’s taxonomy (Knowledge, Comprehension and Application).

### 2.7.2 Revised Bloom’s Taxonomy

Anderson et al. (2001) revised Bloom’s taxonomy to meet emerging educational requirements of the new century. The revised Bloom’s taxonomy maintained the six levels of the original taxonomy but made changes within the categories. Figure 2.5 depicts the six categories of revised Bloom’s taxonomy. It includes remember, understand, apply, analyse, evaluate and create categories.

![Figure 2.5: Revised Bloom's Taxonomy (Anderson et al., 2001)](image)

Thompson et al. (2008) categorised exam questions for a first year computer science programming course based on the revised Bloom’s taxonomy. They collected exam scripts from six different institutions. Each exam script was examined by five researchers. Initially, they found significant differences in the categorisation according to revised Bloom’s taxonomy among evaluators. They discovered that, in some cases, differences in categorisation were due to understanding of teaching context of assessment task. The course instructor explained teaching context of question and then agreed with author on an appropriate cognitive category for an assessment task. They discussed a question
which could fall in Understanding (‘on the basis that this question required students to provide an example of a familiar concept’ p. 156) or Creating (‘on the basis that it asked students to combine code in a way that they had not seen before’ p. 156) categories of revised Bloom’s taxonomy. They concluded that in order to effectively categorise the questions into Bloom’s taxonomy the person should have an in-depth knowledge of the course.

Alaoutinen and Smolander (2010) conducted a study in a programming course by using a self-assessment tool that could be used to follow progression and motivate learning. A web-based questionnaire was used as a tool with revised Bloom’s taxonomy as a base of its scale. The results of the study show that students could place their knowledge well in Bloom’s taxonomy. Furthermore, the students realised that it helped their learning. It also helped teachers in knowing the level of knowledge gained more objectively than the general scale.

2.7.3 Comparison between original and revised Bloom’s taxonomies

The numbers of categories are the same in both the original and revised Bloom’s taxonomies.

The terminologies in the revised Bloom’s taxonomy were changed from nouns to verbs. Knowledge was renamed as remembering, comprehension was renamed as understanding. The synthesis category was eliminated. Instead, the create category was introduced at the highest level, to explain the re-organisation of elements into new structures or patterns. The revised Bloom’s taxonomy also defined four dimensions of knowledge: (A) Factual knowledge, (B) Conceptual knowledge, (C) Procedural knowledge, (D) Metacognitive knowledge. It helps to map learning objectives into two
dimensional matrices. On the other hand, these added aspects such as ‘procedural and metacognitive knowledge outweighs the simplicity of original scheme’ (Fuller et al., 2007; p. 156).

The original Bloom’s taxonomy is simple; it identifies distinct and recognisable aspects of the cognitive domain. Instructors can assess students based on its six categories. On the other hand, its categories have significant overlap and therefore are not easy to apply. There is also debate about the order in the hierarchy of analysis, synthesis and evaluation (Fuller et al., 2007).

2.7.4 SOLO Taxonomy

Biggs and Collis (1982) introduced the SOLO learning taxonomy; Structure of the Observed Learning Outcome, in 1982. It provides a qualitative way to classify cognitive processes (Biggs and Collis, 1982), focusing on the content of learner’s responses in assessment (Fuller et al., 2007). There are five categories in the SOLO taxonomy: prestructural, unistructural, multistructural, relational and extended abstract. Table 2.2 presents the categories used in the SOLO taxonomy and their descriptions.

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prestructural</td>
<td>No understanding – missed the point</td>
</tr>
<tr>
<td>Unistructural</td>
<td>Understanding of a component of relevant knowledge</td>
</tr>
<tr>
<td>Multistructural</td>
<td>Understanding of a few components of relevant knowledge</td>
</tr>
<tr>
<td>Relational</td>
<td>Able to relate multiple components of relevant knowledge</td>
</tr>
<tr>
<td>Extended abstract</td>
<td>Able to generalize set of knowledge to a new domain</td>
</tr>
</tbody>
</table>

Table 2.2: SOLO Taxonomy (Shuhidan, 2012)
The prestructural category is the lowest level where the learner shows no understanding of the topic. At the unistructural level, the learner demonstrates little understanding of the topic. Next is multistructural where the learner indicates understanding of a few components of the topic. The next two levels show the learner’s strong understanding of the topic. At the relational level, the learner can apply the acquired knowledge to a familiar problem or to a data set. At the highest level, extended abstract, the learner is able to demonstrate application of knowledge to solve new problems or in new domains (Biggs and Collis, 1982; Biggs, 1999; Fuller et al., 2007; Shuhidan, 2012).

Fuller et al. (2007) concluded that the SOLO taxonomy encourages a holistic approach that supports deep learning. On the other hand, ‘there is not yet much reported experience of using it for assessment in a range of subjects’ (p. 156).

Lister et al. (2006) analysed novice programmers reading problems by using the SOLO taxonomy. They suggested that students who could not describe a short piece of code at SOLO’s relational categories were not intellectually well equipped to write similar code. Furthermore, they advocated a mix of reading and writing tasks in teaching and assessing novice programmers.

Clear et al. (2008) analysed 14 introductory programming students’ responses for three ‘explain in plain English’ questions by using the following four categories of SOLO taxonomy (Relational, Multistructural, Unistructural, and Prestructural). They found that it was not always straightforward to classify them. They recommended three more categories in the SOLO taxonomy for analysing programming assessments. They named these ‘the final SOLO categories’ comprising the seven following categories Relational,
2 LITERATURE REVIEW

Relational Error, Multistructural, Multistructural Omission, Multistructural Error, Unistructural and Prestructural categories.

As discussed above, Shuhidan et al. (2009) analysed final exam questions given to novice programmers by using SOLO and Bloom’s taxonomies. They classified questions where novices had to write ‘code to calculate the highest and lowest integer from a set of integers passed via the command line’ (p. 95). They found that to attain the relational classification level of the SOLO taxonomy, novices should understand and connect different components of the solution for a given problem.

Corney et al. (2011) conducted a study with novice programmers by using the SOLO taxonomy. They found that in the third week, almost half of their sample students could not answer simple questions which swapped the values in two variables. They claimed that students faced this problem from the beginning of the semester. Furthermore, because traditional programming exercises were used, this gap in learning remained undetected until the end of semester. New pedagogical approaches should be incorporated into the curriculum to detect and fix these problems.

2.7.5 Two Dimensional Adaptation of Bloom’s Taxonomy – The Matrix Taxonomy

The matrix taxonomy provides a framework for assessing learner capabilities in computer science and engineering. The taxonomy was based on research that indicated that producing program code and comprehension of program code are two semi-independent capabilities. Students who can read programs are not necessarily fully capable of writing programs on their own (Fuller et al. 2007).

The matrix taxonomy used the revised Bloom’s taxonomy as a two dimensional matrix as shown in figure 2.5. The dimensions of the matrix represent two different competencies:
interpreting (comprehension of program code) and producing (designing and building program code). The revised Bloom’s taxonomy categories (remember, understand, analysis, evaluate) are placed on the horizontal axis and the remaining two categories (apply, create) are placed on the vertical axis. The lowest level categories are placed on the lower left corner. Students traverse each axis sequentially. Therefore it is not possible to start to create without getting some competency in apply level (Fuller et al. 2007).

<table>
<thead>
<tr>
<th></th>
<th>Create</th>
<th>Apply</th>
<th>None</th>
<th>Remember</th>
<th>Understand</th>
<th>Analysis</th>
<th>Evaluate</th>
</tr>
</thead>
</table>

**Figure 2.6: Matrix Taxonomy (Fuller et al., 2007)**

Fuller et al. (2007) discussed that different students can take different learning paths by using the matrix taxonomy. For example, when a student learns a new programming concept, he is at None/Remember level. ‘If this student continues with learning by imitating a ready example of a program but without deep understanding of the concept, they will achieve the state "Apply/Remember", i.e. applying/trying to apply the concept without real understanding, with trial and error.’ (p. 164) The Create/Evaluate level can be used for a competent practitioner of a programming concept. Fuller et al. (2007) recommended the use of the matrix taxonomy for designing and assessing programming and software engineering courses.
2.8 Summary of factors influencing introductory programming learning

This chapter presented different factors influencing an introductory programming learning, as well as highlighting areas requiring further research. Table 2.3 lists the factors and current practices used in the teaching and learning processes of an IP course. In this study, these factors were addressed by introducing a new teaching approach (ADRI) instead of the traditional approach to enhance the learning process.

Table 2.3: Summary of research results on factors influencing introductory programming learning outcomes

<table>
<thead>
<tr>
<th>Factors</th>
<th>Summary of research results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning to program</td>
<td>More emphasis is given on syntax and semantics of programming languages (Sections 2.1, 2.2, 2.2.2 &amp; 2.6)</td>
</tr>
<tr>
<td>Teaching strategy/model</td>
<td>Traditional approach spends more time on teaching syntax compared to problem solving strategies. Hence does not achieve students’ learning outcomes and high failure and dropout rates are reported (Sections 2.3.2 and 2.3)</td>
</tr>
<tr>
<td>Programming books</td>
<td>Most textbooks discuss problem solving strategies in an initial chapter only and syntax and semantics are emphasised in rest of the chapters (Sections 2.1 and 2.3.2)</td>
</tr>
<tr>
<td>Learning style</td>
<td>Traditional approach promotes surface learning instead of deep learning because problem solving strategies are not emphasised thoroughly in all topics (Section 2.3.2)</td>
</tr>
<tr>
<td>Lecture notes</td>
<td>Examples discussed in the lectures promote programming shortcut (Problem Statement→Codes) (Section 2.3)</td>
</tr>
<tr>
<td>Lab exercises</td>
<td>Problems given in the exercises promote programming shortcut (Problem Statement→Codes) (Section 2.3)</td>
</tr>
<tr>
<td>Software development tools</td>
<td>Most of the editors used in the introductory programming promote programming shortcut (Problem Statement→Codes) (Sections 2.5 and 2.3)</td>
</tr>
<tr>
<td>Programming language practice</td>
<td>Most students struggle with completing programming exercises because they spend most of their time in debugging their programs (Section 2.3.2)</td>
</tr>
</tbody>
</table>
2.9 Conclusion

Learning to program is one of the challenges for novices in the computing discipline. It mainly consists of acquiring programming knowledge (syntax and semantics) and developing problem solving skills. Different studies have revealed that programming textbooks and course instructors spend most of their time in teaching programming knowledge. Furthermore, the problem solving strategies are given very little time or only discussed in the first few lectures. As a consequence, novices learn syntax and semantics of programming language but are unable to write a valid program. Moreover, high failure and dropout rates are reported.

Different course designs for an IP course were introduced. They promoted pair programming, self-directed learning, peer learning, collaborative learning, case studies and reflective reports. Each of these strategies promotes learning process in an IP course.

De Raadt (2008) pointed out that only a small proportion of textbooks (6 out of 40) integrated problem solving strategies throughout the book. Winslow (1996) suggested that the key for novices to turn into expert programmers is practice, practice and practice programming problems.

The traditional approach used in the teaching and learning process of most IP courses promotes programming shortcuts (Problem Statement→Code) (Webster, 1994). Programming examples and problems are presented in this way. Most of the programming editors also follow the same strategy and promote programming shortcut.

The literature review presented in this chapter has highlighted several areas for further investigation to enhance and achieve students’ learning outcomes in the IP course. The
2 LITERATURE REVIEW

research described in this thesis aims to make a contribution in addressing these learning problems faced by the novices.
3 Research Methodology and Design

3.1 Introduction

This chapter describes and explores the relationship between methodology (how research is to be conducted), methods (how to gather evidence), and epistemology (how to understand the knowledge gained from the research processes) used in this study. The conceptual framework of learning theory, which explains how information is absorbed, processed and retained during learning, is described by constructivism theory (Forrester & Jantzie, n.d.). A research design, as a way to define a logical structure for the inquiry (De-Vaus, 2001) and modes of data collection used in this study are also discussed.

Kothari (2004) defines research methodology and research design as:

‘Research methodology is a way to systematically solve the research problem. It may be understood as a science of studying how research is done scientifically. In it I study the various steps that are generally adopted by a researcher in studying his research problem along with the logic behind them’ (p. 8) [and] ‘the research design is the conceptual structure within which research is conducted; it constitutes the blueprint for the collection, measurement and analysis of data. As such the design includes an outline of what the researcher will do from writing the hypothesis and its operational implications to the final analysis of data’. (p. 31)

Marczyk et al. (2005) differentiates between research methodology and research design as:
‘Methodology refers to the principles, procedures, and practices that govern research, whereas research design refers to the plan used to examine the question of interest. Methodology should be thought of as encompassing the entire process of conducting research (i.e., planning and conducting the research study, drawing conclusions, and disseminating the findings). By contrast, research design refers to the many ways in which research can be conducted to answer the question being asked’ (Kazdin, 2003; p. 22).

There are many ways to do research, and all research designs have their own strengths and weaknesses (Mohamed, 2012). Plano-Clark & Creswell (2010) discussed two different approaches for conducting research, qualitative and quantitative. Quantitative research ‘tends to address research problems requiring a description of trends in a population, an explanation of the relationship among variables’ (Plano-Clark & Creswell, 2010; p. 66). It requires a substantial literature review at the beginning to articulate the research questions, justify the research problem and propose the direction for the study. It focuses on specific and narrow research questions, hypotheses and purpose statement. The variables seek measurable and observable data. The data collection in quantitative research should use instruments with preset questions and responses and collect numeric data from a large number of participants. It uses statistical tools for analysing data, focuses on comparing group differences, and describing trends. In interpreting the results, it compares ‘results with prior predictions and past research’ (Plano-Clark & Creswell, 2010; p. 73). It uses unbiased, objective, structured criteria for reporting research findings.
On the other hand, qualitative research ‘tends to address research problems requiring a detailed understanding of a phenomenon, and an exploration because little is known about the problem’ (Plano-Clark & Creswell, 2010; p. 66). The literature review plays a minor role in articulating research questions but it helps in justifying the importance of studying the research problem. It uses general and broad research questions and purpose statements and seeks participants’ experiences in understanding them. The data collection is performed by ‘forms with general, emerging questions to permit the participants to generate responses’ (Plano-Clark & Creswell, 2010; p. 70) and by collecting textual and pictorial data from a small number of participants or sites. A text analysis technique is used for analysing and developing rich descriptions. The interpretation of results is based on personal reflections and the larger meaning of the findings. The reports in qualitative research take a ‘subjective and reflexive approach’ and use flexible structure and evaluation criteria (Plano-Clark & Creswell, 2010; p.74).

The choice of methodology determined by the philosophical underpinning on which the research is founded. The action research methodology is selected for this study to deal with the incremental nature of inquiry in the study, and to address research questions. ‘It [action research] is becoming a more accepted tool for teachers to assess their own teaching strategies and reflect upon their effectiveness’ (Schmidt, 2002; p.1). Moreover, researchers can use it to improve their practices in their discipline (Morton-Cooper, 2000). Clear (2004) mentioned that improvement may include improving the practice, improving the understanding of a practice and improving the situation of the practice where it takes place. Thota et al. (2012) discussed that action research provides iterative improvements in teaching and learning process of introductory programming (IP). They
introduced action research cycles to iteratively design, implement and evaluate an introductory object-oriented programming course using the Java programming language. Shuhidan (2012) used an action research methodology to provide better understanding of teaching and learning processes involved in fundamental programming courses. Likewise, Lyndall (2013) used action research ‘to identify and implement unit improvement initiatives over a three year period for an underperforming unit’ (p. 1). Moreover, Scott (2015) used it to improve student success in a developmental math course at a two-year college.

This chapter is structured as follows. In section 3.2, I describe constructivism learning theory which is generally used in the computer science discipline. Sections 3.3 and 3.4, provide an overview and address critiques of action research methodology respectively. In section 3.5 I provide a brief overview of a case study method. In section 3.6 I discuss the design of this research project. In section 3.7, I describe the themes and goals for each cycle of the action research. An overview and critique of paper-based surveys is described in section 3.8 and focus groups in section 3.9. In section 3.10, I describe the data analysis methods used in this research project. Section 3.11 focuses on ethical consideration for this study and the delimitations and limitations of this study are described in sections 3.12 and 3.13 respectively. Section 3.14 provides a summary of this chapter.

3.2 Learning theories

Learning theories are conceptual frameworks which explain how information is absorbed, processed and retained during learning. Yadin (2011) explained that many learning theories have been developed over the years, such as behaviourism (‘Learning is a change
3 RESEARCH METHODOLOGY AND DESIGN

in behavior’) (Merriam & Bierema, 2014, p.26), constructivism (‘Learning is creating meaning from experience’) (Merriam & Bierema, 2014, p. 36), humanism (‘Learning is about the development of the person’ (Merriam & Bierema, 2014, p.29) and cognitivism (‘Learning is a mental process’)(Merriam & Bierema, 2014, p.31). Constructivism theory is the one which is widely used in science. It is based on Piaget’s theory of child development. ‘According to Piaget, information and data are perceived and maintained using mental structures that represent knowledge’ (Yadin, 2011 p. 72). In learning processes, existing mental structures are compared with the new received information, and if the new received information makes sense it will be integrated with the existing mental structures. This integration is the cause of learning as it changes or renews the mental structure. On the other hand, if the mental structure contradicts the new received information, it will be rejected or changed to fit with the structure. ‘If students are forced to understand the new received information, but if it does not fit their mental structure, they will memorise it without proper understanding, which implies that it is not conceptualised and will not be used in future problem-solving’ (Yadin 2011; p. 72).

Biggs (2003) distinguished between two types of knowledge: declarative and procedural. Declarative knowledge deals with factual information (knowing that) and procedural knowledge (knowing how) – the skills to perform a specific task (McGill and Volet, 1997; Ten-Berge and Van-Hezewijk, 1999). Students need both types of knowledge, particularly in introductory programming. Teachers should define course outcomes which promote both types of knowledge. The teaching process is not just transmission of information from the teacher to the student (declarative knowledge) but rather it helps students to develop their mental structures (procedural knowledge).
Based on this constructivism theory, teachers should prepare course materials which include different activities which enable students to acquire the required knowledge (Yadin, 2011). Moreover, constructivism pedagogy is more appropriate to address the issues raised in the teaching and learning process of IP courses (Ben-Ari, 2001). Wang et al. (2012) reformed an IP course based on constructivism theory. They claimed that constructivism theory is better than objectivity theory which is used in most of the colleges in the teaching process. The objectivity theory views learners as passive recipients of knowledge and teachers impart their knowledge to the students. Boyer et al. (2008) also introduced constructivism pedagogy in teaching of programming to promote factual knowledge and programming skills by incorporating peer learning and authentic feedback model. Brito (2010) introduced constructivism in an IP course to promote active learning processes in students. Thevathayan and Hamilton (2015) introduced flexible and incremental visual constructivist pathways for IP students in their formative assignments. The majority of the students found it beneficial in their learning process.

In this study, constructivism pedagogy was promoted in the teaching and learning process of an IP course. A new approach (ADRI) was incorporated in the teaching and learning process to promote both declarative and procedural knowledge for introductory programming.

3.3 Defining and Designing Action Research

Action research is defined by O'Brien (2001) as ‘learning by doing - a group of people identify a problem, do something to resolve it, see how successful their efforts were, and if not satisfied, try again’ (Para. 3).
According to Carr et al. (1986), ‘Action research is simply a form of self-reflective enquiry undertaken by participants in social situations in order to improve the rationality and justice of their own practices, their understanding of these practices, and the situations in which the practices are carried out’ (p. 162).

Gill & Johnson (2002) argued that ‘By definition most action research projects are pursued through the medium of a case study … This of course raised issues about the extent to which the findings were generalizable to other cases’ (p. 79). O’Brien (2001) mentioned that action research is used to solve real world problems. In this research, action research is used to understand and improve teaching and learning practices in an IP course.

McIntosh (2010) discussed that action research is used in medical and educational areas of research, as both can improve their practices based on the feedback received from their respondents. Philips and Carr (2010) described action research as a practitioner-based form of research in education because the main aim of educators is to conduct research in order to improve student learning and pedagogy. Muir (2007) explained that practitioners apply action research in real situations, receive feedback and modify their theory or model based on it, and trying it again. Mills (2010) discussed action research as a systematic approach to student learning in the classroom.

Carr and Kemmis (1986) mentioned that:
‘Any action research study or project begins with one pattern of practices and understandings in one situation, and ends with another, in which some practices or elements of them are continuous through the improvement process while others are
discontinuous (new elements have been added, old ones have been dropped, and
transformations have occurred in still others). Similarly, understandings undergo a
process of historical transformation. And the situation in which the practices are
conducted will also have been transformed in some ways’ (p. 182).

Ferrance (2000) described how action research can be used with students and instructors
of programming to improve their teaching and learning experiences and as a result
enhance their skills, techniques and strategies. I use action research in this research
project to improve novices’ learning processes in introductory programming.

Shuhidan (2012) divided the action research cycle into following five steps:

I. Defining the issue
II. Planning the action
III. Taking the action
IV. Reflecting and refining
V. Reporting the findings (p. 48-49)

Shuhidan (2012) depicts one action research cycle as shown in figure 3.1:
According to Shuhidan (2012), in the first step of the action research cycle, the researcher makes general observations in the area of interest by looking at existing literature. In the next step, the researcher plans the steps to be taken to handle the identified research problem. In the third step, the planned steps are executed and data is collected. In the fourth step, the collected data is analysed and discussed. The last step is to write a report on the research findings. The findings of the first cycle are fed as improvements into the second cycle which also serves as a bridge between the first and second cycles. The researcher may improve their research by reflecting on positive and negative points in the first cycle. As a result, it may contribute to achieving better output and higher quality research in the second cycle.
In conclusion, action research can be a method to engage in reflective practice (Newman, 2000), and as a tool to improve practice (McNiff & Whitehead, 2010).

### 3.4 Critiques of Action Research

Elliott (1991) argued against action research suggesting that it does not generate new knowledge; rather it improves a practice. Almost 20 years later, many action researchers hold the same view. Huang (2010) explained how ‘action researchers privilege the context of practice over disembodied theory … theory without practice is not theory but speculation’ (p. 93). Sumara and Carson (1997) argued that ‘action research is a lived practice that requires the researcher not only investigate the subject at hand but, as well, provide some account of the way in which the investigation both shapes and is shaped by the researcher’ (p. xiii). McNiff and Whitehead (2010) raised their concern that action research is being used to promote professional development instead of generating theory. They argued that generally practitioners are competent enough to develop their own logical explanations for practice and may require guidance in theorizing this practice. However, Schon (1983) stated that ‘competent practitioners usually know more than they can say. They exhibit a kind of knowing-in-practice, most of which is tacit’ (p. viii).

The critique of action research discussed above provides further justification for its appropriateness as a research methodology for this study. The research questions were designed to promote understanding of practice and further enhance it while theorising about it. Instructors’ tacit knowledge of pedagogy is a key element of their practice in lecture and laboratory contexts. The action research cycles in this study helped in uncovering this tacit knowledge among the teacher staff and making it more explicit.
3.5 Case Study

Most action research projects are based on case studies (Gill & Johnson, 2002). In this study, I follow the same approach and confine the research to the IP course offered at Buraimi University College. The case study involves an in-depth examination of an individual person, group, institution, or event. ‘The goal of the case study is to provide an accurate and complete description of the case’ (Marczyk et al., 2005; p. 147). The case study can comprise single or multiple cases. Each case should be bounded by place, time and/or physical boundaries (Plano-Clark & Creswell, 2010). ‘One major feature of case study methodology is that a range of data collection methods are combined to gather information with the purpose of illuminating a case from different angles’ (Johansson, 2003; p.3). Different data collection methods (qualitative, quantitative or both) can be used in the case study (Johansson, 2003).

The collected data through the case study is analysed to provide a rich description of the case and to develop themes or patterns of information about the case (Plano-Clark & Creswell, 2010). Case studies provide subjective rather than objective information. The results cannot be generalised to a wider population (Singh, 2006).

3.6 Research Design

A research design defines a logical structure of the inquiry (De-Vaus, 2001). For this study, the research design is based on the three entities (student, instructor and content) of the didactic triangle. As explained in section 2.3, these three entities are important in teaching and learning environments. The student is taught by the instructor, who guides the student through the contents. The contents or teaching materials are available for both
student and instructor. The interactions or relationships between these three entities are important to facilitate the learning process (see figure 2.1).

I now give a brief description of these three entities (novices (as student), (IP) instructor, content (IP course)) used in the study context.

3.6.1 Novices
In this research, the term novice means any individual who is enrolled in an IP course to learn basic concepts of programming with little or no programming background. The IP course known as ‘Introduction to algorithm and programming 1’ is a core course for all degrees offered in the Department of Information Technology in Buraimi University College, Oman. There is no pre-requisite for the IP course. The course covers basic topics and concepts of programming by using the C++ language. The course is important for novices to pursue their career in the programming field. Furthermore, it is pre-requisite for some other modules in their degree.

The IP course equips novices with the knowledge and skills of programming knowledge (syntax and semantics) and problem solving strategies. Our aim is to help novices in achieving their objectives (course learning outcomes) by introducing the ADRI based teaching materials which emphasis on both programming knowledge and problem solving strategies equally. In this thesis novices may also be referred to as students or learners.

3.6.2 Instructor
The term instructor is defined as any individual who is lecturing or teaching an IP course. They must have necessary qualifications and experience in programming courses at
tertiary education level. In our context, I use the term instructor for anyone who is involved in preparing and teaching the IP course materials and assessment tools.

### 3.6.3 Details of introductory programming course

The IP course is level one compulsory course for all degrees offered in the Department of Information Technology in Buraimi University College. It covers basic concepts of programming including program analysis and specification, development and design using algorithm such as pseudo code and flowchart, introduction to C++, operators and expressions, control structures, functions and arrays.

The assessment criteria consists of Test 1 (20%), Test 2 (20%), Assignment (10%), and Final Exam (50%).

One hour is allocated for each of the tests. The time allocated for the final exam is two hours. Instructors use different types of questions in the exam. Normally it consists of multiple choice questions, evaluating output from a given piece of code, writing a program for a given problem, writing algorithm (flow chart), converting mathematical expression to C++ expressions, and correcting errors in a given code.

Buraimi University College uses a traditional approach to offering an introductory programming course. Mascolo (2009) explained that ‘traditional teacher-centered pedagogy is generally defined as a style in which the teacher assumes primary responsibility for the communication of knowledge to students…. Teacher-centered pedagogy is often described as being based upon a model of an active teacher and a passive student (p.4)’. In this research project I introduced the ADRI approach which promotes guided participation. ‘The concepts of guided participation proceed from the
idea that learning is neither a teacher-centered nor a student-centered process’ (Mascolo, 2009; p. 11). Teachers play an important role in the communication of programming concepts and students are actively involved in practicing and refining these concepts. On the other hand, the teaching materials based on the traditional approach promote programming shortcuts where novice programmers attempt to write code directly to solve the problem statement (Problem Statement $\rightarrow$ Code) as shown in figure 1.1.

### 3.6.4 Context of learning

The context of learning for this research is Buraimi University College, Oman. The college is affiliated with the California State University, Northridge, USA. Although English is a second language for students, the medium of instruction is English. Most of the students enrolled in the IP course, have to complete a one year foundation program which focuses on their English language skills. Learning a programming language is challenging for novices. Learning it in a foreign language adds to the complexity and challenge.

### 3.7 Action Research themes and goals

In this research, I proposed three cycles of the action research, as shown below in figure 3.2, to achieve my goals. Each cycle is further elaborated by using the five steps discussed above in section 3.3. Each cycle used a new group of respondents. For example, in action research cycle 1, I analysed responses of novice programmers to find their views regarding IP topics from Semester 1, 2013-14. In cycle 2, I introduced ADRI based IP teaching materials and assessments for Semester 1, 2014-15 students. In cycle 3, I introduced improved ADRI based IP teaching materials to semester 2, 2014-15 students.
Likewise the instructors of the IP course who offered this course in a particular semester were involved in the relevant cycles. For example, in cycle 2, instructors who taught the course in semester 1, 2014-15 were involved in a focus group.

Figure 3.2 depicts the stages, participants and activities performed in the three cycles of the action research used in this study. I elaborate on these cycles, the participants and activities in the next section.

Table 3.1 shows the themes and goals for all three cycles involved in this study. In cycle 1, a survey was conducted with the students (control group) to obtain their feedback regarding the IP course. The teaching materials and assessment tools of the IP course based on the traditional approach were analysed.
### Table 3.1: Themes and goals of three cycles of action research

<table>
<thead>
<tr>
<th>Cycle No.</th>
<th>Theme</th>
<th>Participants</th>
<th>Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Novice programmers</td>
<td>Semester 1 &amp; 2, 2013-14</td>
<td>To survey novice programmers to find their views regarding IP topics</td>
</tr>
<tr>
<td></td>
<td>Teaching materials and assessment tools</td>
<td>Academic Year 2010-13</td>
<td>To analyse IP teaching materials and assessment tools</td>
</tr>
<tr>
<td>2</td>
<td>ADRI based IP teaching materials</td>
<td>Semester 1, 2014-15</td>
<td>To prepare and introduce ADRI based IP teaching materials and assessment to novices</td>
</tr>
<tr>
<td></td>
<td>Novice programmers</td>
<td>Semester 1, 2014-15</td>
<td>To survey novice programmers to find their views regarding ADRI based IP teaching materials and assessment tools.</td>
</tr>
<tr>
<td></td>
<td>IP Instructors</td>
<td>Semester 1, 2014-15</td>
<td>To conduct a focus group with IP instructors to find their view regarding ADRI based IP teaching materials and assessment tools.</td>
</tr>
<tr>
<td>3</td>
<td>Improved ADRI based IP teaching materials</td>
<td>Semester 2, 2014-15</td>
<td>To introduce improved ADRI based IP teaching materials and assessment tools to novices.</td>
</tr>
<tr>
<td></td>
<td>Analysis of ADRI based teaching materials</td>
<td>Semester 2, 2014-15</td>
<td>To analyse ADRI based teaching materials against five categories</td>
</tr>
<tr>
<td></td>
<td>IP Final grades</td>
<td>Semester 1 &amp; 2, 2013-14 and Semester 1 &amp; 2, 2014-15</td>
<td>To analyse IP final grades for the last four semesters</td>
</tr>
<tr>
<td></td>
<td>Comparison of traditional and ADRI based teaching materials</td>
<td>Semester 2, 2013-14 and Semester 2, 2014-15</td>
<td>To compare IP teaching materials based on traditional and ADRI approaches</td>
</tr>
</tbody>
</table>
In cycle 2, the course teaching materials were prepared based on the new approach (ADRI) and offered to the students of semester 1, 2014-15. A survey was conducted with the students (treatment group) to obtain their feedback regarding the ADRI approach. A focus group was conducted with the IP instructors to get their in-depth feedback regarding the ADRI approach. In the cycle 3, the ADRI approach was improved based on the feedback received from the cycle 2 and then incorporated into the course. The updated course was offered to the students of semester 2, 2014-15. The teaching materials based on the ADRI approach were analysed and then compared with the teaching materials based on the traditional approach. Finally, the final grades of the students in these four semesters were analysed and compared.

3.7.1 Action Research: Cycle 1

In cycle 1, I focused on identifying the learning difficulties faced by novices in the IP course. There were two activities performed in this cycle as shown above in figure 3.2. Firstly, the survey was conducted with the students. Secondly, the teaching materials and assessment tools (final exams only) of the IP course used at Buraimi University College were analysed. I outline the steps of this cycle in tables 3.2 and 3.3.

In step one, I focused on novices to determine the difficulties they encounter in learning to program, as discussed in chapter 2. Earlier, I discussed that high failure rates or dropout from IP courses is a big challenge for computer science departments. A paper-based survey was developed (step two) and deployed (step 3) to students in the IP course in order to gain an understanding of their learning difficulties. This group is referred to as the control group. Step 2, Planning action, is discussed in section 4.3.2 and the third step, is discussed in section 4.3.1. In the fourth step, I reflected on the data collected from the
first survey with novices. After analysing the collected data using statistical tools, I identified novices’ nomination of the top three most difficult IP concepts and topics, and their top three favourite learning situations and resources. Reflecting on the data and refining my understanding is discussed in section 4.4.1. In the fifth step, I elaborated the results of the first survey in the context of previously conducted studies in the same area of interest. The findings are reported and discussed in section 4.5.1.

<table>
<thead>
<tr>
<th>Step Number</th>
<th>Step Name</th>
<th>Descriptions</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.</td>
<td>Defining the issue</td>
<td>Novices’ preferences on IP concepts, topics, learning situations and resources.</td>
<td>Chapter 2</td>
</tr>
<tr>
<td>ii.</td>
<td>Planning action</td>
<td>Preparation of paper based survey covering IP concepts, topics, learning situations and resources.</td>
<td>Section 4.3.2</td>
</tr>
<tr>
<td>iii.</td>
<td>Taking action</td>
<td>A first survey conducted with novices.</td>
<td>Section 4.3.1</td>
</tr>
<tr>
<td>iv.</td>
<td>Reflecting and refining</td>
<td>Reflect on the data collected from the first survey</td>
<td>Section 4.4.1</td>
</tr>
<tr>
<td>v.</td>
<td>Report findings</td>
<td>Detail findings</td>
<td>Section 4.5.1</td>
</tr>
</tbody>
</table>

In this cycle, I also analysed the current IP course materials and the final examination used at Buraimi University College. The research approach is outlined in table 3.3.

The current teaching materials of the IP course were analysed. Moreover, assessment tools (final exams without students’ answers) for the course for the last three years (2010 to 2013) were also analysed.
3 RESEARCH METHODOLOGY AND DESIGN

Table 3.3: analysis of introductory programming course materials

<table>
<thead>
<tr>
<th>Step Number</th>
<th>Step Name</th>
<th>Descriptions</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.</td>
<td>Defining the issue</td>
<td>Analysis of current IP course materials and assessment tools</td>
<td>section 2.3</td>
</tr>
<tr>
<td>ii.</td>
<td>Planning action</td>
<td>Teaching materials are analysed for types of programming examples/problems, presentation styles, and problem solving steps. Assessment tools are analysed for different types of questions e.g. coding, problem solving, and evaluating output from a given piece of code.</td>
<td>Section 4.3.2</td>
</tr>
<tr>
<td>iii.</td>
<td>Taking action</td>
<td>IP course materials and assessment tools are analysed.</td>
<td>Section 4.3.1</td>
</tr>
<tr>
<td>iv.</td>
<td>Reflecting and refining</td>
<td>Reflection on the data collected from the current IP course materials and assessment tools</td>
<td>Sections 4.4.2 &amp; 4.4.3</td>
</tr>
<tr>
<td>v.</td>
<td>Report findings</td>
<td>Detail findings</td>
<td>Section 4.5.2</td>
</tr>
</tbody>
</table>

The teaching materials were analysed according to four categories - different course parameters e.g. teaching strategies, assessment methods, inclusion of problem solving strategies, lab exams, use of multimedia or visual aids etc., types of Problems and Examples, Presentation Styles, and Four Steps of Problem Solving. This analysis is described in section 4.4.2.

The final examination for the programming course was analysed against five different categories - coding questions, problem solving strategies questions, evaluating output questions from a given piece of code, explanation and definition questions, and miscellaneous questions. This analysis is described in section 4.4.3.
3 RESEARCH METHODOLOGY AND DESIGN

3.7.2 Action Research: Cycle 2

In this cycle, I discuss the steps to prepare the ADRI based teaching materials based on the feedback received from cycle 1. Secondly, a second survey (treatment group) was conducted with novices to obtain their feedback on the ADRI based teaching materials. Thirdly, a focus group was conducted with the IP instructors to get their feedback on the ADRI based teaching approach and materials. I outline the steps of this cycle in tables 3.4, 3.5 and 3.6.

Table 3.4 depicts the steps involved in preparation of the ADRI based teaching materials.

<table>
<thead>
<tr>
<th>Step Number</th>
<th>Step Name</th>
<th>Descriptions</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.</td>
<td>Defining the issue</td>
<td>To prepare ADRI based teaching materials for IP course</td>
<td>Section 4.5</td>
</tr>
<tr>
<td>ii.</td>
<td>Planning action</td>
<td>ADRI based lecture notes, exercises questions, and editor are prepared</td>
<td>Section 5.3</td>
</tr>
<tr>
<td>iii.</td>
<td>Taking action</td>
<td>ADRI based materials are prepared and introduced to novices (of semester 1, 2014-15) in IP course</td>
<td>Section 5.4</td>
</tr>
<tr>
<td>iv.</td>
<td>Reflecting and refining</td>
<td>Reflect on the ADRI based materials based on the feedback received from IP students and instructors</td>
<td>Section 5.5</td>
</tr>
<tr>
<td>v.</td>
<td>Report findings</td>
<td>Detail findings</td>
<td>Section 5.6</td>
</tr>
</tbody>
</table>

The teaching materials based on the ADRI approach were prepared for the IP course. The programming examples in lecture notes and exercises questions in lab sheets were prepared according to the four stages of the ADRI approach which are Approach, Deployment, Result and Improvement (see section 5.3.1).
3 RESEARCH METHODOLOGY AND DESIGN

A simple editor was prepared to facilitate the learning process of novices. The editor was specifically designed to promote the ADRI approach in student learning (see section 5.3.2).

A survey is deployed to students (treatment group) to explore their perceptions of the new learning paradigm. I outline the steps of the second survey in table 3.5.

*Table 3.5: second survey with novices*

<table>
<thead>
<tr>
<th>Step Number</th>
<th>Step Name</th>
<th>Descriptions</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.</td>
<td>Defining the issue</td>
<td>Feedback on ADRI based teaching materials from novices.</td>
<td>Section 5.1</td>
</tr>
<tr>
<td>ii.</td>
<td>Planning action</td>
<td>A second survey is prepared to get feedback on ADRI based approach from novices (of semester 1, 2014-15).</td>
<td>Section 5.5</td>
</tr>
<tr>
<td>iii.</td>
<td>Taking action</td>
<td>A second survey is conducted with novices.</td>
<td>Section 5.5.1</td>
</tr>
<tr>
<td>iv.</td>
<td>Reflecting and refining</td>
<td>Reflect on the data collected from second survey and focus group</td>
<td>Section 5.5.1</td>
</tr>
<tr>
<td>v.</td>
<td>Report findings</td>
<td>Detail findings</td>
<td>Section 5.6.1</td>
</tr>
</tbody>
</table>

The questions in the second survey were the same as in the first survey which helped to determine the impact of the ADRI approach on novices (see section 5.5).

Within the second cycle, a focus group was conducted with the IP instructors to get their in-depth feedback on the ADRI approach as shown in table 3.6. The main objectives of the focus group were to explore instructors’ perceptions of students’ experience with the ADRI approach, and the impact the ADRI approach had on the outcomes of students’ learning (see section 5.5.2).
3 RESEARCH METHODOLOGY AND DESIGN

Table 3.6: focus group with introductory programming instructors

<table>
<thead>
<tr>
<th>Step Number</th>
<th>Step Name</th>
<th>Descriptions</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.</td>
<td>Defining the issue</td>
<td>Feedback on ADRI based teaching materials from IP instructors through focus group.</td>
<td>Section 5.1</td>
</tr>
<tr>
<td>ii.</td>
<td>Planning action</td>
<td>The focus group is organised with IP instructors (of semester 1, 2014-15) to get their feedback on ADRI based approach.</td>
<td>Section 5.5</td>
</tr>
<tr>
<td>iii.</td>
<td>Taking action</td>
<td>The focus group is conducted with IP instructors.</td>
<td>Section 5.5.2.2</td>
</tr>
<tr>
<td>iv.</td>
<td>Reflecting and refining</td>
<td>Reflect on the data collected from the focus group</td>
<td>Section 5.5.2</td>
</tr>
<tr>
<td>v.</td>
<td>Report findings</td>
<td>Detail findings</td>
<td>Section 5.6.2</td>
</tr>
</tbody>
</table>

3.7.3 Action Research: Cycle 3

In this cycle, improvements were made to the teaching materials based on the findings from cycle 2 and then deployed to students in semester 2, 2014-15. Secondly, the teaching materials based on the ADRI approach were analysed. Thirdly, the final grades of the IP course for the four semesters (semester 1 & 2, 2013-14 and semester 1 & 2, 2014-15) were analysed. Lastly, the IP teaching materials based on the traditional and ADRI approaches were compared. I outline the steps of this cycle in tables 3.7, 3.8, 3.9 and 3.10 below.

The teaching materials based on the ADRI approach were updated based on the findings from the cycle 2. The steps are outlined in table 3.7.
The ADRI approach was improved based on the feedback received from cycle 2. The feedback was incorporated into the ADRI approach which was offered to the students in semester 2, 2014-15 (see section 6.2).

Secondly, in this cycle, the teaching materials based on the ADRI approach were analysed. The purpose was to determine the impact of the ADRI approach on the teaching materials. The steps are outlined in table 3.8.

<table>
<thead>
<tr>
<th>Step Number</th>
<th>Step Name</th>
<th>Descriptions</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.</td>
<td>Defining the issue</td>
<td>Improving ADRI based teaching materials</td>
<td>Section 5.6</td>
</tr>
<tr>
<td>ii.</td>
<td>Planning action</td>
<td>Analyse feedback received from cycle 2 to improve ADRI based teaching materials</td>
<td>Section 6.2</td>
</tr>
<tr>
<td>iii.</td>
<td>Taking action</td>
<td>To improve the ADRI based teaching materials based on recommendations</td>
<td>Sections 6.2.1, 6.2.2 &amp; 6.2.3</td>
</tr>
<tr>
<td>iv.</td>
<td>Reflecting and refining</td>
<td>Evaluate the feedback on the ADRI based teaching materials</td>
<td>Sections 6.6</td>
</tr>
<tr>
<td>v.</td>
<td>Report findings</td>
<td>Detail findings</td>
<td>Chapter 6</td>
</tr>
</tbody>
</table>

Table 3.7: improved ADRI based teaching materials

Table 3.8: analysis of the ADRI based teaching materials
The teaching materials based on the ADRI approach were analysed against the five categories—Teaching topics for lecture sessions, Practical topics for lab sessions, Types of programming examples and problems, Presentation style of example and problems, and Four problem solving steps in examples and problems (see section 6.3).

Thirdly, in this cycle, the final grades of the IP course for the four semesters were analysed to determine the impact of the ADRI approach on novices’ performances. The steps are outlined in table 3.9.

<table>
<thead>
<tr>
<th>Step Number</th>
<th>Step Name</th>
<th>Descriptions</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.</td>
<td>Defining the issue</td>
<td>Analysis of final grades of IP course to determine the impact of ADRI approach on novices’ performances</td>
<td>Section 6.1</td>
</tr>
<tr>
<td>ii.</td>
<td>Planning action</td>
<td>IP course final grades of last four semesters (semester 1 &amp; 2, 2013-14 and semester 1 &amp; 2, 2014-15) are collected from registration department.</td>
<td>Section 6.4</td>
</tr>
<tr>
<td>iii.</td>
<td>Taking action</td>
<td>Analysis of final grades of four semesters to determine the trends in failure and dropout rates, and the impacts on final grades.</td>
<td>Section 6.4.1</td>
</tr>
<tr>
<td>iv.</td>
<td>Reflecting and refining</td>
<td>Evaluation of the impact of ADRI approach on novices’ performances</td>
<td>Section 6.6.4</td>
</tr>
<tr>
<td>v.</td>
<td>Report findings</td>
<td>Detail findings</td>
<td>Chapter 6</td>
</tr>
</tbody>
</table>

Finally, in this cycle, the course materials of the IP course based on the traditional and ADRI approaches were compared to determine the impact of the ADRI approach on the course. The steps are outlined in table 3.10.
3 RESEARCH METHODOLOGY AND DESIGN

Table 3.10: comparison of the traditional and ADRI based teaching materials

<table>
<thead>
<tr>
<th>Step Number</th>
<th>Step Name</th>
<th>Descriptions</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.</td>
<td>Defining the issue</td>
<td>Comparison of the IP teaching materials based on the traditional and ADRI approaches to determine the impact of the ADRI approach on the course</td>
<td>Section 6.1</td>
</tr>
<tr>
<td>ii.</td>
<td>Planning action</td>
<td>The teaching materials based on the traditional and ADRI approaches are analysed against five categories</td>
<td>Section 6.5</td>
</tr>
<tr>
<td>iii.</td>
<td>Taking action</td>
<td>Compare the traditional and ADRI based teaching materials based on five different categories listed below</td>
<td>Section 6.5</td>
</tr>
<tr>
<td>iv.</td>
<td>Reflecting and refining</td>
<td>Evaluate the impact of ADRI approach on the course</td>
<td>Section 6.6.5</td>
</tr>
<tr>
<td>v.</td>
<td>Report findings</td>
<td>Detail findings</td>
<td>Chapter 6</td>
</tr>
</tbody>
</table>

The teaching materials based on the traditional and ADRI approaches were compared in the five different categories - Teaching topics for lecture sessions, Practical topics for lab sessions, Types of programming examples and problems, Presentation style of example and problems, and Four problem solving steps in examples and problems (see section 6.6.5).

3.8 Paper-based Survey

Two surveys of students were used in this research, in cycles 1 and 2 as data collection methods (see appendices B and C). The survey is a cost-effective and timely method to collect data from a targeted population. Questions in the survey were standardised and ensured that similar data are collected from respondents. Closed response questions in the survey collect precise responses to measure. There is a decreased potential of errors while
inputting data in a self-administered survey (Anderson & Kanuka, 2003). In the survey, there is no direct connection between the researcher and respondents, therefore, interviewer biases are minimised (Walonick, 2010).

A paper-based survey method was selected instead of online based on research evidence that this would increase the response rate. Nulty (2008) compared response rates of eight different research based studies which were conducted by online and paper-based surveys. He concluded that response rates in paper-based surveys were much higher than online surveys.

### 3.8.1 Addressing critiques of surveys

Question focus and design are amongst the issues raised by critiques of surveys (Merriam & Simpson, 2000; Walonick, 2010). If the phrasing of questions is not clear, respondents may misinterpret them or may not give response at all. To address this, most of the questions included in both surveys were taken from an existing survey developed by Lahtinen et al. (2005) and then the surveys were piloted with several individuals: my doctoral principal supervisor, a colleague who has experience in survey question design, and the IP instructors at my department in Oman. Questions and response measures were revised based on their feedback.

Response bias and authenticity were mitigated by taking the following steps: both surveys were conducted anonymously which helps to address the issue of respondents who do not want to provide true details to potentially contentious questions. Both surveys and related documents (students’ invitation letter, the Plain language statement and consent form) were translated into Arabic (the respondents’ first language) to enhance their understanding of questions (see appendices B and C). One stage cluster sampling was
used to include all the students of the IP course in each of the four semesters to provide them equal opportunity for gaining benefits from the research. ‘In the field of education research cluster sampling techniques are most frequently used and although they have some limitations they have usability in teaching learning situations and educational research’ (Singh, 2006; p. 98). Submissions were confidential because respondents could return completed surveys at their own time to the researchers drop box.

3.9 Focus Group

A focus group was used in cycle 2 of this research as a method of collecting data from teachers (see Appendix D). Stewart & Shamdasani (2014) define a focus group as ‘a group of individuals who discuss a particular topic under the direction of a moderator who promotes interaction and ensures that the discussion remains on the topic of interest’ (p. 40). A focus group provides opportunity for participants to be involved in decision making processes and be treated as experts (Race et al., 1994). It tends to generate rich information on given topics as participants share and reflect on ideas and experiences of each other (OMNI, 2012). If a group works well and trust is developed among participants then focus groups may explore solutions to a particular problem as a unit rather than as individuals (Kitzinger, 1995). The collected data from a focus group is a representation of diverse opinions and ideas of participants. Focus groups are also considered relatively low cost and a quick way to generate a great deal of information (OMNI, 2012). They provide an opportunity to ask follow-up questions to probe deeper and to gain insights on given topics from the participants (Lauer, 1996).
3 RESEARCH METHODOLOGY AND DESIGN

3.9.1 Addressing critiques of focus groups
The individuals in a focus group are expressing their views in a particular context and environment ‘so sometimes it may be difficult for the researcher to clearly identify individual message’ (Gibbs, 1997; Para. 15). The validity and reliability of focus group findings can be undermined due to moderator bias (OMNI, 2012). To address this, a moderator and an assistant moderator took notes of the focus group session independently and anonymously. They compared their notes after the session and produced a session report. The report was sent to the participants for their feedback.

3.10 Methods of Data Analysis
The statistical software package, IBM SPSS (Statistical Package for Social Science) version 20, was used for data analysis in this study. Leedy et al. (2004) stated that ‘with statistics, we can summarize large bodies of data, make predictions about the future trends and determine when different experimental treatments have led to significantly different outcomes. Thus, statistics are among the most powerful tools in the researcher’s toolbox’ (p. 217).

SPSS is a comprehensive system for analysing collected data(Norusis 2006) and accepts data from different types of files and generates tabulated reports, charts, plots, descriptive statistics and complex statistical analyses (Shuhidan 2012)

The following descriptive statistics methods were used to analyse the collected data.

Mean of data: Levine and Stephan (2005) define the concept of mean as ‘A number equal to the sum of the data values for a variable, divided by the number of data values that were
summed’ (p. 37). This method is used in this study because it represents a single number for the whole data set of scores (Shuhidan, 2012).

Frequency of data: ‘A frequency distribution is a listing of frequencies of all categories of the observed values of a variable’ (Freund et al., 2010; p.13). In frequency distribution, the data is grouped into categories and then the number of observations that fall into each category is counted (Freund and Wilson, 2003).

Mann-Whitney U test: A non-parametric test which is used to compare the difference between two independent groups. It is normally used with ordinal data. The test does not assume normally distributed values. It is based on independence of observations. Hole (2013) described how the Mann-Whitney test processes the data as follows:

‘The logic behind the Mann-Whitney test is to rank the data for each condition, and then see how different the two total ranks are. If there is a systematic difference between the two conditions, then most of the high ranks will belong to one condition and most of the low ranks will belong to the other one. As a result, the rank totals will be quite different. On the other hand, if the two conditions are similar, then high and low ranks will be distributed fairly evenly between the two conditions and the rank totals will be fairly similar.’ (p.1)

Hypothesis testing was performed on the results obtained from the Mann-Whitney U test to determine its statistical significance in the context of this study. Sullivan (2009) defines hypothesis testing as ‘a procedure, based on sample evidence and probability, used to test statements regarding a characteristic of one or more populations’ (p. 455). Statements can be either true or false; therefore we write and test generally two hypothesis statements: null hypothesis and alternative hypothesis. ‘The null hypothesis is a statement about the
values of one or more parameters. This hypothesis represents the status quo and is usually not rejected unless the sample results strongly imply that it is false’ (Freund and Wilson; p. 120). On the other hand, ‘the alternative hypothesis is a statement that contradicts the null hypothesis. This hypothesis is declared to be accepted if the null hypothesis is rejected. The alternative hypothesis is often called the research hypothesis’ (Freund & Wilson; p. 120).

3.11 Ethical Considerations

Ethical considerations are important in conducting social research (Maen Al-Hawari, 2004). In social research, the object of inquiry is human beings so extreme care should be taken to avoid any harm to them (Fontana and Frey 1998). According to VanManen (1990) this includes psychological harms such as stress, emotional distress and self-doubt.

In this study, IP students and instructors were involved in data collection. So it was important to take care of their privacy, consent and confidentiality.

To address the ethical issues in this study, approval was sought and given by Deakin University’s Human Ethics Advisory Group (HEAG) before data collection commenced (see appendix A). As discussed earlier, there were two research participant groups, students and instructors. From students, the data was collected through anonymous surveys. From instructors, the data was collected after de-identification process from existing exam papers (without students’ answers) and teaching materials. The de-identification process was carried out by an independent party. An individual consent was obtained from IP course instructors who had taught this course from 2010 till 2013. For students’ grades, the identity was de-identified by an independent party (Registration
3 RESEARCH METHODOLOGY AND DESIGN

Department at Buraimi University College) to maintain anonymity. A written consent was obtained from the Registration Department. The focus group was not recorded and participants’ responses were anonymous in the report.

3.12 Delimitations

Specific constraints were placed on the design of the study which may affect the external validity of the research findings to other settings. The purpose of applying delimitations in this study is to maintain consistency in the type of data gathered. All the three entities (student, instructor and content) of the didactic triangle were involved from one University College in Oman (Buraimi University College). More specifically, instructors were sought out who had taught the IP course at the Buraimi University College.

Two different groups of students were involved in this study as shown below in figure 3.3. The first (control) group consists of students who completed the IP course during semesters 1 & 2, 2013-14 with the traditional approach to teaching. The second (treatment) group consists of students who completed the course in semesters 1 & 2, 2014-15 with the ADRI approach.
3.13 Limitations

Recognised limitations in the study’s design may affect the credibility and validity of results (Guba, 1981). This study involved quantitative and qualitative methods (mixed methods approach) to capture real life and non-controllable variables. One way to address the limitations in action research is to triangulate data in a mixed methods approach.

In this study, the limitations listed below provide a critical analysis of the contributions the findings make to the field of teaching and learning an introductory programming course.

- Instructors who taught an IP course at Buraimi University College (context of current study) during specific academic semesters were recruited.
- As per the nature of mixed methods design (quantitative and qualitative research), the focus was only on the population of one specific location. Findings may not be
3 RESEARCH METHODOLOGY AND DESIGN

generalised to populations of other locations. However, the core concepts of introducing an ADRI approach in the teaching and learning process of an IP course are generalisable to a broader context.

- Participants interpret and react to particular research questions in action research which limits the scope and applicability to other contexts.
- Student-instructor relationships may have impacted on individuals’ response in the surveys. To address this, respondents were not asked about their identity in surveys and submissions were confidential because respondents could return completed surveys at their own time to researcher drop box.

However, despite these limitations, by following sound practices in the research, the findings and conclusions may serve to inform the field of IP education.

3.14 Conclusion

Research methodology provides a systematic way to solve research problems. For this study, the action research methodology was selected as the research paradigm. Action research provides iterative improvements which enabled me to get feedback from the IP instructors and students regarding the ADRI approach. The feedback was incorporated into the next cycle of the learning process. Three cycles of research were included in this study. The second cycle obtained feedback from the first cycle and gave feedback to the third cycle.

The data was collected through surveys, a focus group and analysis of current teaching materials. The student control group provided feedback through the first survey on current teaching and learning approaches used in the IP course. The treatment group was
3 RESEARCH METHODOLOGY AND DESIGN

introduced with the ADRI based teaching materials and then their feedback was collected through second survey regarding the new intervention. A focus group was conducted with the IP instructors regarding the ADRI approach. The collected data from surveys, focus group, analysis of students’ final grades in the four semesters, and a comparison of teaching materials based on the traditional and ADRI approaches helped determine the impact of the ADRI approach on the learning process.

The collected data was analysed by using IBM SPSS software, using statistical methods such as mean, frequency and Mann-Whitney U test.
4  Analysis of Current Teaching and Assessment materials of Introductory Programming Course¹

4.1 Introduction

This chapter describes cycle 1 of the action research used in the research. There were two activities performed in this cycle as shown below in Figure 4.1. Firstly, the first survey was conducted with the students (control group) of the introductory programming (IP) course who finished the course with the traditional approach. Secondly, the current IP course materials based on the traditional approach was analysed. Figure 4.1 below shows the three cycles of the action research and cycle 1 is further highlighted in it.

As discussed earlier in chapter 2, Tavares et al. (2001) identified curriculum organisation and teaching methods as the two main factors for high failure rates in IP courses. Meisalo et al. (2002) pointed out that 30% of their programming course students dropped out because they found the programming exercises too hard and difficult. In this chapter, I involve the three entities of the didactic triangle as described by Kansanen, (1999), instructors, students and curriculum, to explore the learning difficulties that students encounter when studying introductory programming. I first explore the perceptions of students of the barriers and affordances to learning programming. The first survey is conducted with students completing an IP course to obtain their feedback on their

¹The work reported in this chapter was published in ‘Iqbal, Sohail and Coldwell-Neilson, Jo 2016, A model for teaching an introductory programming course using ADRI, Education and Information Technologies, Springer, DOI 10.1007/s10639-016-9474-0
learning achieved during the course. Instructors’ perceptions are captured by analysing current teaching materials and assessment tools used in the course.

Cycle 1 proposes three research questions to address the issue of learning difficulties in introductory programming. First, I explored the issue by looking at the learning and teaching difficulties faced by novices in introductory programming. Second, I analysed the current teaching and assessment materials of introductory programming. Then I proposed the ADRI model for the teaching methodology to address the identified problems in learning how to program.

The research question being addressed in this chapter is:
RQ1. What are the perceptions of students of the barriers and affordances to learning programming?

The following sub-questions are investigated to inform the interpretation of the results of this cycle and to lay the foundation for exploring research questions 2, 3 and 4.

SQ1. What are the current practices in the introductory programming course assessment tools and teaching materials regarding programming knowledge and problem solving strategies?

SQ2. Is an ADRI based approach to teaching suitable to address the recommendations that emerge from research questions RQ1 and SQ1?

RQ1 is explored in section 4.4.1. SQ1 and SQ2 are investigated in sections 4.4.2 and 4.5 respectively.

This chapter firstly introduces the ADRI model. The methodology for Cycle 1 is described in section 4.3 and the results are reported and discussed in section 4.4. The chapter concludes with a summary of the outcomes.

4.2 The ADRI Model

The ADRI (Approach, Deployment, Result, and Improvement) model is an analytical tool which is a well-known quality assurance model for self-review and external review and is used extensively in the education and business sectors (Razvi et al., 2012). Australian and New Zealand universities use the ADRI model for quality audit processes. Furthermore, the Australian Business Excellence Framework and New Zealand Business Excellence Foundation have used the ADRI model to evaluate quality in their Business Excellence Awards (Carroll and Razvi, 2006). The ADRI model was developed from the Plan-Do-
4 ANALYSIS OF CURRENT TEACHING AND ASSESSMENT MATERIALS OF INTRODUCTORY PROGRAMMING COURSE

Check-Act (PDCA) model (Pietrzak and Paliszkiewicz; 2015) developed by Walter Shewhart which was subsequently modified by Deming into the Plan-Do-Study-Act (PDSA) model after realizing that ‘Check’ implied a halting process (Moen and Norman, 2010). Gazza (2015) used the PDSA cycle to improve a new online health policy course. Oliver (2010) discussed that a 360-degree evidence-based approach for curriculum enhancement is based on the ADRI model underpinning the quality audit process used by the Australian Universities Quality Agency (AUQA).

Jantti (2002) explained the four elements of ADRI (Figure 4.2) as follows:

1. **Approach** - Thinking and planning
2. **Deployment** - Implementing and doing
3. **Results** - Monitoring and evaluating
4. **Improvement** - Learning and adapting

*Figure 4.2: Four Stages of the ADRI Model (Jantti, 2002; Iqbal and Harsh, 2013)*
The first stage of the ADRI model is *approach* which consists of thinking and planning about tasks. It furnishes the development of goals, strategies, objectives, outcomes, plans and targets. Planning should clearly identify qualitative and quantitative goals and should indicate steps to achieve these goals (McGregor, 2003; Woodhouse, 2003; Abuid, 2010).

The second stage is *deployment* which provides a platform to execute or implement tasks. It is important that a clear understanding, including steps to achieve goals, should be in place. Therefore it is evident that proper planning should be ensured at the first stage to achieve targets or goals (McGregor, 2003; Woodhouse, 2003; Abuid, 2010).

The third stage is *result*, which refers to the output or findings as consequences of the first and second stages. Furthermore, it explains the process used to solve the problem statement to novices. In other words, the *result* stage refers to what is actually achieved. Here the most important point is that there should be a link between output (*result* stage) and goal (*approach* stage). The *result* stage provides a comparison between the intended output and achieved output. Therefore, it also gives an opportunity to analyse output and draw conclusions (McGregor, 2003; Woodhouse, 2003; Abuid, 2010).

The fourth stage is *improvement* which refers to the conclusions drawn from the *results* and *analysis* phases. This stage identifies what needs to be improved. As mentioned above, ADRI is a continuous cycle of improvement; therefore how to *improve* is left to the next implementation of the ADRI cycle (McGregor, 2003; Woodhouse, 2003; Abuid, 2010).
The four stages of the ADRI model also cover some of the principles, presented by Schneider (1978) for the IP course, listed in section 2.3.2.

The first stage, ‘Approach’, emphasises principles 1, 2, and 3. This stage is mainly concerned with understanding the problem statement and proposing and verifying a general solution and algorithm for it.

The second stage ‘Deployment’ covers principles 5, 7 and 8. It focuses on syntax and semantics of programming languages. Furthermore, testing and debugging of computer programs will also be handled at this stage.

The third (Result) and fourth (Improvement) stages go beyond these principles. The Result stage focuses on whether the problem statement goals are achieved or not. The Improvement stage emphasizes using different constructs to solve the same problem statement so that novices will get more practice.

### 4.3 Methodology for Cycle 1

Different methods were used to explore the research questions for this cycle. RQ1 is explored by executing an anonymous survey with IP students regarding IP topics and other related factors. SQ1 is probed by examining current teaching materials and assessment tools of the IP course. SQ2 is explored by comparing the outcomes of RQ1 and SQ1 with those of the four stages of the ADRI model. This will inform the exploration of RQ3 (impact of ADRI on course materials) and RQ4 (impact on student learning).
4 ANALYSIS OF CURRENT TEACHING AND ASSESSMENT MATERIALS OF INTRODUCTORY PROGRAMMING COURSE

4.3.1 Population and Ethical Consideration

The population being surveyed consisted of IP students in semester 1, 2013-14 at Al-Buraimi University College, Oman. The control group (first survey) consisted of 99 students including 82 female (83%) and 17 male (17%) students. Buraimi University College offers degree programs in two shifts, morning (8am – 2pm) and evening (2pm - 9pm). The morning shift is exclusively offered for female students whereas the evening shift is offered for both male and female students. This practice is not very common in other higher education providers in Oman. Consequently, Buraimi University College attracts more female students compared to male students. This affords a unique opportunity to undertake this investigation with the majority of participants being female, a rare situation in IT. Al-Sebaie (2010) discussed that there was high percentage (almost 65%) of female graduates from IT colleges in Bahrain.

The assessment tools (final exam papers) of those instructors who taught the IP course at Buraimi University College from 2010 to 2013 were accessed and analysed. Current teaching materials in the course were also analysed.

Buraimi University College uses a traditional approach to teaching the IP course. As Mascolo (2009) explained, this is a teacher-centred style of teaching, based on ‘a model of an active teacher and a passive student’ (p.4), whereas the ADRI approach promotes guided participation, in which the teachers play an important role in communicating programming concepts and students are actively involved in practicing and refining these concepts (Mascolo, 2009). As explored in chapter 2 (see section 2.3) teaching materials based on the traditional approach tend to promote programming shortcuts where novice
programmers attempt to write code directly to solve the problem statement (Problem Statement → Code) as shown in figure 1.1.

4.3.2 Preparation of the First Survey
The first survey (provided in appendix B) was prepared to explore the level of difficulty novices experienced when learning about different topics in the IP course. It consists of 38 closed and open ended questions. The first survey has four parts as shown below in figure 4.3.

The first part covers demographic questions related to students’ prior experience in programming, their study major, degree level and gender.

The second part covered questions which are mainly related to the course content. All the topics covered in this course were included. A five-point Likert scale was used, from very difficult to learn (1) to very easy to learn (5). One open-ended question in this section allowed respondents to offer any other relevant comments on the course content. This part of the survey was based on a survey developed by Lahtinen et al. (2005).
Figure 4.3: First survey

The third and fourth parts of the survey focused on questions related to the learning situations and teaching materials of the course. Again, five-point Likert scales were used; questions related to the learning situation use a scale of never (1) to always (5) used in this mode of study, and those relating to the teaching materials used a scale of useless material (1) to very useful material (5). One open-ended question gave respondents the option to comment on any other relevant issues related to teaching and learning aspects of the course which are not otherwise covered by the survey instrument. All the closed ended questions are taken from the survey developed by Lahtinen et al. (2005).

4.4 Results

This section presents the results of the first survey and the analysis of current teaching materials of the IP course. The responses of the first survey were analysed using SPSS software.
4 ANALYSIS OF CURRENT TEACHING AND ASSESSMENT MATERIALS OF INTRODUCTORY PROGRAMMING COURSE

4.4.1 Analysis of student first survey

The student survey consists of following four parts.

1. Demographic details
2. Programming concepts
3. Learning programming
4. Resources

Demographic details

The demographic details of the respondents are as follows:

- 38.4% respondents were enrolled in the Information Systems major, compared to 34.3% from Computer Science and 27.3% from Software Engineering.
- Most of the respondents (72.4%) were pursuing their studies at bachelor level as compared to 3.1% for advanced diploma and 24.5% for diploma.
- The majority of the respondents (82%) were female compared to 18% male.
- Most of the respondents (73.7%) did not have any prior programming experience.

Programming Concepts

The respondents’ perceptions of the ease with which they learn programming concepts is shown in Table 4.1. The means and frequencies of the responses are included in this table.
4 ANALYSIS OF CURRENT TEACHING AND ASSESSMENT MATERIALS OF INTRODUCTORY PROGRAMMING COURSE

Table 4.1: Programming Concepts

<table>
<thead>
<tr>
<th>PROGRAMMING CONCEPTS</th>
<th>Mean</th>
<th>very difficult to learn</th>
<th>difficult to learn</th>
<th>neutral</th>
<th>easy to learn</th>
<th>very easy to learn</th>
<th>Not Applicable</th>
<th>Total N</th>
</tr>
</thead>
<tbody>
<tr>
<td>I found …</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Using the program development environment</td>
<td>2.80</td>
<td>4.1%</td>
<td>15.3%</td>
<td>53.1%</td>
<td>16.3%</td>
<td>4.1%</td>
<td>7.1%</td>
<td>100%</td>
</tr>
<tr>
<td>Gaining access to the computers and networks</td>
<td>3.26</td>
<td>2.1%</td>
<td>12.5%</td>
<td>30.2%</td>
<td>31.3%</td>
<td>16.7%</td>
<td>7.3%</td>
<td>100%</td>
</tr>
<tr>
<td>Understanding problem solving strategies</td>
<td>2.55</td>
<td>17.2%</td>
<td>34.3%</td>
<td>21.2%</td>
<td>21.2%</td>
<td>4.0%</td>
<td>2.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Understanding programming structures</td>
<td>2.53</td>
<td>21.4%</td>
<td>30.6%</td>
<td>26.5%</td>
<td>16.3%</td>
<td>5.1%</td>
<td>0.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Learning the programming language syntax</td>
<td>2.75</td>
<td>11.1%</td>
<td>36.4%</td>
<td>26.3%</td>
<td>17.2%</td>
<td>9.1%</td>
<td>0.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Designing a program to solve a certain task</td>
<td>2.78</td>
<td>6.2%</td>
<td>24.7%</td>
<td>41.2%</td>
<td>14.4%</td>
<td>8.2%</td>
<td>5.2%</td>
<td>100%</td>
</tr>
<tr>
<td>Dividing functionality into procedures</td>
<td>2.85</td>
<td>4.1%</td>
<td>27.8%</td>
<td>33.0%</td>
<td>23.7%</td>
<td>6.2%</td>
<td>5.2%</td>
<td>100%</td>
</tr>
<tr>
<td>Compiling and executing programs</td>
<td>2.93</td>
<td>7.2%</td>
<td>28.9%</td>
<td>33.0%</td>
<td>20.6%</td>
<td>9.3%</td>
<td>1.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Finding bugs in my own program</td>
<td>2.77</td>
<td>13.3%</td>
<td>21.4%</td>
<td>32.7%</td>
<td>20.4%</td>
<td>8.2%</td>
<td>4.1%</td>
<td>100%</td>
</tr>
</tbody>
</table>

The results show that respondents perceived the most significant learning difficulties were with ‘Understanding programming structures’ (2.53), ‘Understanding problem solving strategies’ (2.55) and ‘Learning the programming language syntax’ (2.75). The only programming concept that they appeared to have little or no difficulty with is ‘Gaining access to the computers and networks’ (3.26). The respondents also found ‘Finding bugs in my own program’ (2.77), ‘Designing a program to solve a certain task’ (2.78), ‘Using the program development environment’ (2.80), ‘Dividing functionality into procedures’ (2.85), and ‘Compiling and executing programs’ (2.93) difficult to learn or use.
4 ANALYSIS OF CURRENT TEACHING AND ASSESSMENT MATERIALS OF INTRODUCTORY PROGRAMMING COURSE

The section of the survey related to programming concepts covered all the topics included in the introductory programming course and the outcomes are shown in Table 4.2.

Table 4.2: Teaching Topics

<table>
<thead>
<tr>
<th>TEACHING TOPICS</th>
<th>Mean</th>
<th>very difficult to learn (% / N)</th>
<th>difficult to learn (% / N)</th>
<th>Neutral (% / N)</th>
<th>easy to learn (% / N)</th>
<th>very easy to learn (% / N)</th>
<th>Not Applicable (% / N)</th>
<th>Total %</th>
</tr>
</thead>
<tbody>
<tr>
<td>I found…</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arrays</td>
<td>2.64</td>
<td>19.4%</td>
<td>29.6%</td>
<td>22.4%</td>
<td>19.4%</td>
<td>8.2%</td>
<td>1.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Error handling techniques</td>
<td>3.10</td>
<td>8.1%</td>
<td>13.1%</td>
<td>46.5%</td>
<td>25.3%</td>
<td>7.1%</td>
<td>0.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Expressions</td>
<td>2.76</td>
<td>6.1%</td>
<td>20.2%</td>
<td>44.4%</td>
<td>15.2%</td>
<td>7.1%</td>
<td>7.1%</td>
<td>100%</td>
</tr>
<tr>
<td>Functions</td>
<td>2.49</td>
<td>16.3%</td>
<td>29.6%</td>
<td>19.4%</td>
<td>17.3%</td>
<td>9.2%</td>
<td>8.2%</td>
<td>100%</td>
</tr>
<tr>
<td>Input/output statements</td>
<td>3.40</td>
<td>5.1%</td>
<td>12.2%</td>
<td>32.7%</td>
<td>27.6%</td>
<td>20.4%</td>
<td>2.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Operators</td>
<td>3.04</td>
<td>7.1%</td>
<td>23.2%</td>
<td>34.3%</td>
<td>24.2%</td>
<td>10.1%</td>
<td>1.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Parameters</td>
<td>2.93</td>
<td>4.1%</td>
<td>25.8%</td>
<td>33.0%</td>
<td>21.6%</td>
<td>10.3%</td>
<td>5.2%</td>
<td>100%</td>
</tr>
<tr>
<td>Primitive Data Types</td>
<td>2.79</td>
<td>8.2%</td>
<td>20.4%</td>
<td>44.9%</td>
<td>12.2%</td>
<td>9.2%</td>
<td>5.1%</td>
<td>100%</td>
</tr>
<tr>
<td>Repetition Structures</td>
<td>2.34</td>
<td>25.5%</td>
<td>26.5%</td>
<td>21.4%</td>
<td>11.2%</td>
<td>9.2%</td>
<td>6.1%</td>
<td>100%</td>
</tr>
<tr>
<td>Recursion</td>
<td>2.73</td>
<td>6.5%</td>
<td>18.3%</td>
<td>44.1%</td>
<td>15.1%</td>
<td>7.5%</td>
<td>8.6%</td>
<td>100%</td>
</tr>
<tr>
<td>Selection Structure</td>
<td>2.82</td>
<td>6.3%</td>
<td>27.4%</td>
<td>33.7%</td>
<td>16.8%</td>
<td>10.5%</td>
<td>5.3%</td>
<td>100%</td>
</tr>
<tr>
<td>Variable declaration</td>
<td>2.75</td>
<td>10.4%</td>
<td>14.6%</td>
<td>52.1%</td>
<td>9.4%</td>
<td>8.3%</td>
<td>5.2%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Respondents perceived repetition structures (loops) (2.34), functions (2.49), and arrays (2.4) as the most significantly difficult topics in programming. The concepts that they
4 ANALYSIS OF CURRENT TEACHING AND ASSESSMENT MATERIALS OF INTRODUCTORY PROGRAMMING COURSE

appeared to have little or no difficulty with are ‘Operators’ (3.04), ‘Error handling techniques’ (3.10) and ‘input/output statements’ (3.40). The respondents also found ‘Recursion’ (2.73), ‘variable declaration’ (2.75), ‘Expressions’ (2.76), ‘Primitive data types’ (2.79), ‘Selection structure’ (2.82) and ‘parameters’ (2.93) difficult to learn.

Learning Programming

The next part of the first survey focuses on questions related to learning situations for introductory programming. Table 4.3 below depicts the analysis of learning situations.

<table>
<thead>
<tr>
<th>LEARNING SITUATIONS</th>
<th>Mean</th>
<th>Never (% /N)</th>
<th>Rarely (% /N)</th>
<th>Sometimes (% /N)</th>
<th>Often (% /N)</th>
<th>Always (% /N)</th>
<th>not applicable (% /N)</th>
<th>Total %</th>
</tr>
</thead>
<tbody>
<tr>
<td>I learnt about programming ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In lectures</td>
<td>3.95</td>
<td>7.3%</td>
<td>5.2%</td>
<td>14.6%</td>
<td>26.0%</td>
<td>45.8%</td>
<td>1.0%</td>
<td>100%</td>
</tr>
<tr>
<td>In lab sessions</td>
<td>3.67</td>
<td>11.6%</td>
<td>7.4%</td>
<td>10.5%</td>
<td>16.8%</td>
<td>48.4%</td>
<td>5.3%</td>
<td>100%</td>
</tr>
<tr>
<td>While studying alone</td>
<td>3.17</td>
<td>7.6%</td>
<td>10.9%</td>
<td>34.8%</td>
<td>22.8%</td>
<td>18.5%</td>
<td>5.4%</td>
<td>100%</td>
</tr>
<tr>
<td>While working alone on programming coursework</td>
<td>3.41</td>
<td>6.6%</td>
<td>13.2%</td>
<td>33.0%</td>
<td>22.0%</td>
<td>24.2%</td>
<td>1.1%</td>
<td>100%</td>
</tr>
<tr>
<td>In exercise sessions in small groups</td>
<td>3.32</td>
<td>10.4%</td>
<td>11.5%</td>
<td>30.2%</td>
<td>20.8%</td>
<td>25.0%</td>
<td>2.1%</td>
<td>100%</td>
</tr>
</tbody>
</table>

The most useful learning situations for respondents were ‘in lectures’ (3.95), ‘in lab sessions’ (3.67) and ‘while working alone on programming coursework’ (3.41). The respondents also found ‘in exercise sessions in small groups’ (3.32) and ‘While studying alone’ (3.17) facilitated their learning about programming.
4 ANALYSIS OF CURRENT TEACHING AND ASSESSMENT MATERIALS OF INTRODUCTORY PROGRAMMING COURSE

Resources

The last part of the survey focuses on resources to support teaching of programming. Table 4.4 gives an analysis of respondents’ perceptions of the different kinds of resources available.

<table>
<thead>
<tr>
<th>RESOURCES</th>
<th>Mean</th>
<th>Useless (% / N)</th>
<th>Not very useful (% / N)</th>
<th>Somewhat useful (% / N)</th>
<th>Useful (% / N)</th>
<th>Very useful (% / N)</th>
<th>Not applicable (% / N)</th>
<th>Total N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introductory Programming course book</td>
<td>2.96</td>
<td>8.3%</td>
<td>19.8%</td>
<td>32.3%</td>
<td>26.0%</td>
<td>9.4%</td>
<td>4.2%</td>
<td>100%</td>
</tr>
<tr>
<td>Lecture notes</td>
<td>3.67</td>
<td>4.2%</td>
<td>12.6%</td>
<td>18.9%</td>
<td>40.0%</td>
<td>24.2%</td>
<td>0.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Exercise questions and answers</td>
<td>3.80</td>
<td>4.3%</td>
<td>8.6%</td>
<td>18.3%</td>
<td>35.5%</td>
<td>32.3%</td>
<td>1.1%</td>
<td>100%</td>
</tr>
<tr>
<td>Example programs</td>
<td>3.74</td>
<td>4.2%</td>
<td>5.2%</td>
<td>27.1%</td>
<td>29.2%</td>
<td>32.3%</td>
<td>2.1%</td>
<td>100%</td>
</tr>
<tr>
<td>Still pictures of programming structures</td>
<td>3.36</td>
<td>7.3%</td>
<td>9.4%</td>
<td>25.0%</td>
<td>30.2%</td>
<td>22.9%</td>
<td>5.2%</td>
<td>100%</td>
</tr>
<tr>
<td>Interactive visualisations</td>
<td>3.19</td>
<td>9.4%</td>
<td>8.3%</td>
<td>25.0%</td>
<td>27.1%</td>
<td>21.9%</td>
<td>8.3%</td>
<td>100%</td>
</tr>
</tbody>
</table>

'Exercise questions and answers' (3.80), ‘Example Program’ (3.74) and ‘lecture note’ (3.67) were perceived as the most useful learning resources by respondents. The only concept that they found less useful was ‘introductory programming course book’ (2.96).

The respondents also found ‘Still pictures of programming structures’ (3.36) and ‘interactive visualisations’ (3.19) as useful resources for learning programming.
4 ANALYSIS OF CURRENT TEACHING AND ASSESSMENT MATERIALS OF INTRODUCTORY PROGRAMMING COURSE

4.4.2 Analysis of introductory programming course parameters

This section presents an analysis of the current teaching and assessment materials of the IP course used at Buraimi University College. Five different criteria - analysis of course parameters, types of programming example and problems, presentation styles of programming example and problems, four types of problem solving, and analysis of assessment tool, were used for analysing the teaching and assessment materials as shown below in figure 4.4. The rest of this section explains all these criteria in more details.

Figure 4.4: Analysis of introductory programming materials
4 ANALYSIS OF CURRENT TEACHING AND ASSESSMENT MATERIALS OF INTRODUCTORY PROGRAMMING COURSE

Table 4.5 below shows an analysis of each course parameter including the teaching strategies used in various parts of the introduction programming course.

<table>
<thead>
<tr>
<th>Objective Number</th>
<th>Main Topics</th>
<th>Assessment Methods</th>
<th>Teaching Strategies</th>
<th>Teaching Hours</th>
<th>Inclusion of problem solving</th>
<th>No. of Lab Exercises and Tutorials</th>
<th>No. of Assignments</th>
<th>No. of Lab Exams</th>
<th>No. of Graded Lab Sheets</th>
<th>Discuss Program examples and flow charts</th>
<th>Show Images or diagram</th>
<th>No. of Multimedia or Visual Aids</th>
<th>Total:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-Solution &amp; software development</td>
<td>Test1, Quizzes,</td>
<td>13.4%</td>
<td>6 hrs</td>
<td>13.4%</td>
<td>2 labs</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>-Problem solving strategies e.g. Flowchart, Pseudo code</td>
<td>Final Exam, Assignment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15 hrs</td>
</tr>
<tr>
<td>2</td>
<td>-Data types</td>
<td>Test1, Quizzes,</td>
<td>20%</td>
<td>9 hrs</td>
<td>20%</td>
<td>×</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>-I/O statement</td>
<td>Final Exam,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-Operators</td>
<td>Assignment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-Expression</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-Control structures</td>
<td>Test2, Quizzes,</td>
<td>33.3%</td>
<td>15 hrs</td>
<td>33.3%</td>
<td>×</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>33.3%</td>
</tr>
<tr>
<td></td>
<td>-If statement</td>
<td>Final Exam,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-Loops</td>
<td>Assignment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Functions</td>
<td>Test2, Quizzes,</td>
<td>33.3%</td>
<td>15 hrs</td>
<td>33.3%</td>
<td>×</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>33.3%</td>
</tr>
<tr>
<td></td>
<td>Arrays &amp; String</td>
<td>Final Exam,</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Assignment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

103
This course is 3 credit hours, providing 3 hours of face to face contact per week. It mainly covers basic programming topics such as solution & software development process, problem solving strategies, introduction to C++, control structures, functions and arrays etc. The assessment methods include tests, quizzes, assignment and a final exam. Lectures and lab sheets are used as the primary teaching strategy. Most teaching hours (67%) are allocated to control structures, loops, functions and arrays topics. The fewest teaching hours (13%) are given to solution & software development and problem solving strategies topics. Coverage of Data types, I/O statements, operators and expressions topics are allocated 20% of total teaching hours. Likewise ten lab exercises/tutorials (67%) are prepared for control structures, functions and arrays topics compared to two lab exercises/tutorials (13%) for solution & software development and problem solving strategies topics. Data types, I/O statements, operators and expressions topics are given three lab exercises/tutorials (20%). There is one assignment which covers all the topics. Lab exams and graded lab sheets are not used as assessment methods but rather provide an opportunity for formative assessment. The program examples, flow charts, diagrams are presented in lectures for all topics. Multimedia / visual aids are not used in the teaching process.

Types of Programming Problems and Examples

In the current study, I classified the learning activities for the main topics using the categorization suggested by Wu et al. (1999) as shown in table 4.6. Wu et al. (1999) did a content analysis of programming examples in 16 textbooks used in high school in Taiwan. They divided the programming problem types into five categories: math problem, graphics problems, syntax-oriented problems, real-life problems and others. They explained that math problems include problems like finding the prime numbers,
computing the Nth Fibonacci number and so on. Syntax-oriented problems only demonstrate the purpose of one or more programming features. Daily-life or real-life problems include problems like conversion between Celsius and Fahrenheit temperatures, calculation of monthly dues or bank loans, a card game, and so on. Graphic problems are not included in the introductory programming course and so were omitted from my analysis, resulting in a 4-point categorisation.

For each topic, all the programming examples or problems given in the lecture notes, reading materials and class exercises were categorised against these four categories of example/problem types, and then the total number of programming examples or problems and its percentage were calculated for each category and topic. Lastly, the percentages were calculated separately for lecture notes, reading materials and class exercises in each topic.

After calculating and analysing the average values for each category (see last row of table 4.6 below), it is evident that syntax-oriented problems lead among four types for reading materials (50%) and lecture notes (49%). Math problems are in second position with a share of 29% and 35% for reading materials and lecture notes consecutively. Daily-life problems comprise 15% for reading materials and 13% for lecture notes. The remaining 6% and 3% problems come under category of miscellaneous for reading materials and lecture notes. For class exercises category, math and syntax-oriented problems shares are 41% and 39% consecutively. Daily-life and miscellaneous problems comprise 18% and 3% consecutively.

Through an analysis of the time allocated to each topic by example/problem type, it is evident that syntax oriented problems are allocated the most time to each topic. Math
style problems are allocated the second-most time, while daily life examples are third. Very little time is allocated to examples or problems outside these three types.

Table 4.6: Types of Problems and Examples

<table>
<thead>
<tr>
<th>Main Topics</th>
<th>Lecture Notes</th>
<th>Reading Materials</th>
<th>Class Exercises</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Math</td>
<td>Syntax-oriented</td>
<td>Daily Life</td>
</tr>
<tr>
<td>-Solution &amp; software development</td>
<td>1</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>-Problem solving strategies e.g. -Flowchart -Pseudo code</td>
<td>(43%)</td>
<td>(43%)</td>
<td>(14%)</td>
</tr>
<tr>
<td>-Data types -Operators -Expression</td>
<td>3</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>(20%)</td>
<td>(53%)</td>
<td>(27%)</td>
</tr>
<tr>
<td>-Input / Output statement</td>
<td>1</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(14%)</td>
<td>(72%)</td>
<td>(14%)</td>
</tr>
<tr>
<td>-Control structures -If statement</td>
<td>4</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>(25%)</td>
<td>(38%)</td>
<td>(25%)</td>
</tr>
<tr>
<td>-Loops</td>
<td>6</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>(43%)</td>
<td>(43%)</td>
<td>(0%)</td>
</tr>
<tr>
<td>Functions</td>
<td>6</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(50%)</td>
<td>(42%)</td>
<td>(8%)</td>
</tr>
<tr>
<td>Arrays &amp; String</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>(50%)</td>
<td>(50%)</td>
<td>(0%)</td>
</tr>
<tr>
<td>Average</td>
<td>3.7</td>
<td>5.1</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td>(35%)</td>
<td>(49%)</td>
<td>(13%)</td>
</tr>
</tbody>
</table>
Within these types of examples/problem classifications, the most time in nearly all topics is devoted to lecture notes and reading materials. Class exercises are generally given the least amount of time.

Presentation Styles

De Raadt et al. (2005) stated that IP instructors may appreciate textbooks that include more code examples and that a good textbook includes well-chosen and presented examples. Wu et al. (1999) presented a qualitative analysis of three different presentation styles of the programming examples used in programming textbooks. The details of these presentation styles as follows:

1. Problem Statement → Flowchart or Pseudo code: The problem statement is followed by flowchart or pseudo code
2. Problem Statement → Codes: The problem statement is followed by code with some explanation.
3. Problem Statement → Solution Plan → Codes: The problem statement is followed by solution plan which contains a textual description (like problem analysis, algorithm design, variables etc.) and then the complete code is provided with some explanation.

For each topic, all the programming examples given in the lecture notes and reading materials were categorised against three different presentation styles, then the total number of programming examples and its percentage were calculated for each presentation style and topic. Lastly, the percentages were calculated separately for lecture notes and reading materials in each topic. Table 4.7 shows the different styles used for
presenting programming examples in the lecture notes and reading materials utilised in the IP course.

Table 4.7: Presentation Styles

<table>
<thead>
<tr>
<th>Main Topics</th>
<th>Presentation Styles</th>
<th>Reading Materials</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lecture Notes</td>
<td>Problem Statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>---</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Problem Statement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Flowchart or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pseudo code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>---</td>
</tr>
<tr>
<td>Problem Statement</td>
<td>N (%)</td>
<td>N (%)</td>
</tr>
<tr>
<td>Problem solving strategies e.g.</td>
<td>6 (100%)</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>Data types</td>
<td>0 (0%)</td>
<td>12 (92%)</td>
</tr>
<tr>
<td>Input / Output statement</td>
<td>0 (0%)</td>
<td>6 (86%)</td>
</tr>
<tr>
<td>Control structures</td>
<td>4 (20%)</td>
<td>16 (80%)</td>
</tr>
<tr>
<td>Loops</td>
<td>3 (19%)</td>
<td>13 (79%)</td>
</tr>
<tr>
<td>Functions</td>
<td>0 (0%)</td>
<td>11 (100%)</td>
</tr>
<tr>
<td>Arrays &amp; String</td>
<td>0 (0%)</td>
<td>6 (100%)</td>
</tr>
<tr>
<td>Average</td>
<td>1.8 (20%)</td>
<td>9 (77%)</td>
</tr>
</tbody>
</table>
4 ANALYSIS OF CURRENT TEACHING AND ASSESSMENT MATERIALS OF INTRODUCTORY PROGRAMMING COURSE

On the basis of average values for each presentation style (see last row of table 4.7), it is evident that most of the programming problems/examples are presented by using Problem Statement→Codes presentation styles in lecture notes (77%) and reading materials (70%). The Problem Statement→Flowchart or Pseudo-code is used for 20% lecture notes and 19% reading materials. The least used presentation style is Problem Statement→Solution Plans→Codes. It is used for 3% lecture notes and 12% reading materials.

Four Steps of Problem Solving

Wu et al. (1999) explained problem solving as a four-step process as follows:

1. **The problem analysis step:** The problem statement is explained so that readers can understand it in a better way. No particular method is used at this step.

2. **The solution planning step:** The solution planning is presented by a textual description (like algorithm design, variables, main algorithm etc.).

3. **The coding step:** A complete code with some explanations or comments is given. A sample run is also provided with programs.

4. **The testing/debugging step:** This shows how to debug syntax, logical and run-time errors. A program with planted errors is discussed to show students how to handle errors before a program finally produces the expected output.

Table 4.8 shows analysis of programming examples of IP teaching materials against four problem solving steps.

For each topic, all the programming examples given in the lecture notes and reading materials were categorised against these four problem solving steps, then the total number
of programming examples were calculated for each problem solving step and topic. Lastly, the percentages were calculated separately for lecture notes and reading materials in each topic.

\textit{Table 4.8: Four Problem Solving Steps}

<table>
<thead>
<tr>
<th>Main Topics</th>
<th>Four Problem Solving Steps</th>
<th>Lecture Notes</th>
<th>Reading Materials</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Problem Analysis</td>
<td>Solution Planning</td>
<td>Coding</td>
</tr>
<tr>
<td>-Solution &amp; software development</td>
<td>N (%)</td>
<td>N (%)</td>
<td>N (%)</td>
</tr>
<tr>
<td>-Problem solving strategies e.g. -Flowchart -Pseudo code</td>
<td>0 (0%)</td>
<td>6 (100%)</td>
<td>0 (0%)</td>
</tr>
<tr>
<td>-Data types -Operators -Expression</td>
<td>1 (7%)</td>
<td>1 (7%)</td>
<td>12 (86%)</td>
</tr>
<tr>
<td>-Input / Output statement</td>
<td>1 (11%)</td>
<td>1 (11%)</td>
<td>7 (78%)</td>
</tr>
<tr>
<td>-Control structures -If statement</td>
<td>0 (0%)</td>
<td>4 (20%)</td>
<td>14 (70%)</td>
</tr>
<tr>
<td>-Loops</td>
<td>0 (0%)</td>
<td>3 (19%)</td>
<td>13 (81%)</td>
</tr>
<tr>
<td>Functions</td>
<td>0 (0%)</td>
<td>0 (0%)</td>
<td>11 (100%)</td>
</tr>
<tr>
<td>Arrays &amp; String</td>
<td>0 (0%)</td>
<td>0 (0%)</td>
<td>6 (100%)</td>
</tr>
<tr>
<td>Average</td>
<td>0.3 (3%)</td>
<td>2.1 (22%)</td>
<td>9 (74%)</td>
</tr>
</tbody>
</table>
After calculating and analysing the average values for each problem solving step (see last row of table 4.8), it is evident that coding is the most common problem solving step in lecture notes (74%) and reading materials (61%). The solution planning is placed second with a share of 22% in lecture notes and 20% in reading materials. The problem analysis is discussed in 15% in reading materials and in 3% in lecture notes. The testing/debugging step is the least discussed step in lecture notes (1.4%) and reading materials (4%).

4.4.3 Analysis of Assessment tools

Table 4.9 depicts the analysis of the final examination papers for the programming course.

For each exam paper, all the questions were categorised against five different categories: coding questions, problem solving strategies questions, evaluating output questions from a given piece of code, explanation and definition questions, and miscellaneous questions. The total marks (and percentage) for each category were calculated and are presented in table 4.9.

On average (see the last row in table 4.9), the coding questions (where students are asked to write a program for a given problem statement) are allocated 43% marks available on the examination. Evaluating output questions from a given piece of code include a further 24% marks. Problem solving questions are allocated 15% of the marks, Miscellaneous questions such as fill in the blank, multiple choice etc. a further 17% of total marks and just 1.3% of the marks are allocated for explanation and definition type questions.
### Table 4.9: Analysis of Assessment Tools

<table>
<thead>
<tr>
<th>Final Examination Papers</th>
<th>Coding questions (%) / Marks</th>
<th>Problem Solving Strategies Questions e.g. Flowchart, Pseudo code (%) / Marks</th>
<th>Evaluating Output questions from a given piece of Code (%) / Marks</th>
<th>Explanation and definition Questions (%) / Marks</th>
<th>Miscellaneous (%) / Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>30%</td>
<td>0%</td>
<td>30%</td>
<td>20%</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>P2</td>
<td>44%</td>
<td>24%</td>
<td>16%</td>
<td>0%</td>
<td>16%</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>12</td>
<td>8</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>P3</td>
<td>16%</td>
<td>24%</td>
<td>48%</td>
<td>0%</td>
<td>12%</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>12</td>
<td>24</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>P4</td>
<td>24%</td>
<td>22%</td>
<td>42%</td>
<td>0%</td>
<td>12%</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>11</td>
<td>21</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>P5</td>
<td>32%</td>
<td>24%</td>
<td>30%</td>
<td>0%</td>
<td>14%</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>12</td>
<td>15</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>P6</td>
<td>76%</td>
<td>12%</td>
<td>0%</td>
<td>0%</td>
<td>12%</td>
</tr>
<tr>
<td></td>
<td>38</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>P7</td>
<td>40%</td>
<td>20%</td>
<td>10%</td>
<td>0%</td>
<td>30%</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>10</td>
<td>5</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>P8</td>
<td>36%</td>
<td>0%</td>
<td>44%</td>
<td>0%</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>0</td>
<td>22</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>P9</td>
<td>44%</td>
<td>24%</td>
<td>16%</td>
<td>0%</td>
<td>16%</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>12</td>
<td>8</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>P10</td>
<td>68%</td>
<td>0%</td>
<td>32%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>34</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P11</td>
<td>70%</td>
<td>0%</td>
<td>8%</td>
<td>0%</td>
<td>22%</td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>P12</td>
<td>44%</td>
<td>24%</td>
<td>16%</td>
<td>0%</td>
<td>16%</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>12</td>
<td>8</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>P13</td>
<td>40%</td>
<td>20%</td>
<td>10%</td>
<td>0%</td>
<td>30%</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>10</td>
<td>5</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>P14</td>
<td>44%</td>
<td>24%</td>
<td>16%</td>
<td>0%</td>
<td>16%</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>12</td>
<td>8</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>P15</td>
<td>36%</td>
<td>0%</td>
<td>44%</td>
<td>0%</td>
<td>20%</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>0</td>
<td>22</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>43%</strong></td>
<td><strong>15%</strong></td>
<td><strong>24%</strong></td>
<td><strong>1.3%</strong></td>
<td><strong>17%</strong></td>
</tr>
<tr>
<td></td>
<td><strong>21.5</strong></td>
<td><strong>7.3</strong></td>
<td><strong>12.1</strong></td>
<td><strong>0.7</strong></td>
<td><strong>8.5</strong></td>
</tr>
</tbody>
</table>
4 ANALYSIS OF CURRENT TEACHING AND ASSESSMENT MATERIALS OF INTRODUCTORY PROGRAMMING COURSE

4.5 Discussion

Cycle 1 of the action research is described in this chapter. Two main activities were performed to determine the learning difficulties faced by the novices in the IP course. Firstly, the first survey was conducted with the students’ to determine their perceptions regarding barriers and affordances to learning programming. Secondly, the current practices in the IP course assessment tools and teaching materials regarding programming knowledge and problem solving strategies were analysed.

The analysis of current teaching materials shows that in lectures and lab sessions, more emphasis is given to programming syntax and the least time is allocated for problem solving strategies. Likewise, syntax-oriented and math style programming examples or problems are discussed more and the least time is spent on practical (daily-life) type examples or problems.

These results suggest that an alternative approach is needed which balances the emphasis on syntax with problem solving. I suggest that this approach can be supported by using an ADRI model.

The ADRI approach will be incorporated into the teaching materials of the IP course with a focus on paying equal attention to programming syntax and problem solving strategies. The objective is that novices will acquire programming knowledge and problem solving skills in all examples or problems.
The ADRI model is a general tool for assessment and improvement (Abuid, 2010). It not only provides cycles of improvement but it can also be used by individual and small team members (Carroll and Razvi, 2006; Baird, 2006). It not only helps to understand how someone is doing work but at the same time it identifies shortcomings or pitfalls in a given problem. Therefore, we can say that it provides opportunities for assessment and improved understanding of the given problem.

De Raadt (2008) mentioned that one of the reasons for poor performance of novice programmers (in standardized program generation tests) is the method of preparing and teaching traditional introductory courses which fail to adequately educate the majority of students about the programming and problem solving approaches.

Iqbal et al. (2013) suggest the following tasks and sub tasks in four stages as a part of ADRI model in the context of novice programmers as shown in table 4.10.

(1) Approach: Problem solving strategies

(2) Deployment: Programming knowledge (Syntax and semantics)

(3) Result: Show underlying process to solve problem statements. Analysis and comparison of intended and achieved output. Ensure that intended goals are achieved or not.

(4) Improvement: Learning and recommendations for the instructor and novice programmer

The traditional way of presenting programming examples or problems is to provide the problem statement and then its solution emphasising programming knowledge (syntax
and semantics) rather than problem solving strategies. Novices find it difficult to devise a valid program as they cannot replicate the problem solving strategies.

Table 4.10: Proposed ADRI model four approaches for novice programmers

<table>
<thead>
<tr>
<th>ADRI model four approaches</th>
<th>Task and sub tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage 1: Approach</td>
<td><strong>Problem solving strategies:</strong></td>
</tr>
<tr>
<td></td>
<td>- Understanding and specification (Analyse the problem and ‘what the solution must do’ (Webster, 1994; p. 4).)</td>
</tr>
<tr>
<td></td>
<td>- General solution and algorithm (List the steps to solve problem and specify data types)</td>
</tr>
<tr>
<td></td>
<td>- verify the algorithm (check whether required solution is achieved by following steps)(Webster, 1994)</td>
</tr>
<tr>
<td>Stage 2: Deployment</td>
<td><strong>Programming knowledge (Implementation)</strong></td>
</tr>
<tr>
<td></td>
<td>- ‘Concrete solution” (Webster, 1994; p. 4) (using particular programming language syntax and semantics to develop a program from the algorithm)</td>
</tr>
<tr>
<td></td>
<td>- Testing and debugging syntax errors (Compile the program, find errors, locate the errors in source code and make corrections)(Webster, 1994)</td>
</tr>
<tr>
<td>Stage 3: Results</td>
<td><strong>Analysis:</strong></td>
</tr>
<tr>
<td></td>
<td>- Show underlying processes to solve problem statement</td>
</tr>
<tr>
<td></td>
<td>- Compare intended results with achieved results</td>
</tr>
<tr>
<td></td>
<td>- Figure out any program running errors and correct it</td>
</tr>
<tr>
<td>Stage 4: Improvement</td>
<td><strong>Learning and recommendations:</strong></td>
</tr>
<tr>
<td></td>
<td>For novice programmer:</td>
</tr>
<tr>
<td></td>
<td>- Add more features or functionality in program</td>
</tr>
<tr>
<td></td>
<td>- Try different programming constructs to solve same problem (loops, logical operators)</td>
</tr>
<tr>
<td></td>
<td>For Instructor:</td>
</tr>
<tr>
<td></td>
<td>- Update teaching materials or add more lab questions/sheets</td>
</tr>
</tbody>
</table>
4 ANALYSIS OF CURRENT TEACHING AND ASSESSMENT MATERIALS OF INTRODUCTORY PROGRAMMING COURSE

The ADRI model allows for explanations of programming examples or problems in four stages. The four stages cover problem solving strategies, syntax and semantics, the process used to solve the problem, and improves a deep understanding of programming constructs. Thus, the ADRI model provides a platform for novices to practice all the necessary skills and acquire the knowledge that is required to understand introductory programming concepts, providing greater attention to incorporating problem solving strategies than traditional teaching of introductory programming has done.

This four stage approach helps novices to understand basic programming concepts in a more holistic way. Furthermore, it allows equal emphasis to be paid to problem solving strategies and programming knowledge which helps novices in developing deeper learning of the problem domain. It also provides opportunities for students to better understand the problem statement as the result stage explains the process used to solve the problem statement. Moreover, the model promotes practice by allowing for more programming questions to be included through the improvement stage which helps novices in understanding different programming constructs.

The implementation of the ADRI model in this study promotes more programming practice as compared to traditional teaching. In the first stage, novices have to develop the pseudo code or a flowchart for a problem statement. In the second stage they have to convert the pseudo code or flowchart into a valid computer program. The third stage explains the process used to solve the problem statement. The last stage slightly changes the problem statement so that novices can practice more programming constructs.
4 ANALYSIS OF CURRENT TEACHING AND ASSESSMENT MATERIALS OF INTRODUCTORY PROGRAMMING COURSE

4.5.1 ADRI model to handle programming knowledge and problem solving strategies

The three most difficult issues perceived by respondents in learning introductory programming are ‘understanding programming structures’, ‘learning the programming syntax’ and ‘understanding problem solving strategies’. All these three areas require students not only to understand basic concepts but also to practice them in greater detail. So it is important to develop an IP course which provides an opportunity for students to practice all these three areas throughout the syllabus. The problem solving strategies should be incorporated in all topics of an IP course. Moreover, the students should practice problem solving strategies for programming problems before writing the computer programs. Novices can grasp programming structures in a better way if these are presented to them in different ways such as flowcharts, pseudo-code, and programming language syntax. At the same time, it also helps students to get more exposure to programming structures and debugging skills. It is easier for novices to debug any logical errors in their programs if they have a clear understanding of the processing steps and the outcomes of the problem statement.

The points raised above suggested that incorporating the ADRI approach into the teaching materials will better support student learning. The problem solving strategies such as flowchart, pseudo-code are included in all topics of the IP course (Approach Stage of ADRI model, see section 4.2, Table 4.10 and appendix E). The approach also helps novices to understand programming structures (Deployment Stage of ADRI model). The course materials also include processing steps to solve programming problems or examples (Result and Improvement Stages of ADRI model).
The respondents perceived functions, repetition structures (loops) and arrays as the three most difficult teaching topics in IP course. All these three topics require students to understand programming structures in greater depth. In a traditional teaching, the students are taught all these topics by giving them knowledge and practice of relevant programming syntax. The lack of success in this traditional approach is why there are high failure rates or dropout in IP courses. One solution is to give better exposure of all introductory programming topics to the students by demonstrating these concepts by using problem solving strategies like flow chart and pseudo code in addition to programming syntax. It will help students to better understand programming concepts and give an alternative way to understand programming topics. The ADRI based teaching materials include flow chart and pseudo-code techniques to demonstrate programming concepts for all teaching topics in addition to programming syntax (see section 4.2, Table 4.10 and appendix E).

In the second part of the survey, the respondents reported ‘in lectures’, ‘in lab sessions’ and ‘while working alone on programming coursework’ as the three favourite learning situations for introductory programming. Therefore it is important to develop course material which supports students learning in different situations. The lecture notes and lab exercises should demonstrate programming concepts in different ways. The ADRI based teaching materials will provide programming examples or problems which promote programming syntax and problem solving strategies such as pseudo-code or flowchart. Novices have the opportunity to practice programming syntax and problem solving strategies in all examples and problems. The ADRI based programming examples will
also be discussed in lectures. The programming problems will be practiced in lab sessions or when the students work alone on programming coursework.

The respondents perceived ‘exercise questions and answers’, ‘example program’ and ‘lecture notes’ as the three most useful learning resources for the IP course. Therefore, the course material should provide programming examples in lecture notes and lab exercises. Moreover, the programming examples should be demonstrated in different ways by using problem solving strategies and programming syntax. It will also promote surface and deep learning in novices. In ADRI approach, the programming examples are provided with solutions and discussed in lectures. The programming problems are given for lab sessions. A glossary for the technical terms used in ADRI based teaching materials was prepared and given to the students for better understanding of the programming concepts. An ADRI based editor was prepared to facilitate practice of programming problems in lab sessions, designed specifically to facilitate ADRI based teaching practices to novices.

4.5.2 Analysis of current teaching materials and assessment tools of introductory programming

The problem solving strategies topic is given only 13% teaching time across the whole course. Moreover there are only two lab sheets to practice the problem solving topic. On the other hand, 87% of teaching time and 13 lab sheets are allocated to teach and practice programming language syntax. Lab exams or graded lab sheets are not included as a part of assessment which normally encourages practice in an IP course. The ADRI based teaching materials aimed to increase the amount of time dedicated to problem solving strategies and includes strategies such as pseudo-code and flowchart across all topics in the course. Moreover, pseudo-code and flowchart is incorporated in all lab sheets.
Syntax-oriented programming examples or problems dominated the traditional teaching materials. Math problems are placed on second position and real-life problems come in at third position. To develop the students’ interest in programming problems, it is preferable to introduce more daily-life programming examples or problems. The ADRI based teaching materials embeds syntax-oriented programming examples or problems within a real-life context, transforming the exercises into more interesting problems.

The most prominent presentation style in the current teaching materials for programming problems or examples is, Problem StatementÆCodes. The ADRI based teaching materials instead presents programming problems or examples by using the presentation style of Problem StatementÆProblem solving strategiesÆProgramming knowledge. The flowchart and pseudo-code techniques are used in problem solving strategies. The programming code and processing steps present the programming knowledge. This technique aims to promote deep learning of a problem domain. Moreover, novices will be able to analyse problem statements in greater depth.

The coding step is the most favoured among the four steps of problem solving in the current IP course, followed by the solution planning, problem analysis and testing/debugging steps respectively. The ADRI model four steps (Approach, Deployment, Result, and Improvement) address all these four steps of problem solving. The solution planning and problem analyses are discussed in the Approach step of ADRI model. The coding step is provided in the Deployment step and testing/debugging is addressed in the Improvement step of ADRI model. Moreover, the Result step of ADRI model provides better understanding of problem domain by elaborating the processing
steps of problem statement. It also discusses common syntax or logical errors in programming.

In the current assessment tools, the coding questions are given more weight than other types of questions. Questions prepared according to the ADRI approach will give equal attention to problem solving strategies, programming knowledge and solution planning. This will promote deep learning in novices instead of surface learning.

4.6 Conclusion

In most cases, an IP course is mainly focused on programming knowledge (syntax and semantic). Problem solving strategies are given less importance and attention in respect of teaching time and assessment. As a result, novices tend to have surface learning (or knowledge) of programming concepts instead of deep learning.

An analysis of the introductory programming teaching materials and assessment tools in respect of types of programming examples or problems, presentation style and the four problem solving steps, demonstrates that programming knowledge is dominant in both teaching and assessment practices in the IP course. On the other hand, novices indicated in the survey that problem solving strategies is one of the most difficult issues in learning introductory programming. I propose an ADRI based approach in the IP course to help novices in overcoming these issues. The four stages of ADRI approach promote programming knowledge and problem solving strategies. The programming examples or problems are presented in a way which encourages novices to pay equal attention to programming knowledge and problem solving strategies. Moreover, it also helps novices to understand the underlying processes to solve the problem statement.
In the teaching materials of the IP course, problem solving strategies are discussed only in introductory topics. The proposed ADRI approach incorporates problems solving strategies in all topics of the IP course. Therefore, it supports deep learning of programming concepts.

The ADRI based approach promotes Problem Statement  → Problem solving strategies → Programming knowledge presentation styles in demonstrating programming examples and problems. It helps novices in understanding programming concepts in different ways.
5 ADRI based Introductory Programming Course

5.1 Introduction

This chapter describes cycle 2 of the action research used in this study as shown in figure 5.1. There were two activities performed in this cycle. Firstly, the second survey was conducted with the students of the treatment group who finished the introductory programming (IP) course with the ADRI approach. Secondly, a focus group was run with the instructors of the IP who taught this course with the ADRI approach.

![Figure 5.1: Three cycles of the action research with emphasis on the cycle 2](image)

It is evident from the outcomes of cycle 1 outcomes that more emphasis is given to programming syntax in lectures and lab sessions, and the least time is allocated for...
problem solving strategies. Likewise, syntax-oriented and math style programming examples or problems are discussed more during the classes and the least time is spent on practical (daily-life) type examples or problems. These suggest that an alternative approach is needed which balances the emphasis on learning syntax and problem solving. I have suggested that this approach can be supported by using the ADRI model.

In this chapter, I address the following two research questions:

**RQ1. What are the perceptions of students regarding the ADRI approach during their introductory programming course?**

**RQ2. What are the perceptions of introductory programming instructors regarding the ADRI approach in their teaching process?**

The following sub-question is investigated to inform the interpretation of the results of this cycle.

**SQ1. How can we incorporate the ADRI approach when preparing teaching materials for introductory programming courses?**

Research questions 1 and 2 are discussed in sections 5.5.1 and 5.5.2 respectively. Likewise, sub-question 1 is explored in sections 5.2 and 5.3. The implementation of the ADRI based approach is described in section 5.4, followed by the evaluation of the approach. Section 5.6 discussed the outcomes of the survey and focus group.

### 5.2 ADRI model four stages for novices

The ADRI based approach was introduced in a new design of the course. The ADRI approach for novice programmers (Jantti, 2002; Abuid, 2010; Iqbal & Harsh, 2013) is as shown in Table 5.1. Each stage of the ADRI model and its translation into the approach
for novice programming are described in the following subsections. Table 5.1: ADRI model four approaches for novice programmers

<table>
<thead>
<tr>
<th>ADRI model</th>
<th>ADRI approach for novices</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Approach:</strong></td>
<td><strong>Approach for novice programmers:</strong></td>
</tr>
<tr>
<td>Strategies, planning, goals</td>
<td>Strategies for solving problem statements:</td>
</tr>
<tr>
<td></td>
<td>Pseudo-code, Flowchart, Understanding the problem, Writing and verifying algorithms</td>
</tr>
<tr>
<td><strong>Deployment:</strong></td>
<td><strong>Deployment for novice programmers:</strong></td>
</tr>
<tr>
<td>Implementation and doing</td>
<td>Programming knowledge:</td>
</tr>
<tr>
<td></td>
<td>Translating algorithm (output of step 1) into computer program by using particular programming language</td>
</tr>
<tr>
<td><strong>Result:</strong></td>
<td><strong>Result: for novice programmers:</strong></td>
</tr>
<tr>
<td>Monitoring and evaluating</td>
<td>Process of solving problem statement</td>
</tr>
<tr>
<td></td>
<td>Analysis of program output whether achieved output is the same as the expected output.</td>
</tr>
<tr>
<td><strong>Improvement:</strong></td>
<td><strong>Improvement:</strong></td>
</tr>
<tr>
<td>Learning and adapting</td>
<td>Learning and recommendations</td>
</tr>
<tr>
<td></td>
<td>For novice programmer:</td>
</tr>
<tr>
<td></td>
<td>Add more features or functionality in program</td>
</tr>
<tr>
<td></td>
<td>Try different programming constructs to solve same problem (loops, logical operators)</td>
</tr>
<tr>
<td></td>
<td>For Instructor:</td>
</tr>
<tr>
<td></td>
<td>Update teaching materials or add more lab questions/sheets</td>
</tr>
<tr>
<td></td>
<td>Discuss topics again in the class</td>
</tr>
<tr>
<td></td>
<td>Change teaching methodology</td>
</tr>
</tbody>
</table>
5 ADRI BASED INTRODUCTORY PROGRAMMING COURSE

5.2.1 Approach Stage
Bachu and Bernard (2014) stated that ‘Lack of problem solving skill has been identified as the major cause of students’ failure in introductory programming courses’ (p.1). The Approach stage in the ADRI model is designed to address the lack of focus on problem solving strategies in traditional approaches. It provides a platform for novices to understand the problem statement by using a variety of problem solving strategies such as pseudo-code and flowcharts. This stage helps novices to develop problem solutions without concerning themselves with syntax errors. It will help them to understand the problem domain.

Cutts et al. (2014) stated that ‘Pseudo-code is typically considered to be a blend of formal and natural languages, used for human understanding of algorithms rather than machine understanding’ (p.2). Pseudo-code will help novices to concentrate on algorithm development skills rather than focusing on synthetic details of programming languages (Webopedia, 2015). They can easily write pseudo-code in any text editor without having to concern themselves with program development environments in the early stages of their learning. The pseudo-code requires a less steep a learning curve for novices compared to programming languages. They can also use pseudo-code in structured design techniques (Nishimura, 2007). The pseudo-code cannot be compiled or executed because it is language-independent (Webopedia, 2015). There are no syntax rules for pseudo-code - rather it is written in a formal-style natural language. The novices can also use pseudo-code as a basis for their source program comments. It is sometimes considered as a first stage to writing a program in a computer language (Wikibook, 2015).
Bachu and Bernard (2014) mentioned that ‘Flowcharts are a visual representation of program flows using a combination of arrows and symbols to represent the actions and sequence of the program’ (p. 1). A flowchart provides a graphical representation of a possible solution to a problem rather than textual representation in pseudo-code. Flowcharts are built from standardised symbols which represent different constructs within the solution (Ravichandran, 2001). A flowchart helps novices to understand the flow of logic in an algorithm. It also depicts the relationships between different steps in developing an algorithm to solve a problem. Like pseudo-code, it allows novices to code the program from the flowchart representation of the solution. Flowcharts also represent data flow within the solution to the problem. The flowchart represents the problem solution in a diagrammatic way where different kinds of boxes represent different steps of the problem statement. However they are difficult to modify compared to pseudo-code and special software is required to draw them.

Cooper (2014) suggests that in an IP course it is a good habit for students to write pseudo-code before creating a program. He introduced an IP course based on flow chart and pseudo-code for novices. Bachu and Bernard (2014) introduced a web based strategy game, collaborative online problem solving (COPS), in an IP course to enhance the problem solving skills of novice programmers. ‘COPS has both single player and multiplayer modes and players are required to solve program flowchart puzzles’ (Bachu & Bernard, 2014; p.1). The COPS presents the problem solution as flowchart and equivalent pseudo-code for a given problem statement. The novices ‘have two visual representations of the problem solution and can more easily follow the logic in their
solution’ (p. 1). Therefore both pseudo-code and flowcharts are considered as important tools for teaching and learning of an introductory programming.

5.2.2 Deployment Stage
The deployment stage provides an opportunity for novices to practice programming knowledge (syntax and semantics). They learn structure or grammar of the language (syntax). The syntax is mainly concerned with how to write a valid programming statement. For example, terminate each C++ statement with a semi-colon, enclose an IF statement inside parentheses. On the other hand, the semantics deal with the meaning of the programming statement. The main concern in semantics is whether a programming statement is valid. If so, what does the programming statement mean? (Tucker, 2006) For example, x++; is a syntactically valid C++ statement and it means that the value of the variable x is incremented by one. This stage emphasises both the syntax and semantics of a programming language. All the programming languages have syntax rules and the novices will learn and use these rules to convert the already developed algorithm (pseudo-code and flowchart) in the approach stage into a computer program. The novices acquire knowledge and skills about syntax and semantics in this stage. Moreover, the novices have already developed their algorithm for the problem statement in the approach stage so during the deployment stage they can focus more on the syntax and semantics of the programming language.

5.2.3 Result Stage
The third stage in the ADRI model is Result. It focuses mainly on the input, process and output aspects of the problem statement. The novices will comprehend the flow of logic and data in this stage. The result stage will first show the inputs of a computer program.
Then it will discuss the process used in the program to obtain the output. Furthermore, it will show the expected output of the program. The novices will get a better understanding of the problem domain by analysing its underlying process. Moreover, they will get information about the inputs and outputs of the program. They can use this information to compare their program output with the expected output. Furthermore, they can decide whether the program goals are achieved or not.

5.2.4 Improvement Stage
Improvement is the fourth stage in the ADRI model. It mainly focuses on modifying the problem statement so that novices can get exposure to different programming constructs. For example, if they have solved the problem statement in the approach stage using a For loop, they will solve the same problem statement in the improvement stage with do-while loop. It will help novices to get more practice and understanding of programming constructs, helping them move from surface learning to deep learning of programming concepts. The improvement stage will also give better understanding of different programming concepts and constructs to the novices. They will have the opportunity of comparing different programming constructs to solve the same problem statement which will give them hands-on experience about the implementation challenges for real world problems.

5.3 ADRI based Introductory Programming Course Materials
The course materials used in the IP course were redesigned to take advantage of the benefits of using the ADRI model. Learning resources were developed based on the four stages of the ADRI model. The changes to the programming examples used in lectures
and the programming problems set as laboratory exercises are discussed in the following sub-section. An ADRI based editor was developed to support students in the preparation of programming solutions.

5.3.1 ADRI based Programming Examples or Problems
Programming examples or problems are considered to be important resources for learning to program (Börstler et al., 2009). In this study, for all topics in the introductory programming course, examples and problems were prepared based on the four stages of the ADRI approach. The programming examples are discussed in the class. The programming problems are given to the novices to complete during laboratory sessions (see Appendix E).

All the programming examples and problems have five parts. The first part contains a problem statement. Table 5.2 below, shows the first part where the problem statement is given. In the context of this research project, English is a second language for students and the IP course is offered in the first semester of their degree program. Al-Nuaim et al. (2011) mentioned that students’ lack of English language abilities in the Gulf region is one of the reasons for poor understanding of programming language concepts. Therefore, problem statements (for the ADRI based programming examples and problems), in most of the cases, are simple, easy to understand and comprehensive so that language difficulties are minimised and novices will be able to understand the requirements and expected output clearly.

Wu et al. (1999) categorised programming examples or problems into five types: math problems, graphics problems, syntax-oriented problems, real-life problems and others. I
selected problem statements to cover four of the above mentioned types. The graphic problems are not covered in this course.

Table 5.2: Programming example based on four stages of ADRI approach

<table>
<thead>
<tr>
<th>Step1: Approach – Problem solving strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pseudo-code</strong></td>
</tr>
<tr>
<td>1. Start</td>
</tr>
<tr>
<td>2. Read Number (N)</td>
</tr>
<tr>
<td>3. Calculate remainder (R) of Number with 2</td>
</tr>
<tr>
<td>4. R = N % 2</td>
</tr>
<tr>
<td>5. If (R == 0) Then</td>
</tr>
<tr>
<td>6. Print Even</td>
</tr>
<tr>
<td>7. Else</td>
</tr>
<tr>
<td>8. Print Odd</td>
</tr>
<tr>
<td>9. EndIf</td>
</tr>
<tr>
<td>10. Stop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step2: Programming Knowledge</th>
</tr>
</thead>
<tbody>
<tr>
<td># include &lt;iostream&gt;</td>
</tr>
<tr>
<td>using namespace std;</td>
</tr>
<tr>
<td>int main()</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>int N, R;</td>
</tr>
<tr>
<td>cout&lt;&lt; &quot;Please, enter a number: &quot;;</td>
</tr>
<tr>
<td>cin&gt;&gt; N;</td>
</tr>
<tr>
<td>R = N % 2;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step3: Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expected output:</strong> Please, enter a Number: 3</td>
</tr>
<tr>
<td>3 is Odd number</td>
</tr>
<tr>
<td><strong>Process:</strong> R = 3 % 2 (The remainder is 1 not 0 so it is odd number)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step4: Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Update above program so it checks that the entered number is not negative number.</strong></td>
</tr>
<tr>
<td><strong>Expected Output:</strong> Please, enter a Number: -10</td>
</tr>
<tr>
<td>You entered negative number. Please enter positive number only</td>
</tr>
</tbody>
</table>
The second part (approach stage) covers problem solving strategies. Novices have to write pseudo-code and draw flowchart diagram for a given problem statement. Table 5.2 (above) shows the second part where pseudo-code and flowchart are presented for the problem statement. One of the advantages of this approach for novices is to understand the problem domain without having to concern themselves with the syntax of the programming language. Moreover it gives them an opportunity to think about the problem statement in different ways (write pseudo-code and draw flowchart diagram) which provides them with deeper learning of the programming concepts than they would otherwise achieve. As mentioned earlier, the tasks in this part are completed by writing pseudo-code and drawing a flowchart which also facilitates different learning styles for novices to understand the problem statement in a better way.

The third part (deployment stage) deals with the syntax and semantics of the programming language. Table 5.2 (above) shows the third part where novices translate the pseudo-code or flowchart into a computer program by using C++ language. They are able to focus on the syntax and semantics of the programming language because the logic to solve the problem statement is already developed in the previous stage (problem solving strategies). They will learn and practice programming language commands and rules in this section.

The fourth part (result stage) covers the program process, expected outputs and examples of common syntax errors. Table 5.2 (above) shows the expected output and the program process for the problem statement. One of the important aspects for novices in developing programming logic is to have a proper understanding of the process involved in solving the problem statement. The result stage demonstrates the process used to solve a problem.
statement which helps novices to move from surface learning to deep learning for basic programming concepts. Moreover it shows the format and values of expected output of a program. The novices can compare their achieved output with the expected output. In case of any difference in achieved and expected output, they can explore further and discuss it with their instructors and/or peers. Ultimately, the result stage (fourth part) helps novices to achieve the expected goals set for them to solve the given problem statement. The Result stage also provides information about common syntax errors.

The fifth part (improvement stage) provides more practice and hands-on experience of different programming language constructs. Table 5.2 shows that the initial problem statement is extended in this stage with one additional requirement being added, the requirement to check for positive or negative numbers entered by a user. This additional requirement requires novices to re-think their problem solving strategies and how to translate this strategy into their program. Novices will modify the program which is already developed by them in the third stage to incorporate the new requirements. It gives them more practice to deal with the problem domain and helps them in understanding different programming constructs. It also provides an opportunity for them to interact with instructors by discussing the new requirements in the problem statement. Likewise, instructors can also gain an understanding of students’ performance in the programming tasks, and receive feedback on the effectiveness of their teaching methodology and the students’ overall learning process.

5.3.2 ADRI based Editor
An editor was developed to support novices using the ADRI based approach. It has some distinguishing features compared to other generally available programming editors. It has
been specifically designed to emphasise the four stages of the ADRI approach with separate interfaces for each stage. The exercise questions are embedded in the editor. Moreover the exercise questions can be accessed directly related to particular topics of the introductory programming course.

The ADRI-based editor is a simple editor based on the Java language. Java is a platform independent language (Spell, 2015) so it is compatible with different operating systems.

Figure 5.2: Interface of ADRI based Editor

Figure 5.2 shows one of the user interfaces of the ADRI-based editor. It consists of six menus (File, Edit, ADRI, Question, Topic and Execute). Each menu has further sub-menus which provide further functionality for the novices. There is a separate interface for each stage of the ADRI approach. Each interface shows the stage name (e.g. in figure 5.2 ADRI: Approach (Problem solving strategies)) at the top of the page. Then it shows a problem statement along with the information of the topic to which it belongs in the IP course. For example, in figure 5.2, the problem statement belongs to Algorithms & Pseudo-code topic. A text area is provided for the students to write the solution of a given
problem statement which can be written in either pseudo-code or a programming language.

The file menu has four sub-menu options which are New, Open, Save and Exit. It provides basic functionality to create a new program, open an existing program, save a program, or close the editor.

The edit menu provides three sub-menu which are cut, copy and paste. The novices can use these options to move text from one place to another. Moreover they can save time by reusing text they have developed previously.

The ADRI menu has four sub-menu options reflecting each stage, Approach, Deploy, Result and Improvement. The first sub-menu option, Approach, deals with problem solving strategies. In this sub-menu, novices focus on writing pseudo-code and drawing a flowchart for a given problem statement. The interface of the Approach sub-menu shows title, flowchart button, topic, problem statement and text area. The Edraw software is connected with the editor to assist with drawing the flowchart; it is invoked by pressing the flowchart button as shown in figure 5.3. The second sub-menu option Deploy focuses on programming knowledge (syntax and semantics). In this sub-menu, novices practice how to convert pseudo-code or flowchart into a computer program by using a programming language. This stage emphasises syntax and semantics of the programming language. The interface of the Deploy sub-menu displays title, topic, problem statement and text area. The third sub-menu option, Review, displays the expected output of the problem statement. The interface of the Review sub-menu depicts title and expected output. The last sub-menu option, Improve, also deals with programming knowledge.
(syntax and semantics). The problem statement of the Approach stage is updated or modified in such a way that it provides more practice of different programming constructs to the novices. The interface of the Improve sub-menu displays title, modified problem statement and text area.

*Figure 5.3: Edraw interface for drawing flowcharts*

The Question menu consists of two sub-menus which are Next and Previous as shown below in figure 5.4. The novices can use these sub-menus to move back and forth easily from one exercise question to another. When the question is changed by using the sub-menus options, it is updated for all four stages of the ADRI approach.

*Figure 5.4: Question menu with Next and Previous sub-menus*
The Topic menu has seven sub-menu options which are the topics included in the IP course (e.g. Basic elements of C++) as shown below in figure 5.5. Students can go directly to the exercise questions related to a particular topic of the IP course by using these sub-menus. It saves their time and they can quickly search the exercise questions.

![Topic menu showing all introductory programming topics as sub-menus](image)

*Figure 5.5: Topic menu with introductory programming topics as sub-menu*

The Execute menu provides one sub menu option which is the DOS prompt. The novices can use this option to compile their programs.

The main purpose of the editor is to facilitate the learning process based on the four stages of the ADRI approach. The separate views for each of the four stages of the ADRI approach discourage students from taking programming shortcuts, which helps to address the issues raised in section 2.8 related to the traditional approach. It provides some unique features; in particular all the questions are embedded in it and students can choose questions by topic. On the other hand, it does not provide advance features which were discussed in section 2.5.
5.4 Implementation of ADRI based approach

The IP course offered in semester 1, 2014-15 included the ADRI approach. The ADRI approach was introduced in lectures and lab sessions, and all the programming examples and problems were presented using this new approach. The programming examples that were discussed in lectures and programming problems that were presented in lab sessions were described in terms of the ADRI approach. Students used the ADRI based editor in lab sessions to solve the programming problems. The students could easily access all the programming problems for all the introductory programming topics through the Topic and Question menus of the editor. Moreover, the editor’s separate view for each of the four stages of the ADRI approach helped them in practicing and acquiring the different skills required for the programming domain.

The new approach was also incorporated into assessment tools. The assignment tasks were based on the four stages of the ADRI approach. Students had to write pseudo-code, draw flowcharts, write an underlying process to solve the problem statement and produce a program (using C++) for the given problem statement.

In the exam, different types of questions were given which cover programming knowledge (syntax and semantics) and problem solving skills such as write pseudo-code, draw flowchart, write a program by using C++ language for a given problem statement.

5.5 Evaluation of ADRI based approach

There were two main entities involved in the implementation of the ADRI approach (instructors and students). Likewise, both of them were also involved in evaluating the
ADRI based approach. The students who completed the IP course in semester 1 of 2014-15 completed the second survey using a similar survey instrument as was used in the survey of students in Cycle 1. A focus group was also conducted with the instructors who were involved in teaching this course in semester 1 of 2014-15 to obtain their feedback on the ADRI approach.

5.5.1 Second Survey with treatment group (cycle 2 students)
The second survey was conducted with the students’ completing the introductory programming course in semester 1, 2014-15. The treatment group (second survey) consisted of 118 students including 89 female (75 %) and 29 male (25%) students. The questions in the second survey (treatment group) are the same as the first survey (control group). See appendices B and C for more information about the first and second surveys.

5.5.1.1 Demographic details of second survey respondents
The demographic details of the second survey respondents as follows:

- 36.8 % respondents were enrolled in the Computer Science major, compared to 33.3% from Information Systems and 29.9% from Software Engineering.
- Most of the respondents (69.6 %) were pursuing their studies at bachelor level as compared to 0.9% for advanced diploma and 29.6% for diploma.
- The majority of the respondents (77.4%) were female compared to 22.6% male.
- Most of the respondents (77.4%) did not have any prior programming experience.
5 ADRI BASED INTRODUCTORY PROGRAMMING COURSE

5.5.1.2  Comparison of First and Second Surveys Results
As discussed earlier, the questions in the two surveys were the same so that I could undertake a direct comparison and determine the impact of the ADRI approach on the treatment group.

The results of the first survey were described in chapter 4 (section 4.4). In this chapter, I present the results of the second survey and then compare these with the results of the first survey. I use the Mann-Whitney U Test to compare the outcomes for the control and treatment groups. Hypothesis testing was performed on the set of questions included in the first and second surveys to determine their significance level.

The demographic profiles of the students in both surveys were similar. The distribution of the students in their major area of study and qualification level is almost identical in both surveys. Female students are higher in numbers compared to the male students in both surveys. Those students who have prior programming experience are almost identical in both surveys.

5.5.1.3  Programming Concepts
Table 5.3 depicts the means and frequencies of questions related to the programming concepts part of the second survey. Overall, if we compare the results (means and frequencies) with the control group in cycle 1 (see Table 4.1 for control group), there is a positive improvement in the treatment group suggesting that using the ADRI approach has had a positive influence on student perceptions.
<table>
<thead>
<tr>
<th>PROGRAMMING CONCEPTS</th>
<th>Treatment Group (Second Survey)</th>
<th>Mean</th>
<th>very difficult to learn (%/N)</th>
<th>difficult to learn (%/N)</th>
<th>Neutral (%/N)</th>
<th>easy to learn (%/N)</th>
<th>very easy to learn (%/N)</th>
<th>not applicable (%/N)</th>
<th>Total %</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>I found …</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Using the program development environment</td>
<td>3.17</td>
<td>1.8%</td>
<td>13.3%</td>
<td>55.8%</td>
<td>20.4%</td>
<td>8.0%</td>
<td>0.9%</td>
<td>100%</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>Gaining access to the computers and networks</td>
<td>3.65</td>
<td>0.9%</td>
<td>6.3%</td>
<td>38.7%</td>
<td>30.6%</td>
<td>22.5%</td>
<td>0.9%</td>
<td>100%</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>Understanding problem solving strategies (Flowchart, Pseudo code, Structured English etc.)</td>
<td>3.02</td>
<td>6.8%</td>
<td>17.9%</td>
<td>47.0%</td>
<td>23.1%</td>
<td>5.1%</td>
<td>0.0%</td>
<td>100%</td>
<td>8</td>
<td>21</td>
</tr>
<tr>
<td>Understanding programming structures</td>
<td>2.85</td>
<td>10.3%</td>
<td>19.7%</td>
<td>49.6%</td>
<td>16.2%</td>
<td>4.3%</td>
<td>0.0%</td>
<td>100%</td>
<td>12</td>
<td>23</td>
</tr>
<tr>
<td>Learning the programming language syntax</td>
<td>3.09</td>
<td>5.1%</td>
<td>22.2%</td>
<td>37.6%</td>
<td>24.8%</td>
<td>9.4%</td>
<td>0.9%</td>
<td>100%</td>
<td>6</td>
<td>26</td>
</tr>
<tr>
<td>Designing a program to solve a certain task</td>
<td>2.86</td>
<td>7.7%</td>
<td>25.6%</td>
<td>34.2%</td>
<td>24.8%</td>
<td>5.1%</td>
<td>2.6%</td>
<td>100%</td>
<td>9</td>
<td>30</td>
</tr>
<tr>
<td>Dividing functionality into procedures</td>
<td>3.16</td>
<td>1.8%</td>
<td>17.5%</td>
<td>43.0%</td>
<td>25.4%</td>
<td>9.6%</td>
<td>2.6%</td>
<td>100%</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>Compiling and executing programs</td>
<td>3.15</td>
<td>4.3%</td>
<td>25.9%</td>
<td>32.8%</td>
<td>25.0%</td>
<td>12.1%</td>
<td>0.0%</td>
<td>100%</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>Finding bugs in my own program</td>
<td>3.19</td>
<td>5.9%</td>
<td>18.6%</td>
<td>38.1%</td>
<td>24.6%</td>
<td>12.7%</td>
<td>0.0%</td>
<td>100%</td>
<td>7</td>
<td>22</td>
</tr>
</tbody>
</table>

The means are shown in figure 5.6 below, where means of control and treatment groups are compared for the set of programming concepts questions.
It is clearly evident from figure 5.6 that there is an improvement in the mean values of the treatment group compared to the control group, with the treatment group finding it easier, overall, to learn the programming concepts.

The Mann-Whitney U test is applied to these data and the outcomes are shown in Table 5.4. I performed hypothesis testing (null and alternative hypotheses) on the results produced by the Mann-Whitney U test on the set of questions included in the first and second surveys. The general statements for the null and alternative hypotheses tested in this research are as follows:

*Null hypothesis* ($H_0$) There is no significant difference between values for the same question in the two surveys.

*Alternative hypothesis* ($H_1$) There is significant difference between values for the same question in the two surveys.

The null hypothesis ($H_0$) is accepted, if the value of $p$ (2-tailed Asymptotic Significance) calculated by the Mann-Whitney U test is greater or equal to the $\alpha$ value of 0.05.
Otherwise, if it is less than the α value of 0.05, the alternative hypothesis (H₁) is accepted.

The results of hypothesis testing for the set of programming concepts questions are shown in table 5.4.

**Table 5.4: Mann-Whitney U test for programming concepts**

<table>
<thead>
<tr>
<th>Group</th>
<th>N</th>
<th>Mean Rank</th>
<th>Sum of Ranks</th>
<th>Mann-Whitney U</th>
<th>Asymp. Sig. (2-tailed) P</th>
<th>Hypothesis testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the program development environment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>98</td>
<td>97.02</td>
<td>9508.00</td>
<td>4657.000</td>
<td>.029</td>
<td>p = .029 &lt; 0.05</td>
</tr>
<tr>
<td>Treatment</td>
<td>113</td>
<td>113.79</td>
<td>12858.00</td>
<td></td>
<td></td>
<td>H₁ accepted</td>
</tr>
<tr>
<td>Total</td>
<td>211</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gaining access to the computers and networks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>96</td>
<td>96.45</td>
<td>9259.00</td>
<td></td>
<td></td>
<td>p = .079 &gt; 0.05</td>
</tr>
<tr>
<td>Treatment</td>
<td>111</td>
<td>110.53</td>
<td>12269.00</td>
<td></td>
<td>.079</td>
<td>H₁ rejected</td>
</tr>
<tr>
<td>Total</td>
<td>207</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Understanding problem solving strategies</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>99</td>
<td>94.31</td>
<td>9337.00</td>
<td></td>
<td></td>
<td>p = .001 &lt; 0.05</td>
</tr>
<tr>
<td>Treatment</td>
<td>117</td>
<td>120.50</td>
<td>14099.00</td>
<td></td>
<td>.001</td>
<td>H₁ accepted</td>
</tr>
<tr>
<td>Total</td>
<td>216</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Understanding programming structures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>98</td>
<td>97.54</td>
<td>9558.50</td>
<td></td>
<td></td>
<td>p = .018 &lt; 0.05</td>
</tr>
<tr>
<td>Treatment</td>
<td>117</td>
<td>116.76</td>
<td>13661.50</td>
<td></td>
<td>.018</td>
<td>H₁ accepted</td>
</tr>
<tr>
<td>Total</td>
<td>215</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learning the programming syntax</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>99</td>
<td>98.01</td>
<td>9703.00</td>
<td></td>
<td></td>
<td>p = .019 &lt; 0.05</td>
</tr>
<tr>
<td>Treatment</td>
<td>117</td>
<td>117.38</td>
<td>13733.00</td>
<td></td>
<td>.019</td>
<td>H₁ accepted</td>
</tr>
<tr>
<td>Total</td>
<td>216</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Designing a program to solve a certain task</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>97</td>
<td>105.12</td>
<td>10196.50</td>
<td></td>
<td></td>
<td>p = .594 &gt; 0.05</td>
</tr>
<tr>
<td>Treatment</td>
<td>117</td>
<td>109.47</td>
<td>12808.50</td>
<td></td>
<td>.594</td>
<td>H₁ rejected</td>
</tr>
<tr>
<td>Total</td>
<td>214</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dividing functionality into procedures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>97</td>
<td>97.44</td>
<td>9452.00</td>
<td></td>
<td></td>
<td>p = .050 &gt; 0.05</td>
</tr>
<tr>
<td>Treatment</td>
<td>114</td>
<td>113.28</td>
<td>12914.00</td>
<td></td>
<td>.050</td>
<td>H₁ rejected</td>
</tr>
<tr>
<td>Total</td>
<td>211</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiling and executing programs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>97</td>
<td>100.92</td>
<td>9789.50</td>
<td></td>
<td></td>
<td>p = .173 &gt; 0.05</td>
</tr>
<tr>
<td>Treatment</td>
<td>116</td>
<td>112.08</td>
<td>13001.50</td>
<td></td>
<td>.173</td>
<td>H₁ rejected</td>
</tr>
<tr>
<td>Total</td>
<td>213</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finding bugs in my own program</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>98</td>
<td>97.72</td>
<td>9576.50</td>
<td></td>
<td></td>
<td>p = .017 &lt; 0.05</td>
</tr>
<tr>
<td>Treatment</td>
<td>118</td>
<td>117.45</td>
<td>13859.50</td>
<td></td>
<td>.017</td>
<td>H₁ accepted</td>
</tr>
<tr>
<td>Total</td>
<td>216</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An examination of the findings in Table 5.4 reveals that there are some areas such as Using the program development environment \( (p=0.029) \), Understanding problem solving strategies \( (p=0.001) \), Understanding programming structures \( (p=0.018) \), Learning the programming syntax \( (p=0.019) \), and Finding bugs in my own program \( (p=0.017) \) which have significant differences at the level of \( p<0.05 \). On the other hand, there are some areas where there are insignificant differences between the control and treatment groups such as Gaining access to the computers and networks \( (p=0.079) \), Designing a program to solve a certain task \( (p=0.594) \), Dividing functionality into procedures \( (p=0.050) \), and Compiling and executing programs \( (p=0.173) \).

Table 5.5 shows the mean values and frequencies of the responses related to the set of teaching topics questions of the second survey.

There is a positive improvement in the results of the treatment group, if we compare the frequencies of the control (see Table 4.2 for control group) and treatment groups. Moreover, the improvement is clearly evident in figure 5.7 below, where we compare mean values of the control and treatment groups for of the set of teaching topics.
Table 5.5: Teaching Topics (treatment group)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Mean</th>
<th>very difficult to learn (%/N)</th>
<th>difficult to learn (%/N)</th>
<th>Neutral (%/N)</th>
<th>easy to learn (%/N)</th>
<th>very easy to learn (%/N)</th>
<th>not applicable (%/N)</th>
<th>Total %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrays</td>
<td>3.11</td>
<td>3.6%</td>
<td>15.2%</td>
<td>51.8%</td>
<td>21.4%</td>
<td>7.1%</td>
<td>0.9%</td>
<td>100%</td>
</tr>
<tr>
<td>Error handling techniques</td>
<td>3.19</td>
<td>2.6%</td>
<td>17.1%</td>
<td>46.2%</td>
<td>23.1%</td>
<td>10.3%</td>
<td>0.9%</td>
<td>100%</td>
</tr>
<tr>
<td>Expressions</td>
<td>3.17</td>
<td>3.7%</td>
<td>14.7%</td>
<td>48.6%</td>
<td>26.6%</td>
<td>6.4%</td>
<td>0.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Functions</td>
<td>3.04</td>
<td>3.6%</td>
<td>19.6%</td>
<td>43.8%</td>
<td>22.3%</td>
<td>8.0%</td>
<td>0.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Input / output statements</td>
<td>3.64</td>
<td>1.8%</td>
<td>8.8%</td>
<td>37.2%</td>
<td>28.3%</td>
<td>23.9%</td>
<td>0.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Operators (+,-,*,/!,&amp;&amp;,</td>
<td></td>
<td>)</td>
<td>3.45</td>
<td>1.8%</td>
<td>16.1%</td>
<td>37.5%</td>
<td>25.0%</td>
<td>19.6%</td>
</tr>
<tr>
<td>Parameters</td>
<td>3.28</td>
<td>4.8%</td>
<td>18.1%</td>
<td>37.1%</td>
<td>24.8%</td>
<td>15.2%</td>
<td>0.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Primitive Data Types</td>
<td>3.27</td>
<td>1.8%</td>
<td>15.0%</td>
<td>42.5%</td>
<td>26.5%</td>
<td>12.4%</td>
<td>1.8%</td>
<td>100%</td>
</tr>
<tr>
<td>Repetition Structures (loops)</td>
<td>2.98</td>
<td>4.5%</td>
<td>27.9%</td>
<td>34.2%</td>
<td>22.5%</td>
<td>9.0%</td>
<td>1.8%</td>
<td>100%</td>
</tr>
<tr>
<td>Recursion</td>
<td>3.10</td>
<td>0.0%</td>
<td>21.3%</td>
<td>55.6%</td>
<td>14.8%</td>
<td>8.3%</td>
<td>0.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Selection Structure (if statement)</td>
<td>3.21</td>
<td>5.4%</td>
<td>17.1%</td>
<td>42.3%</td>
<td>21.6%</td>
<td>13.5%</td>
<td>0.0%</td>
<td>100%</td>
</tr>
<tr>
<td>Variables declaration</td>
<td>3.05</td>
<td>2.8%</td>
<td>22.9%</td>
<td>46.8%</td>
<td>17.4%</td>
<td>9.2%</td>
<td>0.0%</td>
<td>100%</td>
</tr>
</tbody>
</table>
The mean values of the treatment group are positively improved for all teaching topics compared to the control group, the treatment group finding it generally easier to learn the specific topics. The results of Mann Whitney U test and hypothesis testing (null and alternative hypotheses) for the set of teaching topics questions are depicted in Table 5.6.

Again, table 5.6 reveals there are significant differences in some areas of teaching topics such as arrays (p=.002), expressions (p=.010), functions (p=.002), operators (p=.013), primitive data types (p=.002), repetition structures loops (p=.000), and selection structure if statement (p=.025) at the level of p<0.05. On the other hand, there are some areas where there are insignificant differences such as error handling techniques (p=.677>0.05), input/output statements (p=.228>0.05), parameters (p=.057>0.05), recursion (p=.086>0.05), and variables declaration (p=.122>0.05).
### Table 5.6: Mann-Whitney U test for teaching topics

<table>
<thead>
<tr>
<th>Group</th>
<th>Control</th>
<th>Treatment</th>
<th>Total</th>
<th>N</th>
<th>Mean</th>
<th>Rank</th>
<th>Sum of Ranks</th>
<th>Mann-Whitney U</th>
<th>Asymp. Sig. (2-tailed)</th>
<th>Hypothesis testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrays</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>98</td>
<td>112</td>
<td>210</td>
<td></td>
<td>92.18</td>
<td>117.15</td>
<td>9034.00</td>
<td>4183.00</td>
<td>.002</td>
<td>$H_1$ accepted</td>
</tr>
<tr>
<td>Treatment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>117.15</td>
<td>13121.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error handling techniques</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>99</td>
<td>117</td>
<td>216</td>
<td></td>
<td>106.69</td>
<td>110.03</td>
<td>10562.50</td>
<td>5612.50</td>
<td>.677</td>
<td>$H_1$ rejected</td>
</tr>
<tr>
<td>Treatment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>110.03</td>
<td>12873.50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expressions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>99</td>
<td>109</td>
<td>208</td>
<td></td>
<td>93.93</td>
<td>114.10</td>
<td>9299.00</td>
<td>4349.00</td>
<td>.010</td>
<td>$H_1$ accepted</td>
</tr>
<tr>
<td>Treatment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>114.10</td>
<td>12437.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>98</td>
<td>112</td>
<td>210</td>
<td></td>
<td>91.71</td>
<td>117.57</td>
<td>9438.00</td>
<td>4488.00</td>
<td>.013</td>
<td>$H_1$ accepted</td>
</tr>
<tr>
<td>Treatment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>117.57</td>
<td>13167.50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input / output statements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>98</td>
<td>113</td>
<td>211</td>
<td></td>
<td>100.78</td>
<td>110.53</td>
<td>9876.00</td>
<td>5025.00</td>
<td>.228</td>
<td>$H_1$ rejected</td>
</tr>
<tr>
<td>Treatment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>110.53</td>
<td>12490.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operators</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>99</td>
<td>112</td>
<td>211</td>
<td></td>
<td>95.33</td>
<td>115.43</td>
<td>9438.00</td>
<td>4488.00</td>
<td>.013</td>
<td>$H_1$ accepted</td>
</tr>
<tr>
<td>Treatment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>115.43</td>
<td>12928.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameters</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>97</td>
<td>105</td>
<td>202</td>
<td></td>
<td>93.64</td>
<td>108.76</td>
<td>9083.50</td>
<td>4330.50</td>
<td>.057</td>
<td>$H_1$ rejected</td>
</tr>
<tr>
<td>Treatment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>108.76</td>
<td>11419.50</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primitive Data Types</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>98</td>
<td>113</td>
<td>211</td>
<td></td>
<td>92.57</td>
<td>117.65</td>
<td>9072.00</td>
<td>4221.00</td>
<td>.002</td>
<td>$H_1$ accepted</td>
</tr>
<tr>
<td>Treatment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>117.65</td>
<td>13294.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Selection Structures (If statement)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>95</td>
<td>111</td>
<td>206</td>
<td></td>
<td>93.81</td>
<td>111.79</td>
<td>8912.00</td>
<td>4352.00</td>
<td>.025</td>
<td>$H_1$ accepted</td>
</tr>
<tr>
<td>Treatment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>111.79</td>
<td>12409.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variables declaration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>96</td>
<td>109</td>
<td>205</td>
<td></td>
<td>96.64</td>
<td>108.61</td>
<td>9277.00</td>
<td>4621.00</td>
<td>.122</td>
<td>$H_1$ rejected</td>
</tr>
<tr>
<td>Treatment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>108.61</td>
<td>11838.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.5.1.4 Learning Programming

The next part of the second survey focuses on different learning situations for the introductory programming course. Table 5.7 shows the mean values and frequencies of responses for the set of learning situations questions.

<table>
<thead>
<tr>
<th>LEARNING PROGRAMMING</th>
<th>Treatment Group (Second Survey)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I learnt about programming …</td>
<td></td>
</tr>
<tr>
<td>In lectures</td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>3.98</td>
</tr>
<tr>
<td>Never (%/N)</td>
<td>2.8%</td>
</tr>
<tr>
<td>Rarely (%/N)</td>
<td>7.4%</td>
</tr>
<tr>
<td>Sometimes (%/N)</td>
<td>21.3%</td>
</tr>
<tr>
<td>very often (%/N)</td>
<td>25.9%</td>
</tr>
<tr>
<td>always (%/N)</td>
<td>42.6%</td>
</tr>
<tr>
<td>not applicable (%/N)</td>
<td>0.0%</td>
</tr>
<tr>
<td>Total %</td>
<td>100%</td>
</tr>
<tr>
<td>N</td>
<td>108</td>
</tr>
<tr>
<td>In lab sessions</td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>3.48</td>
</tr>
<tr>
<td>Never (%/N)</td>
<td>9.8%</td>
</tr>
<tr>
<td>Rarely (%/N)</td>
<td>12.7%</td>
</tr>
<tr>
<td>Sometimes (%/N)</td>
<td>20.6%</td>
</tr>
<tr>
<td>very often (%/N)</td>
<td>23.5%</td>
</tr>
<tr>
<td>always (%/N)</td>
<td>31.4%</td>
</tr>
<tr>
<td>not applicable (%/N)</td>
<td>2.0%</td>
</tr>
<tr>
<td>Total %</td>
<td>100%</td>
</tr>
<tr>
<td>N</td>
<td>102</td>
</tr>
<tr>
<td>While studying alone</td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>3.53</td>
</tr>
<tr>
<td>Never (%/N)</td>
<td>3.9%</td>
</tr>
<tr>
<td>Rarely (%/N)</td>
<td>11.8%</td>
</tr>
<tr>
<td>Sometimes (%/N)</td>
<td>36.3%</td>
</tr>
<tr>
<td>very often (%/N)</td>
<td>18.6%</td>
</tr>
<tr>
<td>always (%/N)</td>
<td>28.4%</td>
</tr>
<tr>
<td>not applicable (%/N)</td>
<td>1.0%</td>
</tr>
<tr>
<td>Total %</td>
<td>100%</td>
</tr>
<tr>
<td>N</td>
<td>102%</td>
</tr>
<tr>
<td>While working alone on programming coursework</td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>3.48</td>
</tr>
<tr>
<td>Never (%/N)</td>
<td>5.9%</td>
</tr>
<tr>
<td>Rarely (%/N)</td>
<td>13.7%</td>
</tr>
<tr>
<td>Sometimes (%/N)</td>
<td>27.5%</td>
</tr>
<tr>
<td>very often (%/N)</td>
<td>27.5%</td>
</tr>
<tr>
<td>always (%/N)</td>
<td>24.5%</td>
</tr>
<tr>
<td>not applicable (%/N)</td>
<td>1.0%</td>
</tr>
<tr>
<td>Total %</td>
<td>100%</td>
</tr>
<tr>
<td>N</td>
<td>102</td>
</tr>
<tr>
<td>In exercise sessions in small groups</td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>3.70</td>
</tr>
<tr>
<td>Never (%/N)</td>
<td>4.7%</td>
</tr>
<tr>
<td>Rarely (%/N)</td>
<td>14.0%</td>
</tr>
<tr>
<td>Sometimes (%/N)</td>
<td>19.6%</td>
</tr>
<tr>
<td>very often (%/N)</td>
<td>29.9%</td>
</tr>
<tr>
<td>always (%/N)</td>
<td>31.8%</td>
</tr>
<tr>
<td>not applicable (%/N)</td>
<td>0.0%</td>
</tr>
<tr>
<td>Total %</td>
<td>100%</td>
</tr>
<tr>
<td>N</td>
<td>107</td>
</tr>
</tbody>
</table>

The values of frequencies for the set of learning situations questions belonging to the treatment group are improved in all of the questions except ‘in lab session’ where it is decreased by 0.19% (see Table 4.3 for control group). The comparison of mean values of the set of learning situations questions for the control and treatment groups is depicted in figure 5.8.
The results of Mann Whitney U test and hypothesis testing (null and alternative hypotheses) for the set of teaching topics questions are depicted in Table 5.8, which reveals there are no significant differences in the improvement in learning situations. Although there are marginal increases in improvement across the majority of the learning situations, it is interesting to note that there is a reduction in the satisfaction with the ADRI approach ‘in lab sessions’.

![Figure 5.8: Mean comparison of learning situations](image-url)
### Table 5.8: Mann-Whitney U test for learning situations

<table>
<thead>
<tr>
<th>Group</th>
<th>N</th>
<th>Mean Rank</th>
<th>Sum of Ranks</th>
<th>Mann-Whitney U</th>
<th>Asymp. Sig. (2-tailed)</th>
<th>Hypothesis testing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>In Lectures</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>96</td>
<td>103.71</td>
<td>9956.50</td>
<td>5067.50</td>
<td>.769</td>
<td>$p = .769 &gt; 0.05$</td>
</tr>
<tr>
<td>Treatment</td>
<td>108</td>
<td>101.42</td>
<td>10953.50</td>
<td></td>
<td></td>
<td>$H_1$ rejected</td>
</tr>
<tr>
<td>Total</td>
<td>204</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>In lab sessions</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>95</td>
<td>105.37</td>
<td>10010.50</td>
<td></td>
<td>.115</td>
<td>$p = .115 &gt; 0.05$</td>
</tr>
<tr>
<td>Treatment</td>
<td>102</td>
<td>93.06</td>
<td>9492.50</td>
<td></td>
<td></td>
<td>$H_1$ rejected</td>
</tr>
<tr>
<td>Total</td>
<td>197</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>while studying alone</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>92</td>
<td>90.96</td>
<td>8368.50</td>
<td></td>
<td>.110</td>
<td>$p = .110 &gt; 0.05$</td>
</tr>
<tr>
<td>Treatment</td>
<td>102</td>
<td>103.40</td>
<td>10546.50</td>
<td></td>
<td></td>
<td>$H_1$ rejected</td>
</tr>
<tr>
<td>Total</td>
<td>194</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>while working alone on programming coursework</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>91</td>
<td>95.14</td>
<td>8657.50</td>
<td></td>
<td>.652</td>
<td>$p = .652 &gt; 0.05$</td>
</tr>
<tr>
<td>Treatment</td>
<td>102</td>
<td>98.66</td>
<td>10063.50</td>
<td></td>
<td></td>
<td>$H_1$ rejected</td>
</tr>
<tr>
<td>Total</td>
<td>193</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>In exercise sessions in small groups</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Control</td>
<td>96</td>
<td>93.78</td>
<td>9003.00</td>
<td></td>
<td>.052</td>
<td>$p = .052 &gt; 0.05$</td>
</tr>
<tr>
<td>Treatment</td>
<td>107</td>
<td>109.37</td>
<td>11703.00</td>
<td></td>
<td></td>
<td>$H_1$ rejected</td>
</tr>
<tr>
<td>Total</td>
<td>203</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 5.5.1.5 Resources

Table 5.9 depicts the frequencies and mean values of the set of resources questions for the treatment group. The frequencies of the treatment group are improved positively compared to the control group (see Table 4.4 for control group) for the set of resources questions.
<table>
<thead>
<tr>
<th>RESOURCES</th>
<th>Mean (%/N)</th>
<th>Useless (%/N)</th>
<th>Not very useful (%/N)</th>
<th>Somewhat useful (%/N)</th>
<th>Useful (%/N)</th>
<th>Very useful (%/N)</th>
<th>Not applicable (%/N)</th>
<th>Total %</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introductory Programming course book</td>
<td>3.43</td>
<td>7.4%</td>
<td>13.0%</td>
<td>26.9%</td>
<td>30.6%</td>
<td>21.3%</td>
<td>0.9%</td>
<td>100%</td>
<td>108</td>
</tr>
<tr>
<td>Lecture notes</td>
<td>3.98</td>
<td>3.8%</td>
<td>4.8%</td>
<td>18.3%</td>
<td>35.6%</td>
<td>37.5%</td>
<td>0.0%</td>
<td>100%</td>
<td>104</td>
</tr>
<tr>
<td>Exercise questions and answers</td>
<td>4.36</td>
<td>0.0%</td>
<td>0.0%</td>
<td>18.3%</td>
<td>27.9%</td>
<td>53.8%</td>
<td>0.0%</td>
<td>100%</td>
<td>104</td>
</tr>
<tr>
<td>Example programs</td>
<td>4.04</td>
<td>1.0%</td>
<td>4.8%</td>
<td>19.2%</td>
<td>34.6%</td>
<td>39.4%</td>
<td>1.0%</td>
<td>100%</td>
<td>104</td>
</tr>
<tr>
<td>Still pictures of programming structures</td>
<td>3.91</td>
<td>2.8%</td>
<td>5.6%</td>
<td>26.2%</td>
<td>29.0%</td>
<td>36.4%</td>
<td>0.0%</td>
<td>100%</td>
<td>107</td>
</tr>
<tr>
<td>Interactive visualisations</td>
<td>4.02</td>
<td>1.9%</td>
<td>4.7%</td>
<td>17.8%</td>
<td>31.8%</td>
<td>42.1%</td>
<td>1.9%</td>
<td>100%</td>
<td>107</td>
</tr>
</tbody>
</table>

The mean values of the treatment group are improved positively for all resources questions compared to the control group as shown in figure 5.9.
The results of the Mann-Whitney U test for the set of resources questions are shown in Table 5.10.

Table 5.10 reveals there are significant differences in relation to most resources including introductory programming course book (p=0.007), lecture notes (p=0.036), exercise questions and answers (p=0.001), still pictures of programming structures (p=0.008), and interactive visualisations (p=0.000) at the level of p<0.05. On the other hand, there is one question (example program (p=0.085)) where there is an insignificant difference between control and treatment groups.
### Table 5.10: Mann-Whitney U test for resources

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Group</th>
<th>N</th>
<th>Mean Rank</th>
<th>Sum of Ranks</th>
<th>Mann-Whitney U</th>
<th>Asymp. Sig. (2-tailed)</th>
<th>Hypothesis testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introductory programming course book</td>
<td>Control</td>
<td>96</td>
<td>91.10</td>
<td>8745.50</td>
<td>4089.500</td>
<td>.007</td>
<td>( p = .007 &lt; 0.05 ) ( H_1 ) accepted</td>
</tr>
<tr>
<td></td>
<td>Treatment</td>
<td>108</td>
<td>112.63</td>
<td>12164.50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>204</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lecture note</td>
<td>Control</td>
<td>95</td>
<td>91.47</td>
<td>8689.50</td>
<td>4129.500</td>
<td>.036</td>
<td>( p = .036 &lt; 0.05 ) ( H_1 ) accepted</td>
</tr>
<tr>
<td></td>
<td>Treatment</td>
<td>104</td>
<td>107.79</td>
<td>11210.50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>199</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exercise questions and answers</td>
<td>Control</td>
<td>93</td>
<td>85.14</td>
<td>7918.00</td>
<td>3547.000</td>
<td>.001</td>
<td>( p = .001 &lt; 0.05 ) ( H_1 ) accepted</td>
</tr>
<tr>
<td></td>
<td>Treatment</td>
<td>104</td>
<td>111.39</td>
<td>11585.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>197</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example programs</td>
<td>Control</td>
<td>96</td>
<td>93.50</td>
<td>8976.00</td>
<td>4320.000</td>
<td>.085</td>
<td>( p = .085 &gt; 0.05 ) ( H_1 ) rejected</td>
</tr>
<tr>
<td></td>
<td>Treatment</td>
<td>104</td>
<td>106.96</td>
<td>11124.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Still pictures of programming structures</td>
<td>Control</td>
<td>96</td>
<td>90.83</td>
<td>8720.00</td>
<td>4064.000</td>
<td>.008</td>
<td>( p = .008 &lt; 0.05 ) ( H_1 ) accepted</td>
</tr>
<tr>
<td></td>
<td>Treatment</td>
<td>107</td>
<td>112.02</td>
<td>11986.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>203</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interactive visualisations</td>
<td>Control</td>
<td>96</td>
<td>84.70</td>
<td>8131.50</td>
<td>3475.500</td>
<td>.000</td>
<td>( p = .000 &lt; 0.05 ) ( H_1 ) accepted</td>
</tr>
<tr>
<td></td>
<td>Treatment</td>
<td>107</td>
<td>117.52</td>
<td>12574.50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>203</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 5.5.2 Focus Group with Introductory Programming Instructors

A focus group was conducted with the IP course instructors who taught in semester 1, 2014-15 to seek in-depth feedback on the affordances and barriers of the ADRI approach for students. The focus group consists of five participants including the moderator and assistant moderator. The focus group lasted for 90 minutes and was not audio-recorded. The moderator and assistant moderator took notes, anonymously and independently, of the discussion during the session. They compared their notes after the session and produced a joint session report. This report was sent to the participants to comment on the accuracy of the reporting. Since participants’ responses were noted anonymously, it was
not possible for participants to withdraw their statements after the focus group had finished although they could suggest amendments to the record of the focus group in the report.

5.5.2.1 Focus Group Objectives
As discussed earlier, this research was based on the didactic triangle and instructors were one of the three entities in it. They taught contents to the students who were the other two entities in the didactic triangle. In our context, instructors had a close interaction with the ADRI approach and the students. So it was worthwhile to obtain their feedback on affordances and barriers of the approach for students to improve the whole learning process. This determined the objectives of the focus group. Specifically, the objectives were to explore instructors’ perceptions of:

1. Students’ experiences with the ADRI approach in the introductory programming course;
2. The impact of the ADRI approach on students’ learning in introductory programming course;

and to explore:

3. instructors’ views on the strengths and weaknesses of the ADRI approach compared to traditional teaching approaches;

and to suggest:

4. any further enhancements to the ADRI approach in the context of the introductory programming courses.
5.5.2.2 Conducting the focus group
The moderator and assistant moderator welcomed the participants. At the beginning of the session, the moderator reminded participants of the purpose of the focus group and set the ground rules for participant conduct during the session, including respecting the ideas and opinions of others, listening and responding to positive and negative remarks, and reminding participants that discussions during the session should remain confidential. The moderator started by asking the participants for their feedback on students experiences using the ADRI approach in the classroom and then for their views of the approach. The moderator encouraged all participants to actively participate and ensured that each had a chance to speak. During the session, the moderator promoted debate by asking open-ended questions. The moderator and assistant moderator took notes independently of the discussions which took place during the session. At the end of the focus group session, the moderator thanked all the participants.

5.5.2.3 Focus group themes
The objectives of the focus group were discussed in section 5.5.2.1, which were mainly focused on instructors’ feedback regarding the ADRI approach in the teaching and learning process of the introductory programming. The themes of the focus group were derived according to these objectives.

The main themes of the focus group were to discuss the following:

**Experience in teaching introductory programming course**
All the participants had a significant amount of teaching experience in introductory programming ranging from 3 to 14 years. All the participants also had considerable teaching experience at Buraimi University College (the site of the study) of between 3 to
8 years. The participants have used C++, Java and visual basic languages to teach IP courses during their academic careers. Currently, the curriculum requires them to teach C++ in the IP course. They were agreed that introductory programming is a challenging course for novices in their first semester of study. They pointed out that course material, teaching styles, language barriers, and transitioning from school to university all contribute to the challenge of learning programming and are the main factors for the high failure rates experienced in such courses. Traditional teaching methods are not successful in overcoming the barriers for students studying introductory programming.

First impression about ADRI based approach

The participants’ first impression of the programming was relatively unanimous. They agreed that programming appears very weird for the novice students. It is always harder for an instructor to teach programming language to the students who are completely unaware of the programming world.

In the context of our institution, the students found it difficult to digest the overload of the ADRI approach. The participants used different methods to make students aware of the ADRI; for example, some of them first discussed the importance of all four phases of ADRI approach to the students; This method was successful to a large extent.

The participants found ADRI a more sophisticated way of helping the students understand programming knowledge. At first it looked complex for the participants, and the implementation was perceived as a challenging task for them. Accepting the implementation challenges, the participants found it very interesting, comprehensive and a professional methodology of teaching.
Overall the feedback from the participants concluded that as instructors, the ADRI approach looked challenging, but later on it appeared to be one of the best teaching methodologies. However, from the students’ perspective, the instructors thought it was relatively difficult to get past the initial stages, but once the students fully understood it, they kept going very smoothly.

**Strengths of the ADRI approach**

The participants compared the ADRI approach with the traditional approach to summarise the strengths of ADRI approach.

The participants all agreed that ADRI is better than the traditional approach. The traditional approaches focus on all the phases separately, while ADRI integrates all the phases to make it easy for the students to understand the different aspects of problem, solution, testing and improvement together. It is easy for a student, to collectively study all the phases. The student understands the problem first with the help of pseudo-code and flowchart. Once the student understands the problem they can move toward the solution of the problem in the form of programming code. The participants agreed that the last two phases of the ADRI approach further enhance the students’ level of logic understanding. The student provides the stated input to the program and then observes the output. Furthermore, it shows the process involved in solving the problem statement. It improves the understanding of the student and the student realises in reality the reasons behind writing the program. The improvement phase, helps in increasing students understand of the solved problem. It helps him/her in developing the logic and in fact helps student to understand the exact meaning and purpose of logic.
Further regarding the structure of ADRI, the participants discussed and agreed that the ADRI approach encourages students to undertake (or think about) program testing. The participants observed that the students tested their programs with values other than those provided in the ADRI exercises. The participants, in such cases, drew their attention to test special values in their program and then see the results. As an example, the students were asked to test the result of a program which divides an integer by another by keeping one integer value to zero. The students found themselves involved in testing and it made it easy for the instructor to explain to them the program testing process.

The participants debated and agreed that the students stopped taking shortcuts (problem statement $\rightarrow$ coding) in programming when the ADRI was introduced to them. In the majority of cases, the students wrote one program and then modified it to solve further related problems. The participants had mixed views about whether this problem has been solved to some extent with the ADRI approach.

Overall, the participants agreed that the ADRI approach provided a number of strong aspects and was helpful for both the students and the instructors.

**ADRI approach weakness**

The focus group participants pointed out some of the major weakness of the ADRI approach. The participants also agreed that these weaknesses were just time based and evaporated gradually with the passage of time and improved understanding of the students.

The participants agreed that it was relatively difficult for the students to start with the ADRI approach. The students tended to be bored quickly if they were asked to solve long
exercises. The same problem appeared in the adaptation of ADRI, the students felt bored when they were studying the four phases as one. However, the participants agreed that they educated the students about the relationship between the four phases. The majority of the participants agreed upon the fact that students gradually adopted the ADRI approach fully.

Similarly the participants also observed that their students were initially struggling to handle the complexity of ADRI. Here the complexity overcame students completed more exercises.

**Impact of ADRI approach on students learning**

The overall impact of the ADRI approach on the students learning appears positive. The students accepted the notion of implementing the ADRI approach in their following programming course.

The participants discussed the impact of ADRI over different categories of students. In the programming classes, the participants had two cohorts of students; the fresh students who were studying the subject for first time and the students who had failed it earlier and were now repeating the course. Once fully understood and adopted by all the students, the fresh students felt very comfortable and impressed with the ADRI approach. The feedback was collected from the students who were repeating the course. The participants agreed that these students felt comfortable with the new approach too and appreciated the new approach.

Similarly, the student's behavior towards doing the exercises and assignments was also improved in the most recent semester. They were comfortable and interested to carry on
with their assignments and exercises. This is because they had a better understanding of
the problem solution and were more confident with following the ADRI approach.

The participants agreed that the students obtained relatively higher grades with the ADRI
approach as compare to the previous semester results.

Summarising the general views of the participants, it was concluded that ADRI approach
had a positive impact on the students learning and the course learning objective were
better achieved as compare to the previous semester. The student's problem solving
strategies and programming knowledge were all improved. They were comfortable in
solving a programming problem using the ADRI approach phases.

*Comparison of ADRI approach with traditional teaching approaches*

The participants agreed that the traditional approach for teaching programming courses
had some inconsistencies. One of the major drawbacks of the traditional approach was the
problem of integrating all the tools together. The students were taught pseudo-code,
flowchart and then programming in a sequential manner. In such cases, the students had
to revise the previous work before going into coding. ADRI has made it simple and easy
for students to transition from one phase to the next. The students can go through all the
steps in a single page with the extra flavour of improvement.

It was also a part of the debate among the participants that unlike the traditional approach
the ADRI approach put together all the problem solving methodologies. In other words,
the ADRI approach provides a broader view of programming as compared to the
traditional approach.
Most of the participants also pointed out that it was simpler to convey the improvement phase knowledge to students. The students used to consider it as an extra piece of burden. However, the smoothness in the ADRI approach has made it simpler and easier for the students.

**Suggestions for further improvement in ADRI approach**

The focus group participants had a long discussion over possible improvements to ADRI approach. Their suggestions can be put into 3 categories; firstly, the improvement of the ADRI documentation; secondly, the syllabus changes in the institutions; and lastly, the addition of extra functionalities in the ADRI approach.

The participants discussed and then agreed that the ADRI document must be updated and an extra section of glossary should be added at the end. This glossary should explain all the terms used in the various stages, particularly around pseudo code. The glossary should be written and compiled according to the year level of the students in the institution. It should be written in plain English and be in an easy to understand format.

With regard to syllabus changes, some participants addressed the problem of credit hours allotted to the basic programming course. It was discussed and agreed that the credit hours must be increased so that students have more time to understand the basics of the programming language. This will, in part, address the students concerns, highlighted through the second survey, regarding this aspect of implementing the ADRI approach.

In the result category, the participants suggested that an extra section in related to error handling should be included. The students consider programming code like normal English text. It takes a long time for them to realise that the text of programming code is
extremely sensitive and the removal of a single semicolon, for example, can lead to many errors in the program. A teacher spends extra time in every class to explain the common errors such as missing semicolons, case sensitive nature, spelling mistakes and the impact of such errors on the overall program. The participants suggested that it would be a good idea to add an extra section in the ADRI model that addresses the common errors made by the students and that there should be an extra section for error-handling activity. The activity would ask the students to make a mistake for example "remove semi-colon at the end of line # 8" or "change the word main to Main" etc. The students’ exercise is to make the mistake and then record its impact on the program. The instructor’s responsibility is then to explain the impact of the error, the technique of finding and fixing the error and to make students careful to avoid similar mistakes in the future.

Other remarks and feedback

The participants appreciated the ADRI-based editor and agreed that this has a positive impact on the students’ learning. The editor provides all the necessary design features (pseudo-code, flowchart) and programming tools in a single application. The students can easily navigate from phase to phase. It is easy to use and the students did not need a lot of training or tutorials. It is also interesting to use and the students enjoyed working in the editor. The menus are self-explanatory and easy to use and navigate. According to the participants, in the presence of this editor the students' attitude towards programming and problem solving is more professional and practical.

One of the major properties of the editor is that it is platform independent and can be used in any system and in any operating system. It does not need complex installation guides and the students can easily install and run it.
Focus group participants suggested a number of improvements to the editor that would assist students. The steps - how to save and how to open a program - should be included in the Help menu. It will facilitate users to work and interact with the editor. It will also support new users of the editor.

It was also suggested to add more supporting features in the editor and a number of bugs were identified. For example the Next and Previous sub-menus were not working properly under some circumstances. If you first use the Next sub-menu then Previous sub-menu did not work and vice versa.

The participants, overall, had a positive view of the ADRI approach and they suggested it for further programming courses. Similarly, the instructors suggested that the students felt comfortable and satisfied with the approach and the feedback from their classes indicated that its use should be continued in the future.

5.6 Discussion

The impact of the ADRI approach on the teaching and learning process of the introductory programming course is positive after comparing the results of the first and second surveys. The first survey was conducted with those students (control group) who finished the IP course with the traditional approach, while the second survey was conducted with those students (treatment group) who finished the course with the ADRI approach. The same set of questions was asked in both surveys. The surveys captured information about student demographics, students perceptions of the ease with which they learned various programming concepts and finally questions related to the learning
situations and teaching materials of the course. The programming concepts mainly covered questions related to the curriculum of the IP course.

5.6.1 Survey outcomes
The mean comparison of both surveys for the programming concepts questions depicts positive improvement in all questions. The ADRI approach helped the novices improve their learning of programming concepts. It not only focuses on the programming syntax but it also emphasises the problem solving skills. Therefore, there is positive improvement in responses of the novices in both areas of the general programming concepts and teaching topics questions.

The ADRI approach provides an opportunity for the novices to develop problem solving skills by writing a pseudo-code solution and drawing a flowchart for all the questions included in the course. Moreover, it also helps them gain an improved understanding of a problem statement. The ADRI approach focuses on the programming syntax in the Deployment and Improvement stages. The novices gain a better understanding of the programming syntax and constructs by practicing them in two stages. They convert the pseudo-code or flowchart into a computer program in the Deployment stage. Furthermore, they practice more programming constructs by solving slightly different programming statements in the Improvement stage.

The outcomes of the ADRI approach improved significantly the three areas ‘Understanding problem solving strategies’, ‘Understanding programming structures’ and ‘Learning the programming language syntax’. These three programming concepts were perceived most difficult learning areas by the control group participants. The ADRI approach focuses equally on programming solving strategies and programming
knowledge (syntax and semantics). Novices have to devise a solution of a problem statement by using two different problem solving strategies such as flowchart and pseudo-code. Likewise they practice different programming structures in Deployment and Improvement stages of the ADRI model which also helps them to acquire in-depth knowledge of the syntax and semantics of the programming language. This shows that the ADRI approach gives equal emphasis to the whole set of programming skills which covers broad spectrum of a programming domain.

I introduced the ADRI editor to help novices to understand and practice exercise questions. The editor provides separate views for all four stages and the exercise questions are embedded in it. The significant difference in ‘Using the program development environment’ could be attributed to the inclusion of the editor. On the other hand, the editor does not provide a simple process for compiling and executing a program. A user has to type the commands in the DOS prompt. This could have contributed to the small but insignificant improvement in ‘Compiling and executing programs’ question.

The ‘Dividing functionality into procedures’ and ‘Designing a program to solve a certain task’ questions also showed small but insignificant improvements. The result for ‘Dividing functionality into procedures’ is marginal \((p = .050)\) and more attention will be given in the future to improve students experience in ‘Designing a program to solve a certain task’ area by adding more examples and exercises in functions and related topics.

The Result stage of the ADRI approach demonstrates the procedure used in the program to deal with input, output and intermediate processes for solving the problem statement. It
helps novices to understand the problem domain. Moreover, it gives information about
the expected input and output required for the problem statement. Overall, it helps the
novices to achieve deep learning of the programming concepts. The inclusion of
information related to expected output and intermediate processes appears to have led to
the improvement in the students’ perception of ease of learning of programming concepts
and teaching topics.

The ‘error handling techniques’ question shows insignificant improvement since the
ADRI approach did not support it initially. After receiving feedback from the instructors
and students, I introduced the common programming errors section in the Result stage of
the improved ADRI approach (see section 6.2.1). It covers three types of programming
errors such as syntax errors, semantic errors and syntax warnings. Some other advanced
topics in programming such as recursion and parameters were also reported as achieving
insignificant improvement and more attention should be given to those areas in the
teaching process.

In the teaching topic section of the survey, the control group perceived arrays, functions,
and repetition structures (loops) as the most significantly difficult topics in programming.
The outcomes of the ADRI approach showed significant improvement in all these three
areas. The Deployment and Improvement stages of the ADRI model deal with the syntax
and semantics of the programming language. Developing a Flowchart and pseudo-code
for a given problem statement also helped students to have a more precise understanding
of these programming constructs.
One of the reasons for better results of the ADRI approach compared to the traditional approach is to allow novices to develop a better understanding of the problem domain. The Approach stage gives them a better understanding of the problem statement without having to worrying about syntax and semantics of the programming language. They focus only on a solution of the problem statement by writing pseudo-code and drawing the flowchart in the Approach stage. Once they devise the solution to the problem statement, the Deployment stage requires them to convert the solution into a computer program by using a programming language. Moreover, the Improvement stage helps novices in practicing different programming constructs by slightly changing the problem statement. In this way, the ADRI approach emphasises not only the syntax and semantics but also the problem solving strategies. Consequently, there is significant improvement in ‘Understanding programming structures’, ‘Learning the programming syntax’, ‘Understanding problem solving strategies’ and in the related teaching topics.

There are insignificant improvements in all the five questions of the learning situations section. Analysis of the responses of both surveys regarding this section in figure 5.8, shows that participants from both groups are satisfied with all the five learning situations. The mean values of all the five learning situations for both groups are pointing towards the positive side.

The treatment group shows better results compared to the control group in all the teaching topics questions. The ADRI approach gives them a deeper understanding of the programming constructs than the traditional approach. It shows them the flow of logic in the programming constructs by drawing the flowchart and practicing it in a programming language. The flowchart shows them another way of describing the underlying process
such as diagrammatic flow of information in loops, functions and arrays etc. It helps them in developing deep learning of the programming concepts instead of surface learning. As a result, there is significant improvement in the teaching topics questions.

The ‘Example program’ question depicts insignificant improvement. Students become bored quickly if they are asked to solve long exercises. The same problem appeared in the adaptation of ADRI, the students lost motivation when they were studying the four phases as one. As discussed earlier, the ADRI approach discourages the programming shortcut (Problem Statement→Coding) because that focuses only on the programming knowledge (syntax and semantics). Whereas, the four stages of the ADRI approach focus on both problem solving strategies and programming knowledge which are essential skills for the novices in introductory programming. Secondly, the ADRI approach helps the novices understand that programming consists of several tasks such as program design, language features and testing which are not emphasised in the traditional approach.

One of the challenges for the novices in the traditional approach is to understand the hidden processes going on in the memory of the computer during the compilation of the program. Mostly, they do not understand the flow of information in the program and this result in poor understanding of the problem domain. The ADRI approach shows them the underlying process used to solve the problem statement in the Result stage. It demonstrates the input, the logic to solve the problem statement and the output of the problem statement which gives them a clearer and better understanding of the problem domain. Therefore, there is significant improvement in the teaching topics questions of the treatment group compared to the control group.
The treatment group shows positive improvement in all the learning situations questions compared to the control group except ‘in lab session’ question. The decline (-0.19) in students perceptions of ease of learning ‘in lab session’ could be related to the amount of time it takes to complete the exercises using the ADRI approach. The four stages require more time and effort to complete. The current contact hours (3 hours per week) are not sufficient to complete these tasks in the lab sessions. Therefore, the students might not be satisfied with the current contact hours, as reflected in the ‘in lab session’ responses. It could be improved by increasing the contact hours to 4 or 5 hours per week so that the students will get more time to practice the programming problems. Students could be further motivated to complete the exercises by allocating marks for each lab session or exercise.

There is a positive improvement in all the resources questions of the treatment group compared to the control group. The updated teaching materials demonstrate the programming examples and problems based on the four stages of the ADRI approach which promotes program design, language features, and program comprehension. The glossary for the general terms also may enhance their understanding of the programming expressions and help overcome English language barriers. The ADRI editor increases their interaction with the four stages of the ADRI approach. As a result, the treatment group shows more satisfaction than the control group in all the resources questions.

### 5.6.2 Focus group

All the participants agreed that the ADRI approach is better than the traditional approach suggesting that the ADRI approach four stages provide all the basic skills (problems solving strategies and programming knowledge) required by the novices in
comprehending the programming concepts. On the other hand, the participants suggested that the traditional approach covers all these skills at different stages of their learning process but separately. Consequently, it is difficult for the novices to utilise these skills together in a programming context. The participants agreed that the ADRI approach gives them enough practice and exposure to all these skills together so it is easy for them to apply it in the programming context.

The participants also appreciated that the ADRI approach discourages students taking programming shortcuts (problem statement → coding). In the traditional approach, the novices convert the problem statement into a computer program directly so they focus more on the syntax and semantics of a programming language. On the other hand, the four stages of the ADRI approach first emphasis problem solving strategies without much worry about syntax and semantics of the programming language.

The participants agreed that the ADRI approach encourages students to undertake (or think about) program testing. The Result stage shows them the required input and output for the given problem statement. It helps the novices to understand the purpose of the given problem statement. Moreover, it promotes the habit of testing their programs with different sets of inputs and checking the functionality and reliability of their proposed solution for the given problem statement. The participants agree that it also supports the deep learning of the problem domain.

The participants agreed that the students become bored quickly if they are asked to solve long exercises. The same problem appeared in the adaptation of ADRI, the students lost interest when they were studying the four phases as one. The graded lab sheets can be
introduced to develop the students’ interest and motivation for completing the longer exercises. Moreover, the contact hours per week for the IP course can be increased from 3 to 5 which should help the novices in completing the more challenging exercises.

The participants also observed that their students were struggling to handle the complexity of ADRI which comes from the integration of four phases. The concept of the ADRI approach is somewhat difficult for the novices at the beginning but it gives them a broader sense of the problem, deep learning and understanding of the programming field.

The participants discussed that overall the students were satisfied with the ADRI approach. It was also reflected in their assignments and exercises in that semester. The students understood the different ways (pseudo-code, flowchart, and program) to articulate the solution for the assignments and exercises questions.

The participants discussed that in the traditional approach, the students were taught pseudo-code, flowchart and then programming sequentially. The participants noted that in such cases, the students had to revise the previous work before going into coding. On the other hand, the ADRI approach discusses all the steps in each question. In the traditional approach, it is difficult and confusing for the students to understand the connection between pseudo-code, flowchart and coding because in most of the cases it is discussed separately or chronologically. In contrast, the ADRI approach develops a clear connection between these steps of the programming domain.

The participants suggested that the ADRI document must be updated and a glossary should be added to the end. The glossary for the general terms used in the programming examples and problems will be developed and included in the documentation for the next
cycle. It will be written in plain English because the learners are studying this course at level one and English is their second language.

The participants suggested for an extra section in the Result stage of the ADRI approach related to error handling. For the next cycle, the extra section will be included in the programming examples to demonstrate general programming errors. The common syntax and semantics errors will be included. It will help the students minimizing debugging time. Consequently, they will be able to focus more on the problem domain.

The participants appreciated the ADRI editor because it is simple to use, ADRI oriented and easy to run. It should enable students’ to easily access all the exercises’ topics and phases. The teachers suggested adding more supporting features available in other editors. Moreover, the steps to save and open a program should be included in the Help menu. They also pointed out to fix the minor technical problems with the Next and Previous sub-menus. For the next cycle, all these suggestions will be incorporated in the ADRI editor.

5.7 Conclusion

The ADRI approach was incorporated in the teaching materials of the introductory programming. The course was offered to the students of semester 1, 2014-15. For lectures and lab sessions, the programming examples and problems were prepared based on the four stages of ADRI approach. Moreover, the ADRI editor was designed and prepared for the ADRI approach to facilitate the learning process. It was used in lab sessions for practicing the programming problems. All the programming problems were embedded in it and the students could easily access these through the Topic and Question menus. The
5 ADRI BASED INTRODUCTORY PROGRAMMING COURSE

Topic menu lists all the programming problems based on the teaching topics of the IP course. The Question menus can be used to move forward or backward in accessing the programming problems.

A feedback was collected about the ADRI approach from two main entities (students and instructors) of the didactic triangle. The second student survey had the same questions as the first survey to determine the impact of ADRI approach on the learning process. Overall, a comparison of the responses of the first and second surveys showed that the ADRI approach had a positive impact on the learning process in the introductory programming course. Likewise, a focus group was conducted with the IP instructors to get in-depth feedback on the ADRI approach. They appreciated the ADRI approach in learning and teaching process of the course, and gave some suggestions to further improve the ADRI approach which will be incorporated in the third cycle.
6 Improved ADRI based Introductory Programming Course

6.1 Introduction

This chapter describes cycle 3 of this study, the final stage. There were four main activities performed in this cycle as shown below in figure 6.1. Firstly, the teaching materials based on the ADRI approach were improved based on the feedback received from cycle 2 (section 6.2). Secondly, the ADRI based teaching materials were analysed (section 6.3). Thirdly, the final grades for the four semesters of the introductory programming (IP) course in this study were compared and analysed (section 6.4) Lastly, the teaching materials based on the traditional and ADRI approaches were compared and analysed (section 6.5).

Figure 6.1 shows the three cycles of the action research with emphasis on the cycle 3 by highlighting it.
The outcomes of cycle 2 of this action research project provided feedback regarding the ADRI approach from two main entities (students and instructors) of the didactic triangle. The feedback was received from the students through the second survey (treatment group) and from the IP instructors by conducting a focus group. Some suggestions to improve the ADRI approach emerged from the feedback received from these sources as discussed further in section 6.2.

In cycle 3, firstly, the suggestions received from cycle 2 are discussed and then the process to incorporate the suggestions into the ADRI approach is highlighted. The improved course was offered to the IP students in semester 2, 2014-15. Secondly, the teaching materials based on the ADRI approach were analysed. Thirdly, the final grades
for the four semesters were analysed to determine the impact of the ADRI approach on students’ performance. Out of these four semesters, the IP course in two semesters (1 & 2, 2013-14) used the traditional approach and the remaining two (semesters 1 & 2, 2014-15) used the ADRI approach. Lastly, the teaching materials prepared on the basis of the traditional and ADRI approaches are compared.

In this chapter, I address the following two research questions:

\[ RQ3. \text{What is the impact of applying the ADRI approach on introductory programming course materials?} \]

\[ RQ4. \text{What is the impact of applying the ADRI approach on student learning?} \]

The following sub-questions are investigated to inform the interpretation of the results of this cycle.

\[ SQ1. \text{How can we incorporate and improve the ADRI approach based on the suggestions received from the cycle 2?} \]

\[ SQ2. \text{What is the impact of the ADRI approach compared to the traditional approach on the teaching materials of the introductory programming course?} \]

Research question 3 is explored in section 6.3. Research question 4 is investigated in section 6.4. Sub-questions 1 and 2 are explored in sections 6.2 and 6.5 respectively.

This chapter is organised into a number of sections starting with a discussion on suggestions to improve the ADRI approach (section 6.2), followed by an analysis to determine the impact of the ADRI approach on the teaching materials (section 6.3) and students’ performance (section 6.4). The comparison of the teaching materials based on
6 IMPROVED ADRI BASED INTRODUCTORY PROGRAMMING COURSE

the traditional and ADRI approaches is elaborated in section 6.5. The chapter concludes with a discussion (section 6.6) and summary (section 6.7) of the outcomes.

6.2 Suggestions for the ADRI based Introductory Programming Course

Three key suggestions emerged from cycle 2 regarding the ADRI approach. These are:

1. Addition of an extra section in the Result stage of the ADRI approach to provide information about common syntax and semantics errors in programming
2. Inclusion of a glossary for general terms used in the ADRI based materials
3. Addition of supporting features and fixing of minor technical issues identified in the ADRI editor

These three suggestions are explored in detail and their implementation and benefits for the ADRI approach are discussed.

6.2.1 Common Syntax and Semantic Errors Section in the Result Stage

The focus group participants agreed that most students make common syntax errors (such as missing semicolons, undeclared variables etc.) in their programs because they do not have a good idea about their significance. Consequently, both students and instructors spend extra time in class fixing these errors. Therefore, to improve the teaching and learning process, it is beneficial to add an extra section in the Result stage of the ADRI process to provide information about common syntax errors. Moreover, it should minimise the amount of time students spend trying to debug their programs and they can focus more on the problem domain.
The ADRI based teaching materials were updated and this new section added in the Result stage of all relevant programming topics. It was not included in the first topic because the discussion on C++ syntax starts from the second topic. The instructors discussed these common programming errors during their lectures with students studying in semester 2, 2014-15.

Millis (2002) categorises possible programming errors into three types which I included in the teaching materials. These are syntax errors, semantic errors and syntax warnings. A brief discussion and an example of each of these three types follow.

**Syntax Errors**

‘A syntax error is a violation of the syntax, or grammatical rules, of a natural language or a programming language’ (Millis, 2002; p. 5) such as a missing semicolon or undeclared variable name. Millis (2002) discussed that some syntax errors are very common among novice programmers. Moreover, a compiler does not translate a program into executable code if there are syntax errors detected in it. So it is important to increase novices’ awareness of these types of syntax errors. Therefore many of examples were included in the teaching materials.

Kummerfeld and Kay (2003) concluded that ‘Syntax error correction is the first step in the debugging process. It is not possible to continue program development until the program compiles. This means it is a crucial part of the error correction process’ (p. 109). Moreover, novice programmers face more difficulty in understanding error messages which can often appear ambiguous. Denny et al. (2012) conducted a study to analyse the syntax error messages encountered by students while writing short fragments of Java code.
using a tool called CodeWrite. They concluded that ‘specific teaching support around the causes of these errors may be particularly effective’ (p. 80). Moreover, ‘these errors consume a large amount of time, suggested that targeting teaching interventions may yield a significant payoff in terms of increasing student productivity’ (p. 75).

Table 6.1 below shows the improved example of the ADRI based teaching materials where the common programming error section in the Result stage were introduced. All the error examples consist of three parts. The name of the error with error type, an example related to this error with line numbers, reason(s) for this error and its correct solution. In this example, the common syntax error caused by incompatible data types is illustrated. The variable Second is declared as an integer and an attempt is made to assign a string value to it. The compiler will generate an incompatible data type syntax error message.
### Table 6.1: Programming example incorporated with syntax error example

<table>
<thead>
<tr>
<th>Step1: Approach – Problem solving strategies</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pseudo-code</strong></td>
<td><strong>start</strong></td>
</tr>
<tr>
<td>1. Start</td>
<td>Read Second, Minutes, Hour</td>
</tr>
<tr>
<td>2. Read Second, Minutes, Hour</td>
<td>Total_time_second = Second + Minutes *60 + Hour * 3600</td>
</tr>
<tr>
<td>3. Total_time_second = Second + Minutes *60 + Hour * 3600</td>
<td>Print Total_time_second</td>
</tr>
<tr>
<td>4. Print Total_time_second</td>
<td>Stop</td>
</tr>
<tr>
<td>5. Stop</td>
<td></td>
</tr>
</tbody>
</table>

#### Step2: Programming Knowledge

```cpp
#include <iostream>
using namespace std;

int main()
{
    int second, hour, minute, total_time_second;
    cout<< "enter the time in second. 
    cin>>second;
    cout<<endl;
    cout<< "enter the time in minutes. 
    cin>>minute;
    cout<<endl;
```

#### Step3: Result

**Expected output:** enter the time in second: 65
enter the time in minutes: 20
enter the time in hours: 2
The total time in second: 8465
**Process:** total_time_second = 65 + 20*60 + 2*3600
**Achieved output:**

**Goals:**

- Achieved: ☐
- Not Achieved: ☐

#### Syntax Error: Incompatible data types / Initialisation

**Example:**

```cpp
7    int second = “Hello”;
```

**Reason:** integer value is expected in variable second on line 7. The string value (Hello) is assigned to integer variable (second).

#### Step4: Improvement

Update above program that prompts the user to input seconds and calculate hour, minutes and remaining seconds.

**Expected output:** Enter the time in Seconds : 7502

<table>
<thead>
<tr>
<th>Hours</th>
<th>Minutes</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

### Semantic Errors

‘A semantic error is a violation of the rules of meaning of a natural language or a programming language’ (Millis, 2002; p. 15), such as array index out of bounds error,
infinite loop error etc. Semantic errors are much harder to detect and correct compared to syntax errors. Most of the time, the compiler does not generate any warnings, when there are semantic errors in a program. Even if it converts the program into executable code it does not work correctly upon running (Millis, 2002). Some examples from this category have been included in the teaching materials.

Brown and Altadmari (2014) conducted a study to evaluate the error messages of over 100,000 students collected through a blackbox data collection project. They concluded that semantic errors generally occur more frequently than the syntax errors.

Table 6.2 below shows the Result stage of one of the programs from the teaching materials used in the course with an example of semantic error. Here the array size is set to be 10 and its index will start from 0 with an upper bound of 9. When this array in the FOR loop is accessed, the index values start at 1 and go up to 10. There is no index in the array at value 10 which causes an array index out of bound semantic error.

**Table 6.2: Example of Semantic error**

<table>
<thead>
<tr>
<th>Step3: Result</th>
<th>Semantic Error: Array Index out of bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expected output:</strong> Enter 10 integer numbers: 13 2 44 32 21 10 22 33 21 43 The minimum number is 2</td>
<td><strong>Example:</strong> 10 int Array[10]; //... 13 for(int i=1; i&lt;=10; i++) { 14 cout&lt;&lt;Array[i]; }</td>
</tr>
<tr>
<td><strong>Achieved output:</strong></td>
<td><strong>Achieved:</strong> Not Achieved:</td>
</tr>
<tr>
<td><strong>Goals:</strong></td>
<td>Achieved: Not Achieved:</td>
</tr>
</tbody>
</table>
Syntax Warnings

‘A syntax warning is displayed when the compiler has found that the syntax of some part of your code is valid, but it is potentially erroneous anyway (Millis, 2002; p. 11)’ such as uninitialised variables, using = operator instead of == operator in IF statement etc. Syntax warnings are not considered as fatal errors but they are helpful in the debugging process (Millis, 2002). Some examples from this category have been included in the teaching materials.

Table 6.3 below shows an example of a syntax warning from the teaching materials. In this example, the assignment operator (=) is used instead of a comparison operator (==) in the IF statement. Here the program will always assign a value of 50 to a variable num instead of testing for equality. The compiler will issue a syntax warning about it during the compilation process. ‘A syntax warning should always be taken seriously, because there is probably a real error in your code if the compiler issues a warning message’ (Millis, 2002; p12). Students tend to ignore warnings since the compiler has successfully generated the executable code.

Table 6.3: Example of syntax warning

<table>
<thead>
<tr>
<th>Step3: Result</th>
<th>Syntax Warning: Used assignment operator instead of comparison operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expected output: Please, enter a Number: 3</td>
<td>Example:</td>
</tr>
<tr>
<td>3 is Odd number</td>
<td>10 if (num = 50)</td>
</tr>
</tbody>
</table>
| Process: R = 3 % 2 (The remainder is 1 not 0 so it is odd number) | 11 {
| Achieved output: | 12 cout << “value equals 50”;
| Goals: Achieved: □ Not Achieved: □ | 13 }
| Reason: Comparison operator (==) is expected on line 10 instead of assignment operator (=) |
6.2.2 A glossary of general terms in ADRI based teaching materials

The second suggestion made during the focus group is to include a glossary of general terms used in the ADRI based teaching materials.

Wikipedia (2014) defines a glossary as: ‘A glossary, also known as a vocabulary, or clavis, is an alphabetical list of terms in a particular domain of knowledge with the definitions for those terms’.

In the context of this study, the IP course is offered at the first level. So it is a new area of knowledge for most of the novices. They are not familiar with the programming terms. Therefore it is important to develop a glossary of programming terms and their definitions. It provides novices with a definition of a programming concept in a concise and a simple way. As mentioned earlier, English is a second language for the students participating in this research project potentially compounding difficulties with understanding. Therefore it is more effective to present programming terms in a simple way to enhance their learning process. The glossary was developed to assist students in overcoming not only programming language barriers but also English language barriers that they may encounter with the exercises.

A glossary document was prepared which covers most of the terms used in the ADRI based IP course. The terms used in the pseudo-code examples are also explained in the glossary document. A brief definition, along with examples, is given for most of the terms. These terms are shown in italic font in the ADRI based teaching materials.

Table 6.4 below shows one of the terms from the glossary document related to the pseudo-code section of the ADRI approach. In this example, the name of the term
(compute) and some commonly used equivalent terms (calculate or determine) are mentioned. The equivalent terms are common in pseudo-code terminology because there is no standard language for it. Then the definition of the term (compute) is given. In most of the cases, an example is given for each term to depict its functionality. Lastly, a brief explanation is given about the example.

<table>
<thead>
<tr>
<th>Table 6.4: Example of glossary term</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compute</strong> (other equivalent terms: Calculate, Determine)</td>
</tr>
<tr>
<td>To determine or calculate a value of a variable by mathematical means. It normally involves one or more of the operators, addition, subtraction, multiplication or division. For example,</td>
</tr>
<tr>
<td>Compute INTEREST as BALANCE * RATE</td>
</tr>
<tr>
<td>The above example calculates a value for variable INTEREST by multiplying BALANCE by RATE.</td>
</tr>
</tbody>
</table>

### 6.2.3 Supporting features in the ADRI editor

The third suggestion from the focus group was to improve the ADRI editor by including supporting features. This task was accomplished by adding three extra features in the editor. These are the addition of a Help menu, tool tips for the four stages of the ADRI process, and mnemonics and accelerators for menus and sub menu items.

The Help menu (see figure 6.2 below) is added to provide information about the steps to save the pseudo-code and C++ program. Moreover, the steps to open the saved programs are also added. In the focus group, participants were agreed that the novices do not have a good idea about different types of files such as pseudo-code and C++ so it is important to guide them in saving and opening these files. To incorporate this suggestion, the steps to
save the different types of files are included in the Help menu. A new window is opened to show steps as shown in figure 6.2.

![Help Menu](image1)

*Figure 6.2: Help menu in the ADRI editor*

Tool tips are added for all the four stages of the ADRI process as shown below in figure 6.3. Microsoft (2015) defines a tooltip as a ‘short description, usually just a few words, that appears when the user holds the mouse pointer briefly over a control or another part of the user interface without clicking’. In this study, it provides a short description of each stage of the ADRI approach to the students. For example, in figure 6.3, it shows a short description of the Deployment stage.

![Tool tip for the Deployment](image2)

*Figure 6.3: Tool tip feature for the ADRI editor*
In editors, it is common to use shortcut keys for menu items to speed up the access process. In Java, it is called ‘enabling keyboard operation’ (Oracle Java documentation, 2015) and there are two ways to accomplish it, mnemonics and accelerators. In this study, these features were incorporated to help the users to interact with the editor. ‘Mnemonics offer a way to use the keyboard to navigate the menu hierarchy, increasing the accessibility of programs. Accelerators, on the other hand, offer keyboard shortcuts to bypass navigating the menu hierarchy’ (Oracle Java documentation, 2015). Mnemonics and accelerators were added for all menus and their items as shown in figure 6.4.

A mnemonic is normally a single character and is used to access a menu if pressed with the Alt key. For example, in the editor, pressing Alt and ‘F’ (capital f) displays the File menu as shown in figure 6.4. The mnemonic letters are underlined in the menu bar and items texts. On the other hand, the accelerator is also a key combination to access menu items. For example, in the editor, pressing Ctrl and ‘N’ the New menu item from the File menu is activated, as shown in figure 6.4 (Oracle Java documentation, 2015).

![Figure 6.4: Mnemonic and accelerator features for the ADRI editor](image)
6.3 Impact of the ADRI approach on the teaching materials

The teaching materials of the IP course were prepared based on the four stages of the ADRI approach. Each stage provides a unique skill required for the novices to start programming. The teaching materials based on the ADRI approach were analysed to determine their impact on the distribution of time across problem solving strategies and syntax and semantics, as shown in table 6.5.

Table 6.5 shows the analysis of the ADRI based teaching materials against five categories: Teaching topics for lecture sessions, Practical topics for lab sessions, Types of programming examples and problems, Presentation style of example and problems, and Four problem solving steps in examples and problems.

In the first category, Teaching topics for lecture sessions, the ADRI approach pays equal attention to problem solving strategies (100%) and syntax & semantics (83.3% C++). The discussion on syntax and semantics of C++ starts from the second topic in the course; on the other hand, the discussion on problem solving strategies starts from the first topic. Therefore the percentage(s) of the teaching materials devoted to problem solving strategies is marginally higher than that devoted to syntax and semantics in the course.

Likewise, in the second category, Practical topics for lab sessions, the ADRI approach pays more attention to problem solving strategies (100%) and syntax & semantics (84%). Students practice problem solving strategies such as pseudo code and flowcharts in all questions of the lab exercises. Again, the practice questions for syntax and semantics of C++ start from lab exercises related to the second topic.
Table 6.5: analysis of the ADRI based teaching materials against five categories

<table>
<thead>
<tr>
<th>No</th>
<th>Area of concern</th>
<th>ADRI approach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Examples</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Teaching topics for lecture sessions</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Problem solving strategies (%)</td>
<td>Syntax &amp; semantics (%)</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>83.3%</td>
</tr>
<tr>
<td></td>
<td>(C++ starts from second topic)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Practical topics for lab sessions</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Problem solving strategies (%)</td>
<td>Syntax &amp; semantics (%)</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>84%</td>
</tr>
<tr>
<td></td>
<td>(C++ starts from second topic)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>56</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Types of programming examples and problems</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Types of Problems and Examples</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Math (%)</td>
<td>Syntax-oriented (%)</td>
</tr>
<tr>
<td></td>
<td>62.7%</td>
<td>9.3%</td>
</tr>
<tr>
<td></td>
<td>Daily-life (%)</td>
<td>Miscellaneous (%)</td>
</tr>
<tr>
<td></td>
<td>25.6%</td>
<td>2.4%</td>
</tr>
<tr>
<td></td>
<td>54</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Presentation style of example and problems</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Presentation Style</td>
</tr>
<tr>
<td></td>
<td>Problem Statement -&gt; Flowchart or pseudo code</td>
<td>Problem Statement -&gt; Solution Plans -&gt; Codes</td>
</tr>
<tr>
<td></td>
<td>Lecture (%)</td>
<td>Lab (%)</td>
</tr>
<tr>
<td></td>
<td>16.6%</td>
<td>16.1%</td>
</tr>
<tr>
<td></td>
<td>Lab (%)</td>
<td>Lecture (%)</td>
</tr>
<tr>
<td></td>
<td>83.3%</td>
<td>84%</td>
</tr>
<tr>
<td></td>
<td>Lab (%)</td>
<td>Lectures (%)</td>
</tr>
<tr>
<td></td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>Lab (%)</td>
<td>Lab (%)</td>
</tr>
<tr>
<td></td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Four problem solving steps in examples and problems</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Four Steps of Problem Solving</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Problem Analysis (%)</td>
<td>Solution Planning (%)</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Lab (%)</td>
<td>Lab (%)</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Lab (%)</td>
<td>Lab (%)</td>
</tr>
<tr>
<td></td>
<td>83.3%</td>
<td>84%</td>
</tr>
<tr>
<td></td>
<td>Lab (%)</td>
<td>Lab (%)</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Lab (%)</td>
<td>Lab (%)</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lab (%)</td>
<td>Lab (%)</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>Lab (%)</td>
<td>Lab (%)</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>Lab (%)</td>
<td>Lab (%)</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>Lab (%)</td>
<td>Lab (%)</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td>Lab (%)</td>
<td>Lab (%)</td>
</tr>
</tbody>
</table>

188
In the third category, *Types of programming examples and problems*, math (62.7%) and daily-life (25.6%) types are dominant. A small proportion of examples and problems belong to syntax-oriented (9.3%) and miscellaneous (2.4%) types.

In the fourth category, *Presentation style of example and problems*, most of the examples and problems are presented with Problem Statement → Solution Plans → Codes style in lectures (83.3%) and labs (84%). As mentioned earlier, the first topic in the course covers problem solving strategies and syntax and semantics of C++ starts from the second topic, all the examples and problems of the first topic are presented with Problem Statement → pseudo code and flowchart style in lectures (16.6%) and labs (16.1%). None of the examples and problems is presented with the previous traditional style of Problem Statement → Codes (0%).

In the fifth category, *Four problem solving steps in examples and problems*, the ADRI approach promotes all the four problem solving steps including: problem analysis, solution planning, coding and testing/debugging in preparing examples and problems. The problem analysis and solution planning steps are fully incorporated in all the examples and problems for lectures (100%) and lab (100%) sessions. The coding step is incorporated from the second topic in the course for all the examples and problems in lectures (83.3%) and labs (84%). The fourth step of problem solving, testing/debugging is incorporated in 100% of the examples presented in the lectures and it is incorporated in 50% of problems presented in the lab sessions which will be increased to 100% in the future.
6.4 Impact of the ADRI approach on the students’ performance

The ultimate goal of this study is to enhance students’ performance in the IP course through the introduction of the ADRI approach in the teaching and learning process. Section 1.2 mentioned the incidence of high failure and dropout rates in IP courses. In this study, to determine the impact of the ADRI approach on the students’ performance, I compared the final grades of the IP course for the four semesters of the study, two semesters with the traditional approach and the two semesters with the ADRI approach. The comparison of these semesters will indicate what impact, if any, the ADRI approach had on the students’ performance.

Table 6.6 shows the analysis of the students’ grades for the last four semesters. It presents information about the teaching approach (traditional or ADRI), total number of students in each semester, failure and dropout rates for each semester and the attrition (failure + dropout) rates for each semester. Normally we have a higher number of students in semester 1 than semester 2 each year and this is reflected in the number of students in each semester who participated in this study.

<table>
<thead>
<tr>
<th></th>
<th>Teaching Approach</th>
<th>No. of Students (N)</th>
<th>Failure rate N (%)</th>
<th>Dropout rate N (%)</th>
<th>Attrition rates N (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Semester 1, 2013-14</td>
<td>Traditional</td>
<td>173</td>
<td>23 (13.1%)</td>
<td>22 (12.7%)</td>
</tr>
<tr>
<td>2</td>
<td>Semester 2, 2013-14</td>
<td>Traditional</td>
<td>79</td>
<td>12 (15.2%)</td>
<td>9 (11.4%)</td>
</tr>
<tr>
<td>3</td>
<td>Semester 1, 2014-15</td>
<td>ADRI</td>
<td>135</td>
<td>15 (11.1%)</td>
<td>2 (1.5%)</td>
</tr>
<tr>
<td>4</td>
<td>Semester 2, 2014-15</td>
<td>ADRI</td>
<td>66</td>
<td>6 (9.1%)</td>
<td>2 (3.0%)</td>
</tr>
</tbody>
</table>
6 IMPROVED ADRI BASED INTRODUCTORY PROGRAMMING COURSE

6.4.1 Comparison of the students’ final grades
We compare the students’ performance by using two different criteria (failure and dropout rates) in the IP course. Previously performed studies (Guzdial & Soloway, 2002; Lahtinen et al., 2005; Sykes, 2007; Yadin, A. 2011; Watson & Li, 2014; Zingaro, 2015) in the same area reported their findings by using the similar criteria allowing me to make comparisons. Yadin (2011) carried out an action research for four semesters to address the high failure rate in an IP course. He reported that the percentage of students who failed was reduced after introducing the action research methodology. Watson and Li (2014) conducted a study with sample data from 51 different institutions across 15 different countries to determine the failure and dropout rates in IP courses. They reported that 32.3% students did not pass an IP course. Bennedsen and Caspersen, (2007) reported 33% failure rate in IP courses after receiving responses from 63 different institutions.

Table 6.6 above depicts the failure and dropout rates for the four semesters during which this study was undertaken at Buraimi College. The failure rates for the two semesters offered with the traditional approach are 13.1% and 15.2%. However, the failure rates for other the two semesters offered with the ADRI approach are 11.1% and 9.1%, showing that the ADRI approach had a positive impact on the performance of the students and failure rates have decreased. Likewise, the dropout rates are 12.7% and 11.4% for those semesters offered with the traditional approach, and are 1.5% and 3.0% for those semesters offered with the ADRI approach. It clearly shows that the ADRI approach had a significant impact on the dropout rates.

In table 6.6, I also calculate and examine the overall impact of the attrition rate on the IP course by summing the failure and dropout for each semester. For semesters offered with
the traditional approach, the attrition is 25.8% and 26.6%. On the other hand, the attrition rate in the semesters offered with the ADRI approach is 12.6% and 12.1%. Overall, the attrition rates are less than half in the semesters offered with the ADRI approach compared with the semesters which offered the traditional approach. It clearly shows the ADRI approach had a significant impact on not only the performance of the students (as represented by the improved failure rate) but also the persistence of students (represented by the significantly reduced dropout rate), compared to the traditional approach.

**6.4.2 Impact of the ADRI approach on the students who passed the course**

I determined the impact of the ADRI approach compared to the traditional approach, not only on the students who failed or dropped the course, but also on the students who successfully passed the course. This process informs me of the broader impact of the ADRI approach on the teaching and learning process. Table 6.7 shows the grading scheme used for the IP course at Buraimi University College.

<table>
<thead>
<tr>
<th>Mark Range</th>
<th>95-100</th>
<th>90-94</th>
<th>85-89</th>
<th>80-84</th>
<th>75-79</th>
<th>70-74</th>
<th>65-69</th>
<th>60-64</th>
<th>55-59</th>
<th>50-54</th>
<th>0-49</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grade Points</td>
<td>4</td>
<td>3.7</td>
<td>3.3</td>
<td>3</td>
<td>2.7</td>
<td>2.3</td>
<td>2</td>
<td>1.7</td>
<td>1.3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Grade</td>
<td>A</td>
<td>A-</td>
<td>B+</td>
<td>B</td>
<td>B-</td>
<td>C+</td>
<td>C</td>
<td>C-</td>
<td>D+</td>
<td>D</td>
<td>F</td>
</tr>
</tbody>
</table>

Total marks are 100 and to pass the course, students should obtain 50 marks and above. I allocated the students who successfully passed the course into three categories, high, medium and low achievers as shown in table 6.8. Students who obtained marks between
85 and 100 are considered as high achievers, those in between 65 to 84 are medium achievers and those in between 50 to 64 are considered as low achievers.

Table 6.8: Three categories of students

<table>
<thead>
<tr>
<th>Category</th>
<th>Mark Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>High achiever</td>
<td>85 – 100</td>
</tr>
<tr>
<td>Medium achiever</td>
<td>65 – 84</td>
</tr>
<tr>
<td>Low achiever</td>
<td>50 – 64</td>
</tr>
</tbody>
</table>

I compared the final results of the students over four semesters. Figure 6.5 shows the results of the students from the traditional (control) and ADRI (treatment) groups.

The values are calculated by adding the number of students in each category for both semesters (semesters 1 & 2, 2013-14 for traditional approach and semesters 1 & 2, 2014-15 for ADRI approach) and then dividing by total number of students in each approach.

Figure 6.5: Impact of the ADRI approach on the students who passed the course
The ADRI approach provides a small, but positive improvement in the high achiever category, with 16.9% of the students who passed the course with the ADRI approach in this category compared to the 16.0% for the traditional approach. The medium achiever category shows an increase of 2.3%; 34% of the students achieved in medium achiever category compared to 36.3% for the ADRI approach. There is a much larger positive improvement in the low achiever category of 10.7%, 34.3% of students with the ADRI approach compared to the 23.6% for the traditional approach. The failure rate with the ADRI approach semesters was 10.4% compared to the 14.0% for the traditional approach semesters, a reduction of 3.6%. Likewise, the dropout rate with the ADRI approach semesters was 2.0% compared to the 12.4% for the traditional approach semesters and it is reduced, significantly, by 10.4%. This demonstrates that there is a positive shift in outcomes for students across all achievement categories.

6.5 Comparison of the traditional and ADRI based introductory programming teaching materials

A detailed analysis of the IP course based on the traditional approach was discussed in chapter 4. The ADRI approach was introduced in relation to the preparation of course materials (see section 5.2) to overcome the problems identified with the traditional approach. In this section, I compare the teaching materials based on the traditional and ADRI approaches to determine the impact of the ADRI approach on the course.

The ACM-IEEE Joint Task Force on Computing Curricula (2013) identifies 18 knowledge areas for computing undergraduate programs. Programming knowledge (syntax and semantics) and problem solving strategies are given equal attention in these
knowledge areas. The IP curriculum should promote both skills so that students will be well prepared for their career path and labour market. Tavares et al. (2001) reported that curriculum organisation and teaching methods were the two main factors for high failure rates in IP courses. De Raadt et al. (2005) analysed forty programming books and concluded that most of the books emphasised the syntax of programming languages. Some authors provide a large number of examples but give less attention to problem solving skills. In some cases, authors discuss problem solving skills in early chapters but these skills are not integrated into the remaining chapters.

Table 6.9 below shows the comparison between the traditional and ADRI based teaching materials for the IP course. The teaching materials are compared in five different categories including: *Teaching topics for lecture sessions*, *Practical topics for lab sessions*, *Types of programming examples and problems*, *Presentation style of example and problems*, and *Four problem solving steps in examples and problems*. These five categories were discussed in detail in chapter 4 (see section 4.4.2). The comparison of the traditional and ADRI approaches in these five categories are discussed in terms of the emphasis on the programming skills and problem solving strategies in each category of teaching materials. Table 6.9 is a summary of tables 4.6 and 6.5.
### Table 6.9: Comparison of the traditional and ADRI based teaching materials

<table>
<thead>
<tr>
<th>No.</th>
<th>Area of concern</th>
<th>Traditional approach</th>
<th>ADRI approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Teaching topics for lecture sessions</td>
<td>More emphasis on syntax and semantics compared to problem solving strategies (See table 4.5)</td>
<td>Pay equal attention to syntax &amp; semantics and problem solving strategies (See table 6.5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Syntax &amp; Semantics</td>
<td>Syntax &amp; Semantics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Problem Solving Strategies</td>
<td>Problem Solving Strategies</td>
</tr>
<tr>
<td></td>
<td></td>
<td>86.6%</td>
<td>83.3%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13.4%</td>
<td>(C++ starts from second topic)</td>
</tr>
<tr>
<td>2.</td>
<td>Practical topics for lab sessions</td>
<td>More emphasis on syntax and semantics compared to problem solving strategies (See table 4.5)</td>
<td>Pay equal attention to syntax &amp; semantics and problem solving strategies (See table 6.5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Syntax &amp; Semantics</td>
<td>Syntax &amp; Semantics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Problem Solving Strategies</td>
<td>Problem Solving Strategies</td>
</tr>
<tr>
<td></td>
<td></td>
<td>86.6%</td>
<td>84%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13.4%</td>
<td>(C++ starts from second topic)</td>
</tr>
<tr>
<td>3.</td>
<td>Types of programming examples and problems</td>
<td>Syntax-oriented examples and problems are dominant in reading materials (50%) and lecture notes (49%) (See table 4.6)</td>
<td>Math (62.7%) and daily-life (25.6%) examples and problems are dominant (See table 6.5)</td>
</tr>
<tr>
<td>4.</td>
<td>Presentation style of example and problems</td>
<td>Problem Statement→Codes presentation style is dominant in lecture notes (77%) and reading materials (70%) (See table 4.7)</td>
<td>Problem Statement→Solution Plans→Codes presentation style is dominant in lectures (83.3%) and labs (84%) (See table 6.5)</td>
</tr>
<tr>
<td>5.</td>
<td>Four problem solving steps in examples and problems</td>
<td>Coding step is dominant in lecture notes (74%) and reading materials (61%) (See table 4.8)</td>
<td>All the four problem solving steps are addressed (See table 6.5)</td>
</tr>
</tbody>
</table>
It is clear from table 6.9 that in Teaching topics for lecture sessions category, the ADRI approach pays equal attention to syntax & semantics and problem solving skills compared to the traditional approach which emphasises only syntax & semantics. The same trend is depicted in Practical topics for lab sessions category. For Types of programming examples and problems category, syntax-oriented is more dominant in the traditional approach compared to math and daily-life which are dominant in the ADRI approach. Problem Statement → Codes style is dominant in the traditional approach under Presentation style of example and problems category compared to the ADRI approach where Problem Statement→Solution Plans→Codes style is dominant. Lastly in Four problem solving steps in examples and problems category, coding is dominant in the traditional approach compared to the ADRI approach where all the four problem solving steps are dominant. The change in emphasis from coding dominating to a more equal presentation of problem solving and coding was the aim of the redesign and modification of the teaching materials in cycles 2 and 3 and this comparison demonstrates that the goal was successfully achieved.

6.6 Discussion

This cycle of action research incorporates and improves the ADRI approach based on the feedback received from cycle 2. The feedback suggested three improvements which were the introduction of an extra section (common syntax errors) in the Result stage of the ADRI approach, inclusion of a glossary for general terms in the ADRI based materials, and addition of supporting features, and highlighted some minor technical issues (Next and Previous sub-menus) in the ADRI editor that required correction.
6 IMPROVED ADRI BASED INTRODUCTORY PROGRAMMING COURSE

6.6.1 Extra section in the Result stage
This additional section helped the students and instructors in the programming lab sessions. The approach of introducing the section about the common syntax and semantic errors helped students in overcoming programming mistakes so that they could utilise more time on the problem domain instead of debugging process. The instructors also had more time to discuss the program design and logic instead of fixing syntax and semantic errors in the students’ code. Overall, it helped in facilitating the deep learning of the programming concepts rather than focusing on “fixing code”.

The new section also helped in reducing the debugging time. The students were given an awareness of some of the programming errors in advance which would be reflected in their programs. The students’ efficiency was improved and this could give them more time for practice and completing the exercises. Moreover, through increased practice, it would help them in developing their programming skills. It could also help in overcoming the frustration that students experience when their programs do not run but rather the compilation process highlights syntax errors. Further it allowed students to get a better understanding of the programming constructs. It provides opportunities for students to critically think about the programming constructs which promotes deep learning.

Three different types of errors helped the students to gain a better understanding of the programming domain. Identification of semantic errors helped them in developing their problem solving skills. On the other hand, the syntax errors and warnings gave them a better understanding of the programming constructs.
6.6.2 Glossary document for general terms
As discussed earlier, in this research, English is a second language for the students so the
glossary document was developed to help them in grasping the programming concepts.
And provide a quick reference for the students.

Most of the programming books focus on syntax and its related terminology. The glossary
document developed for this project covers terminologies from both areas (problem
solving and programming knowledge), to help the students to achieve deep learning of the
programming domain and a better understanding of the problem solving strategies.

6.6.3 New features in the ADRI editor
As discussed earlier, programming is a new domain for the students so they do not have a
good idea about programming file extensions and organising appropriate folders to save
them. The Help menu guides users in saving their programs to make the editor more user-
friendly. Saving the program with proper file extensions and in an appropriate folder is an
important step in a compilation process.

Tool tips were added for all the four stages of the ADRI to help the students to better
understand each stage when they are working in the editor. It also gives a brief and quick
understanding of the ADRI approach to other users if they use the editor without reading
much about the ADRI approach.

Mnemonics and accelerators features provide users more than one option (keyboard or
mouse) to access menus and sub-menus.

Overall, the new features enhance the editor functionality, interaction, and accessibility
for the users without overloading the novice user with unnecessary and advanced
functionality. Ultimately, it promotes the learning and teaching process of introductory programming.

6.6.4 Analysis of final grades of last four semesters

Overall, the ADRI approach showed improvement in the failure and dropout rates compared to the traditional approach. This result is consistent with previous findings of Yadin (2011) who carried out an action research for four semesters to address the high failure rate in an introductory programming course.

Since the ADRI approach provides a clearer picture and a systemic approach of the programming domain to the students from the beginning of the semester, the students’ interest was developed and they were more willing to involve themselves in the learning process. This ultimately impacts on the dropout rates which were reduced significantly.

The dropout rate (10.4%) was reduced significantly which strongly suggest that the students felt more comfortable with the teaching and learning process offered by the ADRI approach. Further, the failure rate (3.6%) was reduced which suggests that the ADRI approach is also helpful for lower achieving students in attaining their learning outcomes. These two factors resulted in a much improved attrition rate. This trend will help us to retain and recruit students in computer science and related disciplines. Moreover, the increased persistence demonstrated by students during the latter two semesters adds value to their learning on a number of levels, not just in programming but also provides them a strong foundation for pursuing their following studies and, ultimately, their chosen career path.
6 IMPROVED ADRI BASED INTRODUCTORY PROGRAMMING COURSE

The ADRI approach provided improvement in the high and medium achiever categories. It also provided a huge improvement in the low achiever category. The students in this category are considered at academic risk, which may push them to fail or drop the course, which is reflected by the high failure and dropout rates in the semesters offered by the traditional approach. The ADRI approach developed their interest and confidence in the programming domain by giving them foundation knowledge and a set of skills, and this is reflected in the reduction in the attrition rate.

6.6.5 Comparison of the traditional and ADRI approaches in the teaching materials

The IP teaching materials based on the traditional and ADRI approaches were analysed. In ‘Teaching topics for lecture sessions’ category, the ADRI approach enhanced the students’ knowledge and confidence in the programming domain. They realised and grasped the programming concepts in a better context which helps them in developing their programming skills. The multi-stage approach to developing a solution helped students in developing their flow of logic in a structured way and to achieve their objectives in the course.

In Practical topics for lab sessions category, the ADRI approach promoted deep learning and developed program design, language features and program comprehension skills in the novices. In contrast, most of the traditional approach lab exercises promote surface learning.

For Types of programming examples and problems category, math and daily-life are dominant in the ADRI approach which helped in developing the students’ interest in the course. The students already had a good knowledge of maths from their school and
current courses so it was comparatively easy for them to solve the problem statement based on it. Moreover, it was easy for the students to understand the requirements of the problem statements based on daily-life scenarios. Syntax-oriented type of programming examples and problems are dominant in the traditional approach which gives good understanding of the programming knowledge to the students but it does not promote problem solving skills.

The traditional approach style (Problem Statement $\rightarrow$ Codes) promotes programming shortcuts. However, for novices it is very important to practice program design as suggested in some research studies (Winslow 1996; Jenkins 2002; De Raadt et al. 2005; ACM-IEEE Joint Task Force on Computing Curricula, 2013). The ADRI approach discourages programming shortcut and promotes the three-step approach (Problem Statement$\rightarrow$Solution Plans$\rightarrow$Codes) which develops program design, language features and program comprehension skills. The whole process aims to give deep learning of the programming domain to the students.

For last category (Four problem solving steps in examples and problems), the ADRI approach promotes deep learning by emphasising the four steps of problem solving (problem analysis, solution planning, coding, and testing/debugging) in examples and problems. On the other hand, the traditional approach emphasises only on the coding which gives surface learning of the programming concepts to the students.

### 6.7 Conclusion

The ADRI based teaching materials were improved based on the feedback received from the students and instructors in cycle 2. There were three suggestions incorporated into
improving the ADRI course; addition of common syntax and semantic errors in the Result stage, inclusion of glossary for general terms used in teaching materials, and addition of supporting features in the ADRI editor. The improved ADRI based course was introduced to the students of semester 2, 2014-15.

All the three suggestions impacted positively on the teaching and learning process of the course. They increased the students’ support regarding the ADRI approach and understanding of the programming domain.

To determine the impact of the ADRI approach on the students’ performance, the grades of the last four semesters of the IP course were analysed and compared. The failure and dropout rates were also compared. This showed that the ADRI approach reduced the failure (3.5%) and dropout (10.4%) rates and had a positive impact on the students’ performance. Overall, the attrition rate was reduced by 13.8%. The ADRI approach also shows positive improvements in the different categories of the students who passed the course.

The IP teaching materials based on the traditional and ADRI approaches were compared showing that the ADRI approach pays equal attention to programming knowledge (syntax and semantics) and problem solving strategies compared to the traditional approach which focuses more on programming knowledge. The traditional approach promotes programming shortcut presentation style (problem statement \(\rightarrow\) Codes) whereas the ADRI approach reduces shortcuts by following the presentation style (Problem Statement\(\rightarrow\) Solution Plans\(\rightarrow\) Codes) which promotes deep learning of the programming domain.
7 Conclusions

7.1 Summary of the research and its contribution

Learning to program is considered to be one of the difficult areas of study for a significant number of novices despite extensive research attempting to address the issue. Consequently, as discussed in chapter 2, high failure and dropout rates are often reported (Guzdial & Soloway, 2002; Lahtinen et al., 2005; Sykes, 2007; Yadin, A. 2011; Watson & Li, 2014; Zingaro, 2015). In this study, I included the three entities of the didactic triangle as suggested by Kansanen, (1999), instructors, students and content, to explore the learning difficulties that students encounter when studying introductory programming.

The ADRI (Approach, Deployment, Result, and Improvement) model was introduced in the teaching and learning process of the introductory programming (IP) course at Buraimi University College, Oman, to address the research problem.

The research was guided by the overarching research question:

*Does the implementation of the ADRI model in an introductory programming teaching context improve student outcomes?*

Action research methodology was selected to explore the research questions and three cycles were executed to investigate the research problem. Nine activities were performed during the span of these three cycles. The IP students, instructors and curriculum were involved in addressing and investigating the research questions. The ADRI model was employed as a theoretical framework for this investigation. The ADRI model is an analytical tool used extensively in the education and business sectors and, as discussed in
7 CONCLUSIONS

chapter 4, Australian and New Zealand universities are using this approach for quality audit purposes. In my implementation, the Approach stage in the model deals with problem solving strategies, the Deployment stage handles programming knowledge (syntax and semantics), the Result stage depicts the input, process and output requirements of the problem statement, and the Improvement stage addresses different programming constructs. Figure 7.1 below shows the ADRI model in which all the four stages are depicted in the circle and corresponding ADRI model stages for teach programming are shown in rectangles.

![Figure 7.1: Four stages of the ADRI model](image)

The implementation of the ADRI approach was performed by following three cycles as follows:

205
7 CONCLUSIONS

In cycle 1, the students who finished the IP course with the traditional approach were involved in exploring their perceptions of the barriers and affordances to learning programming. The first survey was conducted with the students to get their feedback on the topics and associated learning resources in the IP course. Likewise, the IP instructors’ perceptions were included by analysing current teaching materials and assessment tools used in the course. Consequently, the ADRI approach was proposed to address the problems identified in the teaching and learning processes of the IP course. Further details are explored later in this section in relation to the first research sub-question (RQ1).

In cycle 2, the course materials were prepared based on the ADRI approach. The ADRI editor was also prepared to support the ADRI approach in the teaching and learning process of the course. The new materials were introduced to the students of semester 1, 2014-15. The second survey was conducted with the students to get their feedback on the ADRI approach. Moreover, the focus group was conducted with the IP instructors to get their feedback on the ADRI approach. Further details are explored later in this section in relation to the second research sub-question (RQ2).

In cycle 3, the course materials were further improved on the basis of the feedback received from cycle 2. The improved ADRI approach was offered to the students of semester 2, 2014-15. The final grades of the last four semesters of the IP course were analysed and compared to determine the impact of the ADRI approach on the students’ performance. The results of the comparison show improvements in the students’ outcomes after applying the ADRI approach in the teaching and learning context of the IP course. Further details are explored later in this section in relation the third research sub-question (RQ3). Moreover, the IP teaching materials based on the traditional and ADRI
approaches were compared and the results show that the ADRI model provides better approach in preparing teaching materials and improves students’ outcomes (see RQ3 below, for more details).

The four stages of the ADRI approach cover skills required for program design, language features and program comprehension. It follows presentation style (Problem Statement → Solution Plans → Codes) in preparing examples and problems and discourages programming shortcuts (Problem Statement → Codes). The students practice programming knowledge (syntax and semantics) in the Deployment stage and problem solving strategies during the Approach stage in for each problem. The ADRI approach shows the input and process used to solve the problem statement and the output for each problem in the Result stage. The students practice different programming constructs in each question through the Improvement stage. All these factors combine to provide a positive impact on the teaching and learning process and hence improve students’ outcomes in the course.

The overarching research question is broken down into the four sub-questions which will now be discussed.

**RQ1: What are the perceptions of students of the barriers and affordances to learning programming?**

Students are one of the main entities of the didactic triangle followed in this study. Therefore, the perceptions of the students of the barriers and affordances to learning programming were collected through two surveys. The first survey was conducted with the students who finished the course with the traditional approach. The second survey was conducted after introducing the ADRI approach to the students. The results of both
surveys were compared and analysed to determine the impact of the ADRI approach (see section 5.5.1.2) through the use of means and the Mann-Whitney U Test. The comparison shows that the ADRI approach has results in better outcomes for students and provides a positive impact on the learning process compared to the traditional approach.

In both surveys, the same set of questions were asked which captured information about student demographics, various programming concepts, teaching contents, learning situations and teaching materials of the course.

The results from the programming concepts section of the surveys showed that the ADRI approach improved significantly the three areas ‘Understanding problem solving strategies’, ‘Understanding programming structures’ and ‘Learning the programming language syntax’, programming concepts which were perceived as the most difficult learning areas by the control group participants. The significant difference in ‘Using the program development environment’ could be attributed to the inclusion of the ADRI editor. The ‘Compiling and executing programs’, ‘Dividing functionality into procedures’ ‘Gaining access to the computers and networks’ and ‘Designing a program to solve a certain task’ questions also show small but insignificant improvements.

The results from the teaching topic section of the surveys showed that the ADRI approach improved significantly some areas such as arrays, expressions, functions, operators, primitive data types, repetition structures loops, and selection structure if statement. However, some areas such as error handling techniques, input/output statements, parameters, recursion, and variables declaration show small, but insignificant improvements.
The results from the learning situation section showed small but insignificant improvements in questions such as ‘In exercise sessions in small groups’, ‘While working alone on programming coursework’, ‘While studying alone’, ‘In lectures’. The exception was that in the question ‘In lab session’ there was a reduction in the satisfaction with the ADRI approach.

The ADRI approach showed significant differences in relation to most resources areas including introductory programming course book, lecture notes, exercise questions and answers, still pictures of programming structures, and interactive visualisations. One exception was for the question (example program) where there is a small but insignificant improvement.

The four stages of the ADRI approach provided a clear understanding of the programming process to the students. They practiced several activities involved in learning to program such as program design, programming features and program comprehension in each question based on the ADRI approach. This continuous practice for the whole semester helped the students to get a better understanding of the underlying programming concepts compared to the traditional approach. The Approach and Deployment stages developed programming knowledge and problem solving skills in the students which helped them in broadening their programming horizon. The Result and Improvement stages deal with the programming parameters such as input, process, output and different programming constructs which can give deep learning of the programming domain and promote collaborative relationships between the students and instructors. The Result stage helped the students in understanding the requirements of the problem domain and problem statement. Thus the ADRI approach showed a broader sense of
7 CONCLUSIONS

programming for the novices which is consistent with that suggested by Kölling and Rosenberg (1996).

The ADRI approach pays equal attention to the programming knowledge (syntax and semantics) and problem solving strategies. Firstly, the students have to devise the solution of the given problem statement by using problem solving strategies such as pseudo-code and flowchart. Secondly, they convert this devised solution into a computer program. This practice should help the students in deep learning of the programming domain and concepts which promotes students engagement and progression in the course. So the ADRI approach is consistent with that suggested by Australian Computer Society (2013).

As discussed in chapter 2, Lane and VanLehn (2012) discussed that novices do not tend to plan their program before making an attempt to solve it, even though, program planning is one of the important activities in learning programming languages (Ala-Mutka, 2003). The ADRI approach addresses these issues and provides an opportunity for novices to plan their program before starting to code it. The Approach and Result stages of the ADRI model provides help to novices in planning their program. So the ADRI approach is consistent with that suggested by Lane and VanLehn (2012) and Ala-Mutka (2003).

**RQ2 What are the perceptions of instructors of the barriers and affordances to teaching programming?**

In this study, the IP instructors are one of the main entities of the didactic triangle in addition to the students and contents. The perceptions of the instructors were collected through the focus group and analysis of the teaching and assessment materials.
An initial analysis of the teaching materials used in the course revealed that problem solving strategies were given only a small proportion of the lecture (13% teaching hours) and lab (two lab exercises/tutorials out of 15 labs) time. Most of the lecture (87% teaching hours) and lab (13 lab sessions) time were spent on teaching programming knowledge (syntax and semantic). The programming examples and problems play an important role in the teaching and learning process of the IP course. Therefore, all the programming examples or problems given in the lecture notes, reading materials and class exercises were categorised based on three different criteria within four categories. This analysis demonstrated that the *syntax-oriented* category (*reading materials* (50%), *lecture notes* (49%)) was dominant in programming examples and problems and the presentation style, *Problem Statement → Codes* (*lecture notes* (77%) and *reading materials* (70%)) was dominant throughout, promoting programming shortcuts to students. Finally, they were categorised against the various stages of problem solving. It was evident that the *coding* step (*lecture notes* (74%) and *reading materials* (61%)) was dominant in the teaching materials.

The focus group identified the instructors’ first impressions of the ADRI approach, strengths, weaknesses, impact on student’ learning, comparison with traditional approach and potential further improvements. The participants agreed that introductory programming appears very weird for the novice students and it is always hard for an instructor to teach it. They also agreed that the ADRI approach is a sophisticated way of helping the students understands programming knowledge. The participants affirmed that ADRI approach is an improvement on the traditional approach, however they also indicated it was relatively difficult for the students to come to terms with the complexity
of the ADRI approach initially. The overall impact of the ADRI approach on the students learning appeared positive. The participants gave some suggestions to further improve the ADRI approach which were incorporated into the ADRI approach. Three types of programming errors such as syntax errors, semantic errors and syntax warnings were included in the Result stage of the ADRI approach. The glossary document was prepared which includes general terms from problem solving and programming knowledge. More supporting features such as Help menu, tool tip, mnemonics and accelerators were added in the ADRI editor.

*RQ3: What is the impact of applying the ADRI approach on introductory programming course materials?*

The ADRI approach was incorporated in preparing the IP course materials to address the feedback received from the students who finished the course with the traditional approach and the feedback collected after analysing the current teaching materials of the IP course. The main points of the feedback were already discussed above in research question 2 (RQ2). The issues raised through the analysis of the traditional approach were addressed in the ADRI approach. The four stages of ADRI approach promoted programming knowledge and problem solving strategies. The programming examples or problems were presented in such a way to encourage novices to pay equal attention on programming knowledge and problem solving strategies and to help novices to understand the underlying process to solve problem statements. In the traditional based teaching materials of the IP course, problem solving strategies are discussed only in introductory topics. The ADRI approach incorporates problems solving strategies in all topics of the course. Therefore, it should support deep learning of the programming concept and
promote the three-step Problem Statement → Solution Plans → Codes presentation style in demonstrating programming examples and problems. It could help novices in understanding the programming concepts in different ways. Additionally, in cycle 3, the IP teaching materials from both approaches were compared on the basis of five categories such as Teaching topics for lecture sessions, Practical topics for lab sessions, Types of programming examples and problems, Presentation style of example and problems, and Four problem solving steps in examples and problems (see section 6.4).

This study shows that the ADRI approach provided positive impact on the course materials. The feedback received from the instructors and students after introducing the ADRI approach in the course was positive and encouraging. The analysis of final grades among the students also shows that the ADRI approach improved the students’ engagement and progression in the course.

The ADRI based course materials give a clear direction for the students to achieve their course outcomes. Each stage in the ADRI approach emphasises specialised programming skills required such as program design, language features and program comprehension. The Approach stage covers problem solving strategies such as pseudo-code and flowchart. The Deployment stage deals with programming knowledge such as syntax and semantics. The Result stage shows input, output, the process involved in solving the problem statement, and common syntax errors. Lastly, the Improvement stage covers programming knowledge with different programming constructs. Thus the course materials based on the ADRI approach are consistent with those suggested by The ACM-IEEE Joint Task Force on Computing Curricula (2013) to emphasise both programming knowledge and problem solving strategies knowledge areas.
7 CONCLUSIONS

As discussed in chapter 2, Soloway (1986) argued that traditional approaches tend to teach programming knowledge explicitly and problem solving implicitly. Consequently, novices have to develop their own problem solving skills (Rist, 1991). The ADRI approach addresses this issue and provides both skills: programming knowledge and problem solving explicitly. The first stage of the ADRI model, the *Approach* covers problem solving strategies and second stage, the *Deployment* emphasises programming knowledge. So it is obvious that the ADRI approach is consistent with that suggested by Soloway (1986) and Rist (1991).

The ADRI editor also supports the students in practicing the ADRI approach. Its separate view for all the four stages along with embedded exercise questions promotes problem solving and programming knowledge skills. It also encourages students to follow an appropriate programming process:

*Problem Statement ➔ Problem solving strategies ➔ Programming knowledge*

instead of taking coding shortcuts:

*Problem Statement ➔ Codes*

This programming process is consistent with that suggested by Webster (1994).

The glossary document for general programming terms also provides a simple and quick reference for the students, written in plain English. It covers the terms from the programming knowledge (syntax and semantics) and problem solving areas.

The comparison of the teaching materials based on the traditional and ADRI approaches show that *Teaching topics for lecture sessions* category, the ADRI approach pays equal
attention to programming knowledge (syntax & semantics) and problem solving skills compared to the traditional approach who only emphasises programming knowledge. The same trend is described in *Practical topics for lab sessions* category. For *Types of programming examples and problems* category, syntax-orientation is more dominant in the traditional approach compared to math and daily-life which are dominant in the ADRI approach. Problem Statement → Codes style is dominant in the traditional approach under *Presentation style of example and problems* category compared to the ADRI approach where Problem Statement → Solution Plans → Codes style is dominant. Lastly in *Four problem solving steps in examples and problems* category, coding is dominant in the traditional approach compared to the ADRI approach where all the four problem solving steps are dominant. The comparison reveals that the ADRI approach impacts positively on the teaching materials in relation to achieving the course outcomes. Moreover, it presents the programming concepts in a structured way to help the students in following the course materials smoothly.

*RQ4: What is the impact of applying the ADRI approach on student learning?*

The ultimate goal of this study is to enhance the students’ learning outcomes in the IP course by applying the ADRI approach. Therefore, it evaluates the impact of the ADRI approach on the student learning in two ways. Firstly, it compares the students’ responses obtained by conducting surveys with each group of students. The comparison shows that the ADRI approach provided better results compared to the traditional approach. Secondly, the final grades of the course for over four semesters were compared against failure and dropout rates. This criteria was used in previous studies (Guzdial & Soloway, 2002; Lahtinen et al., 2005; Sykes, 2007; Yadin, A. 2011; Watson & Li, 2014; Zingaro,
7 CONCLUSIONS

2015) to report on the students’ performance. Figure 7.2 below shows the failure rate for the last four semesters. The semesters with the ADRI approach show better results compared to the traditional approach. The failure rates in those semesters offered with the ADRI approach were lower compared to those semesters offered with the traditional approach. This trend shows the students’ progression in the course. The ADRI approach helps the students in getting a better understanding of the programming domain.

![Failure Rate Graph](image)

*Figure 7.2: Failure rates in the IP course*

Figure 7.3 below depicts the dropout rates for the last four semesters. The ADRI approach had significant positive impact on dropout rates compared to the traditional approach. This impacts the students’ engagement and enrolment in computer science discipline.
Figure 7.3: Dropout rates in the introductory programming course

Figure 7.4 below shows the attrition rate (failure + dropout) for the last four semesters of the course. The ADRI approach reduced the attrition rate to almost half of the traditional approach. The overall trend shows that the ADRI approach impacted significantly on the students learning. They felt more comfortable with the new approach which helped them in achieving their objectives. This result is consistent with previous findings of Yadin (2011) who carried out an action research for four semesters to address the high failure rate in an IP course.

The ADRI approach not only improved the performance amongst at risk students but also the performance of higher achieving students although the degree of improvement in performance reduced as the overall results increased.
Figure 7.4: Attrition rate in the introductory programming course

It is evident from the above discussion that the ADRI approach impact positively on the students’ learning in the course. The students’ responses in the survey and the grades in the semesters offered with the ADRI approach were better compared to the traditional approach.

**ADRI approach addresses factors influencing an introductory programming course**

I discussed factors influencing introductory programming learning in section 2.8 and highlighted and elaborated areas requiring further research in table 2.3. In this study, I addressed these areas by introducing the ADRI approach in the learning and teaching process of an IP course. Table 7.1 shows how the ADRI approach addresses these areas highlighted as problematic in the traditional approach.
### Table 7.1: ADRI approach addresses factors influencing an introductory programming course

<table>
<thead>
<tr>
<th>Factors</th>
<th>ADRI approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning to program</td>
<td>Equal attention is paid on syntax &amp; semantics and problem solving strategies (Sections 5.2, 5.3 &amp; 6.2)</td>
</tr>
<tr>
<td>Teaching strategy/model</td>
<td>Four stages of the ADRI approach covers programming knowledge (syntax &amp; semantics) and problem solving strategies. All the examples and exercises present in the lectures and labs are based on the ADRI approach (Section 5.4)</td>
</tr>
<tr>
<td>Programming books</td>
<td>ADRI based teaching materials introduce all the examples and exercises based on the four stages of the ADRI approach (Section 5.3) (Appendix E)</td>
</tr>
<tr>
<td>Learning style</td>
<td>ADRI approach promotes deep learning because problem solving strategies and programming knowledge are emphasized thoroughly in all topics (Section 5.2)</td>
</tr>
<tr>
<td>Lecture notes</td>
<td>Examples discussed in the lectures promote deep learning (Problem Statement → Solution Plans → Codes) (Section 5.3.1)</td>
</tr>
<tr>
<td>Lab exercises</td>
<td>Problems given in the exercises promote deep learning (Problem Statement → Solution Plans → Codes) (Section 5.3.1)</td>
</tr>
<tr>
<td>Software development tools</td>
<td>ADRI editor promotes presentation style of (Problem Statement → Solution Plans → Codes) (Section 5.3.2)</td>
</tr>
<tr>
<td>Programming language practice</td>
<td>Four stages of the ADRI approach promote practice and cover program design, language features and program comprehension in all examples and exercises (Section 5.3)</td>
</tr>
</tbody>
</table>

The ADRI approach promotes deep learning of the programming concepts compared to the traditional approach. It presents all the examples and exercises by using the presentation style (Problem Statement → Solution Plans → Codes) which gives broader sense of programming to the students.
7 CONCLUSIONS

7.2 Contribution to practice

Learning to program is considered to be a difficult and challenging task for significant numbers of the students in the IP course despite extensive research attempting to address the issue. Consequently, high failure or dropout rates are reported in different studies. It also affects the students’ engagement and enrolment in the computer science discipline. Research continues to investigate how to improve the students’ learning outcomes in introductory programming. The research in this study has added to this body of knowledge by providing a new approach in teaching and learning process of the introductory programming to enhance the students’ learning outcomes. In particular the research has shown promising results in reducing the high failure and dropout rates in the course.

The findings of the present study provide evidence in support of using the ADRI approach in the teaching and learning process of the course. The four stages of the ADRI approach help the students in grasping the programming concepts. It gives a clear direction to the students in achieving their learning outcomes in the course.

The results demonstrate that the ADRI approach promotes programming knowledge and problem solving skills. It helps the students in getting a better understanding of the programming domain. The four stages of the ADRI approach pay equal attention on programming knowledge and problem solving skills and the students practice them in each ADRI based question.

The results show that the ADRI approach provides a better practice in solving the problem statement. It emphasises that the students should follow the proper programming
process (Problem Statement → Solution plans → Codes) instead of programming shortcut (Problem Statement → Codes). On the other hand, the results show that it takes a long time for the students to finish the ADRI based exercises. Ultimately, it gives the students more opportunity to practice programming questions which increases their programming horizon. Moreover, it is aligned with Winslow’s (1996) findings that if novices want to turn into expert programmers, then they have to practice, practice and practice programming problems.

The results depict that the ADRI approach has a positive impact in controlling the failure and dropout rates in the course. It also impacts positively on all students who passed the course. The ADRI approach provides better understanding and practice of program design, language features and program comprehension to the students. The Approach stage deals with program design, the Deployment and Improvement stages promote language features, and the Result stage supports program comprehension.

The results demonstrate that the ADRI editor engages the students in understanding the programming concepts. Its separate view for all the four stages along with embedded ADRI based questions promotes the ADRI approach. The editor is developed in Java language which is compatible with all operating systems. Moreover it is easy to use and install.

The results show that the ADRI approach impacts positively on the students’ learning compared to the traditional approach. The comparison of the first and second surveys show that the students gave better responses for the ADRI approach compared to the traditional approach. Likewise, the comparison of the final grades of the last four
7 CONCLUSIONS

semesters also indicates that the ADRI approach provides better results compared to the traditional approach.

In the present study, the IP course materials were developed based on the ADRI approach. The results show that the course materials provide positive impact in achieving the students’ learning outcomes. The ADRI approach also promotes students retention and engagement in computer science discipline.

This research study achieves the objectives set in the overarching question.

*Does the implementation of the ADRI model in an introductory programming teaching context improve student outcomes?*

The ADRI approach improves positively students’ learning outcomes in the IP course. It not only helps in reducing the attrition rate but also supports students who passed the course. It provides a new presentation style for programming examples and exercises which discourages programming shortcuts. The four stages of the ADRI approach cover a set of different skills required for novices. The ADRI editor facilitates the learning process of the IP course. Overall, the implementation of the ADRI approach in the IP course provides a positive impact on the teaching and learning process.

7.3 Implications for future research

The scope of this study was limited to only one college. In future work, I would like to investigate the impact of the ADRI approach on the teaching and learning process of an IP course in other institutions. In this way, the feedback will be collected from many sources and impact will be measured in many contexts. This process will help in
7 CONCLUSIONS

highlighting further improvements that could be made to the ADRI approach (and potentially identifying as yet unidentified issues with the approach). Moreover, it will provide broader view of the ADRI approach in the context of teaching and learning processes associated with introductory programming.

In the current study, the ADRI approach was introduced in the IP course. The four stages of the ADRI approach provide skills which are required in most learning areas in the computer science discipline. I plan to introduce the ADRI approach in other fields of computer science studies such as database, data structure etc. to determine if its impact has a wider application than just teaching introductory programming. Investigating whether using the approach impacts on higher level programming courses is also worthy of investigation.

The ADRI editor was prepared and used to promote the ADRI approach in the learning process. It provides basic functionalities compared to other editors available in the market for programming. I plan to add more advance features such as macros, debug, search etc. to provide more support for the users.

The glossary document was prepared to cover most of the terms used in the ADRI based IP course to support students’ learning process. The instructors and students appreciated the document because it explains the terms in a simple and concise way. I plan to extend this support and will hyperlink terms in the ADRI based teaching materials such as examples, exercises and editor to the glossary. It will increase students’ accessibility to the glossary document and support their learning process.
Students’ evaluation of the course is not included in this study. The reason is unavailability of the relevant data due to changes in institutional practices during the course of this study. I plan to include student evaluation in future to obtain an understanding of the ADRI approach from the students’ perspective beyond the achievement of grades. Further I plan to conduct a qualitative study which will include a focus group or in-depth interviews with a sample of students to get more in-depth feedback about the ADRI approach. This will generate rich information, diverse opinions and ideas, and an opportunity to ask follow-up questions to probe deeper insights.

In the current study, the students’ performance was analysed on the basis of failure and dropout rates in the course. In future work, I plan to analyse the students’ performance in each stage of the ADRI approach. It will help me to determine the impact of each stage of the ADRI approach on the learning process and indicate areas for improvements.

The analysis of both surveys indicated a unique opportunity to study groups where the majority of students are female. In future work, I plan to analyse the impact of the ADRI approach on different genders.

7.4 Concluding Remarks

In conclusion, the research described in this thesis shows significant impact of the ADRI approach in achieving improved learning outcomes within the IP course. All the three entities (student, instructor and content) of the didactic triangle were involved. The ADRI approach was incorporated in the teaching and learning process of the course. The four stages of the ADRI approach promote both programming knowledge and problem solving strategies. The ADRI editor facilitates learners in practicing the ADRI approach.
Consequently, the ADRI approach reduces attrition rate (failure and dropout) and impacts positively on the students who passed the course. Moreover, it discourages students from taking programming shortcuts and promotes a more suitable, sustainable programming approach for novices.
Reference List


REFERENCE LIST


Bachu, E, & Bernard, M 2014, ‘Visualizing problem solving in a strategy game for teaching programming’, proceedings of the WorldComp 2014, Las Vegas, USA


Carroll, M, & Palermo, J 2006, ‘Increasing national capability for quality higher education the case of the Sultanate of Oman’, proceedings of the Australian Association for Institution Research annual forum, AAIR, Coffs Harbour, NSW, Australia

Chaparro, EA, Yuksel, A, Romero, P, & Bryant, S 2005, ‘Factors affecting the perceived effectiveness of pair programming in higher education’, proceedings of 17th Workshop of the psychology of programming interest group, Sussex University, UK, pp 5-18


Cooper, G 2014, ‘Using Visual Logic with Pseudo code to Teach an Introductory Programming Course’, proceedings of WorldComp 2014, Las Vegas, USA


Corney, M, Teague, D, & Thomas, R 2010, ‘Engaging Students in Programming’, Proceedings of the 12th Australasian Computing Education Conference (ACE 2010), Australian Computer Society, Brisbane, Australia, pp. 63-72


Denny, P, Luxton-Reilly, A, & Tempero, E 2012, ‘All syntax errors are not equal’, proceedings of the ITiCSE ’12, ACM, July 3-5, pp. 75-80, Israel


Freund, RJ, Mohr, D, & Wilson, WJ 2010, Statistical Methods, 3rd Edition, Academic Press, USA


REFERENCE LIST


Hole, G 2013, Research Skills: The Mann-Whitney Test, handout version 1.0, accessed on 05, October, 2015 <http://users.sussex.ac.uk/~grahamh/RM1web/teaching08-RS.html>

Hook, LJ & Eckerdal, A 2015, ‘On the bimodality in an introductory programming course’, proceedings of IEEE International conference on learning and teaching in computer and engineering (LaTiCE), Taipei, Taiwan, 9-12 April, 2015, pp. 79-86

Huang, HB 2010, ‘What is good action research? Why the resurgent interest?’, Action Research, Vol.8, No.1, pp. 93-109, Sage Publication


232
REFERENCE LIST


Kazdin, AE 1992, Research design in clinical psychology, 2nd Edition, Allyn & Bacon, USA


Kinnunen, P 2009, ‘Challenges of teaching and studying programming at a university of technology- viewpoints of students, teachers and the university’, PhD thesis, Helsinki University of Technology, Finland
REFERENCE LIST


Lane, HC, & Van Lehn, K 2012, ‘Teaching the tacit knowledge of programming to novices with natural language tutoring’, Computer Science Education, Vol. 15, No. 3 pp.183-201


Levine, DM & Stephan, DF 2005, Even You Can Learn Statistics, Pearson, USA


Lyndall, B 2013, ‘Action research and unit improvement in higher education: a case study approach’, Proceedings of the 19th APRRES Conference, RMIT University, Melbourne, VIC.


Mahmoud, QH, Dodosiewicz, W, & Swayne, D 2004, ‘Redesigning introductory computer programming with html, JavaScript and Java’, Proceedings of the 32 SIGCSE technical symposium on computer science education, ACM, USA, pp. 120-124

Marczyk, G, DeMatteo, D, & Festinger, D 2005, Essentials of Research Design and Methodology, John Wiley & Sons, Inc., USA

Mascolo, MF 2009, ‘Beyond student-centered and teacher-centered pedagogy; Teaching and learning as guided participation’, Pedagogy and the Human Sciences, Vol. 1, No. 1, pp. 3-27

Matthiasdottir, A 2006, ‘How to teach programming languages to novices students? Lecturing or not?’, Proceedings of International Conference on Computer Systems and technologies – CompSysTech’06, University of Veliko Tar novo, Bulgaria, pp. 13-1 – 13-7


McIntosh, P 2010, Action Research and Reflective Practice: Creative and Visual Methods to Facilitate Reflection and Learning, 1st Edition, Routledge, UK.
REFERENCE LIST


Meisalo, V, Suhonen, J, Sutinen, E, & Torvinen, S 2002, ‘Formative evaluation scheme for a web-based course design’, *Proceedings of the 7th ITiCSE*, ACM, University of Aarhus, Denmark, pp. 130-134


Microsoft 2015, *what is a Tool Tip?*, retrieved on August 19, 2015


<https://ir.library.dc-uoit.ca/handle/10155/536>


<pkpinc.com/files/NA01MoenNormanFullpaper.pdf>

236


REFERENCE LIST


Scott, RG. 2015, ‘Improving student success in developmental math courses at a two-year college: An action research study’, PhD Thesis, Capella University, USA

Shuhidan, SM 2012, ‘Probing the minds of novice programmers through guided learning’, PhD thesis, RMIT University, Australia


REFERENCE LIST


Sumara, D, & Carson, T 1997, Action Research as a Living Practice, New York: Peter Lang Publisher.


Tavares, J, Brzezinski, I, Huet, I, Cabreal, A, & Neri, D 2001, ‘Having coffee with professors and students to talk about higher education pedagogy and academic success’, Proceedings of the 24th International HERDSA conference, Newcastle, Australia


240
REFERENCE LIST


Thomas, L, Ratcliffe, M, Thomasson, B 2004, ‘Scaffolding with object diagrams in first year programming classes: some unexpected results’, proceedings of the 35th SIGCSE technical symposium on Computer science education, ACM, USA, pp. 250-254


Thompson, S 1996, How to program it, University of Kent, retrieved on May08, 2013 <http://www.cs.kent.ac.uk/people/staff/sjt/Haskell_craft/HowToProgIt.html>


VanManen, M 1990, Researching lived experience, 1st Edition, New York, State University of New York, USA


Wiedenbeck, S, LaBelle, D, & Kain, VNR 2004, ‘Factors Affecting Course Outcomes in Introductory Programming’, proceedings of 16th Workshop of the Psychology of Programming Interest Group, Carlow, Ireland, pp. 97-110


REFERENCE LIST


Appendix A - Ethics Approval

Associate Professor Jo Coldwell-Neilson  
School of Information Technology  
Faculty of Science Engineering & Built Environment  
Geelong, Waurn Ponds Campus  
C.c Sohail Iqbal Malik  

1 December 2014

Dear Jo and Sohail

STEC-01-2014-COLDWELL-NEILSON “Role of ADRI model in teaching and assessing novice programmers”

Thank you for submitting a modification to the above project for consideration by the Faculty Human Ethics Advisory Group (HEAG). The HEAG recognized that the project complies with the National Statement on Ethical Conduct in Human Research (2007) and has approved it. You may commence the project upon receipt of this communication.

The approval period is for three years. It is your responsibility to contact the Faculty HEAG immediately should any of the following occur:

• Serious or unexpected adverse effects on the participants
• Any proposed changes in the protocol, including extensions of time
• Any changes to the research team or changes to contact details
• Any events which might affect the continuing ethical acceptability of the project
• The project is discontinued before the expected date of completion.

You will be required to submit an annual report giving details of the progress of your research. Please forward your first annual report on 1/12/15 Failure to do so may result in the termination of the project. Once the project is completed, you will be required to submit a final report informing the HEAG of its completion.

Please ensure that the Deakin logo is on the Plain Language Statement and Consent Forms. You should also ensure that the project ID is inserted in the complaints clause on the Plain Language Statement, and be reminded that the project number must always be quoted in any communication with the HEAG to avoid delays. All communication should be directed to sciethic@deakin.edu.au

The Faculty HEAG and/or Deakin University Human Research Ethics Committee (HREC) may need to audit this project as part of the requirements for monitoring set out in the National Statement on Ethical Conduct in Human Research (2007).

If you have any queries in the future, please do not hesitate to contact me.

We wish you well with your research.

Kind regards,

Sandra Dunoon  
Secretary, Human Ethics Advisory Group (HEAG)  
Faculty of Science Engineering & Built Environment
Appendix B - First Survey
**First Survey**

**Students’ feedback on current practices in introductory programming course**

"استطلاع رأي الطلاب حول التعليمات الحالية في مقرر “مقدمة في البرمجة”"

**Objective:**
The main objective of this survey is to determine the learning experiences of novice programmers in the Introductory Programming course.

"الهدف الأساسي من هذا الاستبيان هو التحقق والتعرف على خبرات العلم التي يكتسبها المبرمجون الجدد خلال مقرر مادة "مقدمة في البرمجة""

**Instructions for Return:** Fill in the survey at your own time and return it to your instructor or student researcher drop box.

"تعليمات من أجل إعادة الاستبيان : فم تعبئة الاستبيان حسب الوقت المناسب، كأو ثم أخذها إلى معملك أو مكتب الباحث"

### DEMOGRAPHIC DETAILS:

<table>
<thead>
<tr>
<th>Question</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is your major?</td>
<td>Computer Science, Information Systems, Software Engineering</td>
</tr>
<tr>
<td>What is your degree?</td>
<td>Diploma, Advanced Diploma, Bachelor</td>
</tr>
<tr>
<td>What is your gender?</td>
<td>Male, Female</td>
</tr>
<tr>
<td>Did you have any programming experience before taking this course?</td>
<td>Yes, No</td>
</tr>
</tbody>
</table>

---

*Note: The image contains Arabic text which translates to the above content.*
### PROGRAMMING CONCEPTS

<table>
<thead>
<tr>
<th></th>
<th>Very Difficult to Learn</th>
<th>Difficult to Learn</th>
<th>Neutral</th>
<th>Easy to Learn</th>
<th>Very Easy to Learn</th>
<th>Not Applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td>I found ...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Using the program development environment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gaining access to the computers and networks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Understanding problem solving strategies (Flowchart, Pseudo code, Structured English etc)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Understanding programming structures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learning the programming language syntax</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Designing a program to solve a certain task</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dividing functionality into procedures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiling and executing programs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finding bugs in my own program</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I found learning about...</td>
<td>very difficult to learn</td>
<td>difficult to learn</td>
<td>neutral</td>
<td>easy to learn</td>
<td>very easy to learn</td>
<td>not applicable</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------</td>
<td>--------------------</td>
<td>--------</td>
<td>--------------</td>
<td>-------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Arrays</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error handling techniques</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expressions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input/output statements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operators (++, -,-, /, /)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameters</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primitive Data Types</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Repetition Structures (loops)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recursion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Selection Structure (if statement)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variables declaration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Are there any other comments you would like to offer about the content of the programming course?

---

---

---

---

---

---

---

---
### APPENDIX B. FIRST SURVEY

#### LEARNING PROGRAMMING

<table>
<thead>
<tr>
<th></th>
<th>Never</th>
<th>Rarely</th>
<th>Sometimes</th>
<th>very often</th>
<th>Always</th>
<th>not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td>I learnt about programming...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In lectures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In lab sessions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>While studying alone</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>While working alone on programming coursework</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In exercise sessions in small groups</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESOURCES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Useless</td>
<td>Not very useful</td>
<td>Somewhat useful</td>
<td>Useful</td>
<td>Very useful</td>
<td>not applicable</td>
</tr>
<tr>
<td></td>
<td>غير مفيد</td>
<td>ليس مفيدا جنبا إلى جنب</td>
<td>مفيد إلى حد ما</td>
<td>مفيد</td>
<td>مفيد جدا</td>
<td>لا علاقة</td>
</tr>
<tr>
<td>Introductory Programming course book</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lecture note</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exercise questions and answers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Example programs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Still pictures of programming structures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interactive visualizations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Are there any other comments you would like to offer about your experience of learning programming?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>هل لديك أي ملاحظات أخري تريد أن تضيفها عن تجربتك في تعلم البرمجة؟</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix C – Second Survey
Second Survey

Students’ feedback on ADRI based teaching and learning practices in introductory programming course

Objective:
The main objective of this survey is to determine the impact of ADRI based teaching and learning practices on novice programmers in the Introductory Programming course.

Instructions for Return: Fill in the survey at your own time and return it to your instructor or student researcher drop box.

DEMOGRAPHIC DETAILS:

What is your major?
☐ Computer Science ☐ Information Systems ☐ Software Engineering
(هندسة البرمجيات) (نظم المعلومات) (هندسة البرمجيات) (نظم المعلومات) (هندسة البرمجيات)

What is your degree?
☐ Diploma ☐ Advanced Diploma ☐ Bachelor
(بكالوريوس) (دبلوم متقدم) (بكالوريوس) (دبلوم متقدم) (بكالوريوس) (دبلوم متقدم)

What is your gender?
☐ Male ☐ Female
(ذكر) (أنثى) (ذكر) (أنثى) (ذكر) (أنثى)

Did you have any programming experience before taking this course?
☐ Yes ☐ No
(نعم) (لا) (نعم) (لا) (نعم) (لا)

251
## How do you assess the ADRI based teaching approach in understanding the following programming concepts?

<table>
<thead>
<tr>
<th>Concept</th>
<th>Very Difficult to Learn</th>
<th>Difficult to Learn</th>
<th>Neutral</th>
<th>Easy to Learn</th>
<th>Very Easy to Learn</th>
<th>Not Applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the program development environment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gaining access to the computers and networks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Understanding problem solving strategies (Flowchart, Pseudo code, Structured English etc)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Understanding programming structures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Learning the programming language syntax</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Designing a program to solve a certain task</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dividing functionality into procedures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiling and executing programs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finding bugs in my own program</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

I found …

Not applicable لـ علاقة
### APPENDIX C. SECOND SURVEY

<table>
<thead>
<tr>
<th></th>
<th>very difficult to learn</th>
<th>difficult to learn</th>
<th>neutral</th>
<th>easy to learn</th>
<th>very easy to learn</th>
<th>Not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td>I found learning about</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arrays</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Error handling techniques</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expressions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input / output statements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operators (+, -, *, /, !, &amp;&amp;,</td>
<td></td>
<td>)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameters</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Primitive Data Types</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Repetition Structures (loops)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Selection Structure (if statement)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variables declaration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Are there any other comments you would like to offer about the ADRI based contents of the programming course?

هل لديك أي ملاحظات أخرى تود أن تضيفها عن محوريات مقرر البرمجة المبني على "الدري"؟

________________________________________________________________________________________

________________________________________________________________________________________

________________________________________________________________________________________

________________________________________________________________________________________

________________________________________________________________________________________

________________________________________________________________________________________
APPENDIX C. SECOND SURVEY

<table>
<thead>
<tr>
<th>PROGRAMMING CONCEPTS</th>
<th>Never</th>
<th>Rarely</th>
<th>Sometimes</th>
<th>very often</th>
<th>Always</th>
<th>Not applicable</th>
</tr>
</thead>
<tbody>
<tr>
<td>I learnt about programming…</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In lectures</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>In lab sessions</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>While studying alone</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>While working alone on programming coursework</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>In exercise sessions in small groups</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>How useful did you find the following ADRI based teaching and learning resources for understanding introductory programming?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Introductory Programming course book</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Lecture note</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Exercise questions and answers</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Example programs</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Still pictures of programming structures</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Interactive visualizations</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>

Are there any other comments you would like to offer about your experience of learning programming?

:\n
254
Appendix D – Focus Group

Focus Group Objectives

To explore instructors’ feedback on:

- Students’ experiences with ADRI approach in introductory programming course.
- Impact of ADRI approach on students learning in introductory programming course

To explore instructors’ views on:

- ADRI approach strengths and weaknesses
- comparison of ADRI approach with traditional teaching approach in introductory programming course
- further enhancements in ADRI approach for introductory or other programming courses

Themes

1. Experience in teaching introductory programming course
2. First impression about ADRI based approach
3. Strengths of the ADRI approach
4. ADRI approach weakness
5. Impact of ADRI approach on students learning
6. Comparison of ADRI approach with traditional teaching approaches
7. Suggestions for further improvement in ADRI approach
8. Other remarks and feedback
Appendix E - ADRI based Examples, Exercises and Syntax Errors questions
Example 1:

Design an algorithm that will read name, balance, and rate then calculate interest, and print name and interest.

**Step 1: Approach – Problem solving strategies**

(a) Solve above mentioned problem by using Pseudo code techniques

1. Start
2. Read NAME, BALANCE, RATE
3. Compute INTEREST as
   \[ \text{BALANCE} \times \text{RATE} \]
4. Write NAME and INTEREST
5. Stop

(b) Approach – Flowchart

```
Start

Read NAME, BALANCE, RATE

INTEREST = BALANCE \times RATE

Write NAME, INTEREST

Stop
```

**Step 2: Programming Knowledge**

Not discussed in this chapter

**Step 3: Result**

<table>
<thead>
<tr>
<th>Input:</th>
<th>Process:</th>
<th>Output:</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME : Naushad</td>
<td>INTEREST = 20 \times 0.25</td>
<td>Name: Naushad</td>
</tr>
<tr>
<td>BALANCE: 20</td>
<td></td>
<td>Interest: 5</td>
</tr>
<tr>
<td>RATE : 0.25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Step 4: Improvement**

Update above algorithm to calculate BALANCE if INTEREST is 7 and RATE is increased from 0.25 to 0.35.

1. Start
2. Input NAME
3. Input INTEREST
4. Input RATE
5. Compute the BALANCE as
   \[ \text{BALANCE} = \text{INTEREST} / \text{RATE} \]
6. Write NAME, BALANCE
7. Stop

Calculate BALANCE for Inputs:

| NAME : Ahmad | INTEREST : 7 | RATE : 0.35 |
Example 2:

Design an algorithm that reads two different numbers and displays the largest number.

**Step 1: Approach – Problem solving strategies**

(a) Solve above mentioned problem by using Pseudo code techniques

1. Start
2. Read VALUE1, VALUE2
3. If (VALUE1 > VALUE2) Then
   - MAX = VALUE1
4. Else
   - MAX = VALUE2
5. End If
6. Write “The largest value is” MAX
7. Stop

(b) Approach – Flowchart

**Step 2: Programming Knowledge**

Not discussed in this chapter

**Step 3: Result**

Input:

<table>
<thead>
<tr>
<th>VALUE1</th>
<th>VALUE2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Process:

$2 > 4$

Output:

The largest value is 4

**Step 4: Improvement**

Update above algorithm that will read two numbers then displays the smallest number.

1. Start
2. Input X
3. Input Y
4. If (X < Y) Then
   - min = X
5. Else
   - min = Y
6. End If
7. Write “The smallest number is” min
8. Stop

Calculate smallest number for input:

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>45</td>
<td>70</td>
</tr>
</tbody>
</table>
APPENDIX E.  ADRI BASED EXAMPLES, EXERCISES AND SYNTAX ERRORS QUESTIONS

Example 3:

Write a C++ program that prompts the user to input the elapsed time for an event in hours, minutes, and seconds. The program then outputs the elapsed time in seconds.

Step 1: Approach – Problem solving strategies

<table>
<thead>
<tr>
<th>Pseudo-code</th>
</tr>
</thead>
<tbody>
<tr>
<td>6. Start</td>
</tr>
<tr>
<td>7. Read Second, Minutes, Hour</td>
</tr>
<tr>
<td>8. Total_time_second = Second + Minutes * 60 + Hour * 3600</td>
</tr>
<tr>
<td>9. Print Total_time_second</td>
</tr>
<tr>
<td>10. Stop</td>
</tr>
</tbody>
</table>

Flowchart:

```
start
  Read Second, Minutes, Hour
  Total_time_second = Second + Minutes * 60 + Hour * 3600
  Print Total_time_second
  stop
```

Step 2: Programming Knowledge

```cpp
#include <iostream>
using namespace std;

int main() {
  int second, hour, minute, total_time_second;
  cout<< "enter the time in second." ;
  cin>>second;
  cout<<endl;
  cout<< "enter the time in minutes." ;
  cin>>minute;
  cout<<endl;
  cout<< "enter the time in hours." ;
  cin>>hour;
  cout<<endl;
  total_time_second = second + minute*60 + hour*3600;
  cout<<"the total time in second = "<<total_time_second<<endl;
  return 0;}
```

Step 3: Result

Expected output:

| enter the time in second:   65 |
| enter the time in minutes:  20 |
| enter the time in hours:    2 |
The total time in second: 8465

Achieved output:

Goals:  
Achieved: [ ]  Not Achieved: [ ]

Syntax Error: Incompatible data types / Initialization

Example:

```cpp
7    int second = "Hello";
```

Reason: integer value is expected in variable second on line 7. The string value (Hello) is assigned to integer variable.

Step 4: Improvement

Update above program that prompts the user to input seconds and calculate hour, minutes and remaining seconds.

Expected output: Enter the time in Seconds : 7502

<table>
<thead>
<tr>
<th>Hours</th>
<th>Minutes</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

259
**Example 4:**

Write a program that will read the radius of a circle then calculate the area and circumference of the circle and print area and circumference.

\[
\text{Area} = \pi \times \text{Radius} \times \text{Radius} \\
\text{Circumference} = 2 \times \pi \times \text{Radius}
\]

**Step 1: Approach – Problem solving strategies**

<table>
<thead>
<tr>
<th>Pseudo-code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Start</td>
</tr>
<tr>
<td>2. Read radius R</td>
</tr>
<tr>
<td>3. ( \pi = 3.14 )</td>
</tr>
<tr>
<td>4. Calculate area (AR) = ( \pi \times R \times R )</td>
</tr>
<tr>
<td>5. Calculate circumference (CR) = ( 2 \times \pi \times R )</td>
</tr>
<tr>
<td>6. Print AR, CR</td>
</tr>
<tr>
<td>7. Stop</td>
</tr>
</tbody>
</table>

**Flowchart**

- Start
- Read R
- \( \pi = 3.14 \)
- AR = \( \pi \times R \times R \)
- CR = \( 2 \times \pi \times R \)
- Print AR, CR
- Stop

**Step 2: Programming Knowledge**

```cpp
#include <iostream>
using namespace std;

int main()
{
    float R, AR, CR;
    const float \( \pi \) = 3.14;

    cout << "Enter Radius: " << cin >> R;
    AR = \( \pi \times R \times R \);
    CR = \( 2 \times \pi \times R \);
    cout << "Area = \"; AR << \" \n Circumference = \"; CR << \"."
    return 0;
}
```

**Step 3: Result**

**Expected output:** Enter Radius: 5
Area = 78.5
Circumference = 31.4

**Process:**
AR = \( 3.14 \times 5 \times 5 \)
CR = \( 2 \times 3.14 \times 5 \)

**Achieved output:**

<table>
<thead>
<tr>
<th>Goals: Achieved:</th>
<th>Not Achieved:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax Error:** Missing Reference to Namespace (Undeclared identifier cout)

1. `#include <iostream>`
2. `int main()`
3. `{`
4. `cout << “welcome to C++;”;
5. return 0;`}

**Reason:** Namespace std is missing after line 1. So include ‘using namespace std;’ after line 1

**Step 4: Improvement**

Update above program so that it also calculates and prints diameter of the circle (Diameter = \( 2 \times \text{Radius} \))

**Expected Output:** Enter Radius: 4
Diameter = 8
Example 5:

Write a C++ program that mimics a calculator. The program should take as an input, two integers and the operation (+, -, *, /) to be performed. It should then output the numbers, the operator, and the result. (For division, if the denominator is zero, output an appropriate message.)

**Step 1: Approach – Problem solving strategies**

**Pseudo-code**

1. Start
2. Read two integers (n1, n2) and an operator (op)
3. If (op == '+') Then
   Print (n1 + n2)
4. ElseIf (op == '-') Then
   Print (n1 - n2)
5. ElseIf (op == '*') Then
   Print (n1 * n2)
6. ElseIf (op == '/') Then
   Print (n1 / n2)
7. Else
8. Print "wrong operator"
9. End If
10. Stop

**Step 2: Programming Knowledge**

```cpp
#include<iostream>
using namespace std;

int main()
{
    int n1, n2;
    char op;
    cout << "enter two integer. " ;
    cin>>n1>>n2;
    cout<< endl;
    cout << "Enter the operation * for multiplication, / for division
          + for addition, - for subtraction " <<endl;
    cin>>op;
    cout<< endl;
    if (op == '+')
        cout << n1 << op << n2 << " = " << n1+n2 << endl;
    else if (op == '-')
        cout << n1 << op << n2 << " = " << n1-n2 << endl;
    else if (op == '*')
        cout << n1 << op << n2 << " = " << n1*n2 << endl;
    else if (op == '/')
    {
        if (n2 !=0)
            cout << n1 << op << n2 << " = " << n1/n2 << endl;
        else
            cout << "you cannot divide over zero" << endl;
    }
    else
        cout("invalid operator") << endl;
    return 0;
}
```

**Step 3: Result**

**Expected output:** Enter two integers: 2 4
Enter the operation * for multiplication, / for division, + for addition, - for subtraction:

**Process:** 2 * 4 = 8

**Achieved output:**

**Goals:** Achieved: ☐ Not Achieved: ☐

**Logical Error:** Invalid If statement syntax
11 if (x > 50);
12 {
13 cout << "x";
14 }
Reason: Remove extra semicolon after if statement at line 11

**Step 4: Improvement**

Update above program using switch statement

**Expected Output:** Enter two integers: 5 3
Enter the operation:

**Process:** 5 - 3 = 2
Example 6:

Write a program that prompts the user to input set of integer items ending with item that is negative then find summation of positive items. (Using while loop)

### Step 1: Approach – Problem solving strategies

**Pseudo-code**

1. Start
2. Set total to zero
3. Read number N
4. while (N >= 0) Read N
   total = total + N
5. Endwhile
6. display total
7. Stop

### Flowchart

Start

1. total = 0
2. Read N
3. while (N >= 0)
   4. total = total + N
5. Endwhile
6. display total
7. Stop

### Step 2: Programming Knowledge

```c++
#include<iostream>
#include<sstream>
using namespace std;

int main()
{
    int total, N;
    total = 0;
    cin >> N;
    while (N >= 0)
    {
        total = total + N;
        cin >> N;
    }
    cout<<"Sum=“<<total;
    return 0;
}
```

### Step 3: Result

**Expected output:** Enter the number: 3 2 4 1 -6
Sum=10

**Process:** Sum = 3 + 2 + 4 + 1

**Achieved output:**

**Goals:** Achieved: [ ] Not Achieved: [ ]

**Syntax Error:** While loop condition

```c++
while (N >= 0)
{
    total = total + N;
    cout<<"Enter the number: ";
    cin >> N;
}
```

**Reason:** Infinite loop due to wrong use of a decrement operator at line 14. Variable i should use an increment operator (i++).

### Step 4: Improvement

Update above program using do while loop.

**Expected Output:** Enter numbers: 3 2 4 1 -6
Sum=10

262
Example 7:

Write a C++ program to find area of a rectangle using function.

Area of Rectangle = Length * Width

**Step 1: Approach – Problem solving strategies**

<table>
<thead>
<tr>
<th>Pseudo-code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Start</td>
</tr>
<tr>
<td>2. Read length L and width W</td>
</tr>
<tr>
<td>3. CallFunction Area(L, W)</td>
</tr>
<tr>
<td>4. Store returned value to AR = Area(L, W)</td>
</tr>
<tr>
<td>5. Print AR</td>
</tr>
<tr>
<td>6. Stop</td>
</tr>
<tr>
<td>7. Area(L, W)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Return(L * W)</td>
</tr>
</tbody>
</table>

**Flowchart**

**Step 2: Programming Knowledge**

```cpp
#include <iostream>
using namespace std;

float Area(float l, float w)
{
    float area;
    area = l * w;
    return (area);
}

int main()
{
    float L, W, AR;
    cout << "Enter Length and Width:
    cin >> L >> W;
    AR = Area(L, W);
    cout << " Area of rectangle = " << AR;
    return (0);
}
```

**Step 3: Result**

**Expected output:** Enter Length and Width: 4 5
Area of a rectangle = 20

**Process:** Area (4, 5)
area = 4 * 5 = 20

**Achieved output:**

**Goals:**

- Achieved: [ ]
- Not Achieved: [ ]

**Syntax Error:** Function call doesn’t match prototype

4       void add (int a, int b)
5       {
6           //......
7       }
8       int main ()
9       {
10          add(2,3,5);
11       }

Reason: add Function requires two parameters. At line 10, three parameters are passed into add function.

Remove one parameter from add function at line 10.

**Step 4: Improvement**

Update above program to calculate perimeter of a rectangle using function.

Perimeter of a rectangle = 2 (length + width)

**Expected Output:** Enter Length and Width: 3 6
Perimeter of a rectangle = 18
Example 8:

Write a program that asks the user to enter 10 numbers each in two arrays. The program then sums the values of both arrays and put them in third array and prints the values of third array.

Step 1: Approach – Problem solving strategies

<table>
<thead>
<tr>
<th>Pseudo-code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Start</td>
</tr>
<tr>
<td>2. Set count to 0</td>
</tr>
<tr>
<td>3. do</td>
</tr>
<tr>
<td>Read number num</td>
</tr>
<tr>
<td>Store num to A[count]</td>
</tr>
<tr>
<td>Increment count by 1</td>
</tr>
<tr>
<td>4. While(count &lt; 10)</td>
</tr>
<tr>
<td>5. Set count to 0</td>
</tr>
<tr>
<td>6. do</td>
</tr>
<tr>
<td>Read number num</td>
</tr>
<tr>
<td>Store num to B[count]</td>
</tr>
<tr>
<td>Increment count by 1</td>
</tr>
<tr>
<td>7. While(count &lt; 10)</td>
</tr>
<tr>
<td>8. Set count to 0</td>
</tr>
<tr>
<td>9. do</td>
</tr>
<tr>
<td>Print C[count]</td>
</tr>
<tr>
<td>Increment count by 1</td>
</tr>
<tr>
<td>10. While(count &lt; 10)</td>
</tr>
<tr>
<td>11. Stop</td>
</tr>
</tbody>
</table>

Flowchart

Step 2: Programming Knowledge

```c++
#include <iostream>
using namespace std;

int main()
{
    int A[10], B[10], C[10], count;
    cout << "Enter 10 numbers in first array:
    for (count = 0; count < 10; count++)
        cin >> A[count];
    cout << "Enter 10 numbers in second array:
    for (count = 0; count < 10; count++)
        cin >> B[count];
    cout << "Sum of first and second arrays in third array:
    for (count = 0; count < 10; count++)
        cout << C[count] << "    ";
    }
    cout << endl;
    return 0;
}
```

Step 3: Result

Expected output:
Enter 10 numbers in first array:  1 2 3 4 5 6 7 8 9 10
Enter 10 numbers in second array:  1 2 3 4 5 6 7 8 9 10
Sum of first and second arrays in third array:
2 4 6 8 10 12 14 16 18 20


Achieved output:

<table>
<thead>
<tr>
<th>Goals: Achieved:</th>
<th>Not Achieved:</th>
</tr>
</thead>
</table>

Step 4: Improvement

Update same program that will print the elements of third array in reverse order

Expected Output: Enter 10 numbers in first array: 1 2 3 4 5 6 7 8 9 10
Enter 10 numbers in second array: 1 2 3 4 5 6 7 8 9 10
Elements of third array in reverse order: 20 18 16 14 12 10 8 6 4 2

Semantic Error: Array Index out of bounds

Example:
10 int Array[10];
13 for(int i=1; i<10; i++)
14 cout<<Array[i];
Reason: Array index starts at 0, so for loop variable i should initialize with 0 instead of 1 at line 13.

Exercise 1:

Design an algorithm that will read the price of an object then calculate the discount and price after discount.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>If price between 10 and 20</td>
<td>2%</td>
</tr>
<tr>
<td>If price between 21 and 30</td>
<td>3%</td>
</tr>
<tr>
<td>If price more than 30</td>
<td>4%</td>
</tr>
</tbody>
</table>

**Step 1: Approach – Problem solving strategies**

Pseudo-code

**Flowchart**

**Step 2: Programming Knowledge**

Not discussed in this chapter

**Step 3: Result**

Expected output:

Input:
Enter price of an object: 15

Output:
Discount: 0.30
Price after discount: 14.70

Achieved output:

Goals: Achieved: [] Not Achieved: []

**Step 4: Improvement**

Update above algorithm that will read price of an object. If user enter zero or negative price then program will display a message "Price is Invalid" otherwise program will calculate the discount and price after discount.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Discount</th>
</tr>
</thead>
<tbody>
<tr>
<td>If price between 10 and 20</td>
<td>2%</td>
</tr>
<tr>
<td>If price between 21 and 30</td>
<td>3%</td>
</tr>
<tr>
<td>If price more than 30</td>
<td>4%</td>
</tr>
</tbody>
</table>

Input:
Price = -10

Output:
Price is invalid
Exercise 2:

Write a program that prompts the user to enter pay rate and hoursWorked. The program then calculates and prints wage by multiplying pay rate with hoursWorked.

<table>
<thead>
<tr>
<th>Step 1: Approach – Problem solving strategies</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pseudo-code</td>
<td></td>
</tr>
</tbody>
</table>

Step 2: Programming Knowledge

Step 3: Result
Expected output: Pay rate: $ 8
HoursWorked: 20
Your wage is: $ 160

Process:

Achieved output:
Goals: Achieved: [ ] Not Achieved: [ ]

Step 4: Improvement
Update above program so that it calculates wage after adding 10% bonus.

Expected Output:
Pay rate: $ 8
HoursWorked: 20
Your wage including 10% bonus is: $ 176
Exercise 3:

Write a program that prompts the user to input first test, second test, assignment and final exam marks. The program then calculates and prints total marks.

Step 1: Approach – Problem solving strategies

<table>
<thead>
<tr>
<th>Pseudo-code</th>
<th>Flowchart</th>
</tr>
</thead>
</table>

Step 2: Programming Knowledge

Step 3: Result

Expected output: First Test: 17  
Second Test: 14  
Assignment: 6  
Final Exam: 45  
Total Marks: 82

Process:

Achieved output:

Goals: Achieved: [ ] Not Achieved: [ ]

Step 4: Improvement

Update above program so that it calculates and prints percentage of total marks out of 150 marks.

Expected Output:

First Test: 17  
Second Test: 14  
Assignment: 6  
Final Exam: 45  
Total Marks: 82  
Percentage out of 150 marks: 54.67%
Exercise 4:

Write a program that will receive the weight of a parcel and determine the delivery charge for that parcel. (delivery charge = weight * cost per kg)

Charges are calculated as follows:

<table>
<thead>
<tr>
<th>Parcel Weight (kg)</th>
<th>Cost per kg ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;3.5 kg</td>
<td>$4.50 per kg</td>
</tr>
<tr>
<td>3.5–6 kg</td>
<td>$3.75 per kg</td>
</tr>
<tr>
<td>&gt;6 kg</td>
<td>$2.85 per kg</td>
</tr>
</tbody>
</table>

Step 1: Approach – Problem solving strategies

**Pseudo-code**

**Flowchart**

Step 2: Programming Knowledge

Step 3: Result

**Expected output:** Enter weight of a parcel in kg: 3

Delivery charges: $13.5

**Process:**

**Achieved output:**

**Goals:** Achieved: [ ] Not Achieved: [ ]

Step 4: Improvement

Change above program so that it deducts 5% discount on delivery charges.

**Expected Output:** Enter weight of a parcel in kg: 4.5

Delivery charges is $: 16.9
Exercise 5:

Write a program that prompts the user to enter number of employees working in an organization. The program then determines and prints the size of the organization by using following scheme:

<table>
<thead>
<tr>
<th>Number of Employees</th>
<th>Organization Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less than or equal to 10</td>
<td>Very Small Business</td>
</tr>
<tr>
<td>11 – 50</td>
<td>Small Business</td>
</tr>
<tr>
<td>51 – 100</td>
<td>Mid Size Business</td>
</tr>
<tr>
<td>Greater than 100</td>
<td>Large Business</td>
</tr>
</tbody>
</table>

**Step 1: Approach – Problem solving strategies**

<table>
<thead>
<tr>
<th>Pseudo-code</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Step 2: Programming Knowledge**

**Step 3: Result**

**Expected output:** Enter number of employees: 55
The organization is “Mid Size Business”

**Process:**
**Achieved output:**
**Goals:** Achieved: □ Not Achieved: □

**Step 4: Improvement**
Change above program so that it gives appropriate message when a user enters a negative number.

**Expected Output:** Enter number of employees: -6
Please enter positive numbers.
Exercise 6:

Write a program that displays the following pattern by using for loop:

*  
**  
***  
****  
*****

**Step 1: Approach – Problem solving strategies**

<table>
<thead>
<tr>
<th>Pseudo-code</th>
<th>Flowchart</th>
</tr>
</thead>
</table>

**Step 2: Programming Knowledge**

**Step 3: Result**

Expected output:

*  
**  
***  
****  
*****

Process:

Achieved output:

Goals: Achieved: [ ] Not Achieved: [ ]

**Step 4: Improvement**

Update above program so that it displays the following pattern by using for loop:

*****
****
***
**
*
Exercise 7:

Write a function, calculateMonths, which asks the user to enter age as a parameter. The function then computes and prints his or her age in months.

Step 1: Approach – Problem solving strategies

<table>
<thead>
<tr>
<th>Pseudo-code</th>
<th>Flowchart</th>
</tr>
</thead>
</table>

Step 2: Programming Knowledge

Step 3: Result

**Expected output:** Enter your age: 35
Your age in months is: 420

**Process:**

**Achieved output:**

<table>
<thead>
<tr>
<th>Goals:</th>
<th>Achieved:</th>
<th>Not Achieved:</th>
</tr>
</thead>
</table>

Step 4: Improvement

Update above program and write another function, calculateYears. The program asks additional input from the user as age in months and then converts and prints his or her age in years.

**Expected Output:** Enter your age in months: 480
Your age in years is: 40
Exercise 8:

Write a C++ program that specifies four one-dimensional arrays named item, price, quantity, and total; each array should be capable of holding 3 elements.

- The arrays store the order for the sandwiches as follows
  - Item: 'X', 'Y', 'Z'
  - Price: 3.29, 5.79, 2.91
  - Quantity: 8, 7, 2
  - Use a for loop to calculate the total amount for each item and store the total amount in the array called total
    - total = price * quantity
  - Use a for loop, output the item and the total amount associated with the item

Step 1: Approach – Problem solving strategies

<table>
<thead>
<tr>
<th>Pseudo-code</th>
<th>Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 2: Programming Knowledge


Step 3: Result

Expected output: Item       Total
X          26.3
Y          40.5
Z          5.8

Process:

Achieved output:

Goals: Achieved: [ ] Not Achieved: [ ]

Step 4: Improvement

Change above program so that it displays all the relevant information of items.

Expected Output: Item       Price     Quantity   Total
X          3.29       8    26.3
Y          5.79       7    40.5
Z          2.91       2    5.8
Syntax Errors Questions
Each of the following programs contains one error. The error may be logical or syntax error. Rewrite the line where you find the error, or explain what the error is.

// Finds the area of a circle
#include<iostream.h>
void main()
{
    float pi = 3.141;
    float area, r;
    cout<<"Enter Radius ";
    cin>>r
    area = pi *(r*r); //Area of circle
    cout<<"Area is : ";
    cout<<area;
}

// Finds the speed of a vehicle
#include<iostream.h>
void main()
{
    float speed;
    float distance, time;
    cout<<"Enter Distance: ";
    cin>>distance;
    cout<<"Enter Time: ";
    cin>>time;
    speed = distance/time;
    if(speed > 120);
        cout<<"Fine = 10 R.O";
    else
        cout<<"No Fine";
}

/*
Home Work
This program reads two integer value and then Determine which is greater.
#include<iostream.h>
void main()
{
    int x, y;
    cout<<"Enter x: ";
    cin>>x;
    cout<<"Enter y: ";
    cin>>y;
    if(x>y)
        cout<<"x is greater";
    else
        cout<<"y is greater";
}

#include<iostream.h>
void main()
{
    int number = 4;
    // Formula for square
    square = number * number;
    cout<< "Square is: ";
    cout<<square;
}
### APPENDIX E. ADRI BASED EXAMPLES, EXERCISES AND SYNTAX ERRORS QUESTIONS

```cpp
#include<iostream.h>
void main()
{
    float sub_1, sub_2;
    float average;
    cout<<"Enter sub1  ";cin>> sub_1;
    cout<<"Enter sub2  ";cin>> sub_2;
    average = (sub_1 + sub_2)/2;
    if(average >= 50)
        cout<<"Pass"
    else
        cout>>"Fail"
}
```

```cpp
#include<iostream.h>
void main()
{
    float total_bill;
    float discount;
    float total_payable;
    cout<<"Total Bill is: ";
    cin>> total_bill;
    discount = 2;
    total_payable = total_bill - discount;
    cout>>"Last bill is: ";
    cout<<total_payable;
}
```

```cpp
#include<iostream.h>
void main()
{
    int no;
    cout<<"Enter your Number: ";
    cin>> no;
    if(no == 5)
        cout<< "Lucky"
    else
        cout<<"No Lucky"
}
```

```cpp
#include<iostream.h>
#include<stdio.h>
void main()
{
    int lucky_no;
    cout<<"Enter your Lucky Number: ";
    cin>> lucky_no;
    cout<< "Your Lucky Number is "
    cout<<lucky_no;
}
```

---

274
#include<iostream.h>
void main()
int number;
int square;
cout<<"Enter a Number:";
cin>> number;
square = number * number; // Formula for square
cout<< "Square is: ";
cout<<square;
}

#include<iostream.h>
void main()
float sub_1;
float average;
cout<<"Enter sub1 ";cin>> sub_1;
cout<<"Enter sub2 ";cin>> sub_2;
average = (sub_1 + sub_2)/2;
if(average >= 50)
cout<<"Pass";
else
cout<<"Fail";
}

#include<iostream.h>
void main()
char c;
cout<<"Enter Character: ";
cin>>&c;
switch(c)
{
case 'a': cout<<"Aaisha";break;
case 'm': cout<<"Maitha";break;
case 'l': cout<<"Lamya"; break;
default: cout<<"Error";
}

#include<iostream.h>
void main()
int no;
cout<<"Enter your Number:";
cin>> no;
if(no = 5)
cout<< "Lucky";
else
cout<<"No Lucky";
}
#include <iostream.h>
void main()
{
 float discount, bill;
 float total = 25;
 discount = 2;
 if(total>=20)
 {
  bill = total - discount;
  cout<<bill;
 }
 else
 { 
  cout<<"Total: "
  cout<<total;
 }
}

#include<iostream.h>
void main()
{
 int marks[4] = {2, 5, 7, 7, 3};
 int i;
 int sum = 0;
 for(i=0; i<5; i++)
 { 
  cout<<"Marks: ";
  cout<<marks[i];
  cout<<"\n";
 }
}

/* Square of first 6 integers */
I am Maryam
#include<iostream.h>
void main()
{
 int i;
 int square;
 for(i=0; i<=5; i++)
 { 
  square = i*i;
  cout<<square;
  cout<<"\n";
 }

// Finds the speed of a vehicle
#include<iostream.h>
void main()
{
 float speed;
 float distance, time;
 cout<<"Enter Distance: ";
 cin>>distance;
 cout<<"Enter Time: ";
 cin<<time;
 speed = distance/time;
 cout<<speed;
}
#include <iostream.h>
void main()
{
    int numbers[3] = {4.5, 7.0, 8.2};
    int i;
    for(i=0; i<3; i++)
    {
        cout<<i;
        cout<<" ";
        cout<<numbers[i];
        cout<<"\n";
    }
    cout<<"The End";
}

#include<iostream.h>
void main()
{
    char c;
    c = 'a';
    switch(c)
    {
        case 'a': cout<"Ahmed"; break;
        case 'b': cout<"Badriya"; break;
        case 'f': cout<"Fatema";
        default: cout<"Error"; break;
    }
}

#include <iostream.h>
void main()
{
    int pin;
    int mypin = 5213;
    cout<"Enter PIN:";
    cin< pin;
    if(pin == 5213)
        cout<"LogIn";
    else
        cout<"InCorrect Pin";
}

#include <iostream.h>
void main()
{
    float discount, bill;
    float total = 25;
    discount = 2;
    if(total>20)
    {
        bill = total - discount;
        cout<bill;
    }
    else
    {
        cout<"Total: ";
        cout<total;
    }
}

/*
Ths program calculates the
Amount to be paid after
discount
*/
#include <iostream.h>
void main()
{
}
## APPENDIX E. ADRI BASED EXAMPLES, EXERCISES AND SYNTAX ERRORS QUESTIONS

// Finds the speed of a vehicle
#include<iostream.h>
void main()
{
    float speed;
    float distance, time;
    cout<<"Enter Distance: ";
    cin>>distance;
    cout<<"Enter Time: ";
    cin>>time;
    speed = distance/time;
    cout<<speed;
}

#include<iostream.h>
void main()
{
    int i;
    for(i=0; i<3; i++)
    {
        cout<<ids[i];
        cout<<"\n";
    }
}

#include<iostream.h>
void main()
{
    int i;
    char ch = i;
    for(i=0; i<=8; i++)
    {
        cout<<i;
        cout<<ch;
        cout<<"\n";
    }
    cout<<"the end of program";
}

#include<iostream.h>
void main()
{
    int i;
    for(i=0; i<3; i++)
    {
        cout<<ids[i];
        cout<<"\n";
    }
}

// Display the contents of Array
#include<iostream.h>
void main()
{
    float marks[4] = {10.5, 6.5, 5.5, 17};
    int i;
    for(i=0; i<4; i++)
    {
        cout<<marks[i];
        cout<<"\n";
    }
}