

THE USE OF *ALICE*, A VISUAL ENVIRONMENT FOR
TEACHING AND LEARNING OBJECT-ORIENTED
PROGRAMMING

by

JERALINE DWARIKA

submitted in accordance with the requirements
for the degree of

MASTER OF SCIENCE

in the subject

INFORMATION SYSTEMS

at the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: PROF M.R. de VILLIERS

OCTOBER 2013

I declare that **THE USE OF *ALICE*, A VISUAL ENVIRONMENT FOR TEACHING AND LEARNING OBJECT-ORIENTED PROGRAMMING** is my own work, and that all the sources that I have used or quoted in the study have been indicated and acknowledged by means of complete references.

Mrs Jeraline Dwarika

Date

Abstract

University students learning object-oriented programming (OOP) encounter many complexities. This study undertook empirical research aimed at analysing learners' interactions with the *Alice* visual programming environment, which seeks to engage and motivate learners to grasp concepts of OOP, whilst creating animated movies and video games.

A mixed-methods approach was employed, using questionnaire surveys and interviews to investigate learners' experiences with *Alice* and their understanding of OOP. Findings indicated that learners lacked problem-solving abilities; were unable to grasp programming concepts on an abstract level and spent insufficient time practicing programming exercises. *Alice* proved to be an effective tool in helping to address these challenges and in improving learners' grasp of OOP. Learners found *Alice* to have good usability.

Furthermore, test and exam results revealed a statistically significant difference between performances of learners who had been taught *Alice* in comparison to similar learners who were not exposed to the *Alice* intervention.

KEYWORDS

Alice, Object-oriented programming, Visual programming environments, Abstraction, Problem-solving, Motivation, Visualisation, Animation authoring tools, Pedagogy, Data mining

Acknowledgements

“Let us be grateful to people who make us happy; they are the charming gardeners who make our souls blossom.” ~ Marcel Proust

I wish to express my gratitude to:

To my inspirational supervisor, Professor M.R.(Ruth) de Villers, even if every flower in the world had a voice I couldn't send as many as it would take to say thanks enough for your patience, encouragement and meticulous attention to detail. I am truly grateful for your valuable guidance and commitment throughout this study. Most especially, thank you for believing in me and for helping me believe in my capabilities as a researcher.

To my Mircha and Mentor, Suresh Dwarika, for your unconditional love and support. Thank you for always standing by me, for your patience and for dedicating your time to proof reading this dissertation. Time is love, and I could never have achieved this without you, my love. I love you.

Thank you to my parents, Dharam Raj and Anitha Anniroot, for your love and unwavered support, you are both pillars of strength in my life.

To my brother, Justin, for always being there, I am truly grateful.

To the management at DUT (Higher Education Employee Assistance Scheme), for funding received towards my UNISA registration fees.

Professor T. Nepal, the dean of the Faculty of Accounting and Informatics at DUT, for your concern and interest in my studies, and for financial assistance in support of my trip to Berlin Germany in March 2012 for the presentation of a full paper at the IADIS Information Systems Conference.

Professor S. Moyo, the director of DUT's Research and Postgraduate Support, for the research grant allocation seed funding and for providing funds towards my trip to Berlin, Germany. I am grateful for the permission granted for me to conduct research at DUT.

Professor H. Lotriet and the Research Ethical Committee at UNISA, for providing ethical clearance for this study.

Dr. Filistéa Naudé at UNISA, for your assistance with general queries and for sourcing articles.

Mrs. Sara Bibi Mitha at DUT, for your assistance with Endnote referencing and for sourcing articles.

To my statistician, Mr. Deepak Singh, for your insightfulness and expertise.

Mr. Colin Thakur, to you and your staff at the Enterprise Development Unit at DUT, for arranging the refreshments and certificates for the 2012 Alice Workshop.

Mrs. D. Heukelman and Mrs. K. Singh, in your role as HOD: Department of Information Technology, for allowing me access to the IT Department lab facilities to conduct the Alice Workshops, and for affording me a semester of study leave to complete this Masters Degree.

Njabulo Samson Shongwe and Nicole Seobaran, for assisting me in the lab during the Alice Workshop.

To Development Software 2 learners of 2011 and 2012 from the Department of Information Technology at DUT, for participating in the Alice Workshop.

&

To my beloved God for granting me wisdom, knowledge and understanding.

This dissertation is dedicated to
my beautiful daughter, Dhiya,
and loving husband, Suresh Dwarika.
I thank God for blessing me with you both.

Table of Contents

Abstract	iii
Acknowledgements.....	iv
Chapter 1: Introduction	1
1.1 Introduction.....	1
1.2 Background.....	2
1.3 Problem statement	3
1.4 Research goals and objectives.....	6
1.4.1 Research goals	6
1.4.2 Field of investigation	6
1.5 Rationale	7
1.5.1 Learners as beneficiaries	7
1.5.2 Academics as beneficiaries	9
1.6 Research question and associated sub-questions.....	9
1.6.1 Primary research question	9
1.6.2 Sub-questions	9
1.7 Research design and methodology	10
1.7.1 Research model: Framework for design	10
1.7.2 Research process.....	12
1.7.3 Research methods	13
1.7.3.1 Questionnaires	13
1.7.3.2 Interviews	14
1.7.3.3 Test and exam learner data.....	14
1.7.4 Overview of the data collection and analysis process	15
1.8 Research ethics	16
1.9 Scope of the study	17
1.9.1 Domain of the study.....	17
1.9.2 Assumptions	18
1.9.3 Limitations of the scope and research methods.....	18
1.10 Structure of the dissertation	19
1.11 Summary and conclusion	23
Chapter 2: Teaching and Learning Programming	24
2.1 Introduction.....	24
2.2 Programming pedagogy, learning theories, problem-solving strategies and motivation	25
2.2.1 Teaching and learning computer programming	25
2.2.2 Theory of programming behaviour.....	27
2.2.3 Cognitive approach.....	27
2.2.4 Constructivist approach.....	29
2.2.5 Behavioural approach	30
2.2.6 Problem-solving strategies	31
2.2.6.1 Top-down approach	31
2.2.6.2 Bottom-up approach	31
2.2.6.3 Bricolage (trial-and-error) strategy.....	32
2.2.6.4 Hill climbing strategy	32
2.2.6.5 Systematic and as-needed strategies for maintenance	33
2.2.7 Motivation	33

2.3	Learning object-oriented programming	35
2.3.1	What is object-oriented programming?.....	35
2.3.2	Challenges faced by learners of object-oriented programming.....	36
2.3.2.1	Lack of motivation for programming	38
2.3.2.2	Complex syntax, logic and semantics.....	38
2.3.2.3	The need for immediate feedback and identifying results of computation as a created program runs	39
2.3.2.4	Difficulties in understanding compound logic and the application of algorithmic problem-solving skills.....	40
2.4	Techniques to help teach object-oriented programming.....	41
2.4.1	Algorithmic thinking and expression.....	42
2.4.2	Abstraction	42
2.4.3	Objects-first strategy.....	42
2.4.4	Three-dimensional animation authoring tools and visualisation.....	44
2.5	Summary and conclusion	46
Chapter 3:	<i>Alice</i> Visual Programming Environment	47
3.1	Introduction.....	47
3.2	What is a visual programming environment?	47
3.3	Related works: Visual programming environments	48
3.3.1	<i>Logo</i>	48
3.3.2	<i>Karel</i> the robot.....	49
3.3.3	<i>Kara</i>	49
3.3.4	<i>BlueJ</i>	49
3.3.5	<i>Greenfoot</i>	50
3.3.6	<i>Second Life</i>	50
3.3.7	<i>Muppets</i>	50
3.3.8	<i>Scratch</i>	51
3.3.9	<i>Etoys</i>	52
3.3.10	<i>Lego Mindstorms</i>	52
3.4	<i>Alice</i> 2.2 Environment	53
3.4.1	What is <i>Alice</i> ?.....	53
3.4.2	The relevance of <i>Alice</i> visual environment to the current study	54
3.4.3	An overview of <i>Alice</i>	55
3.4.3.1	Background to <i>Alice</i>	55
3.4.3.2	Sample scene and code	55
3.4.4	The approach used by <i>Alice</i> to teach programming.....	61
3.4.4.1	How does <i>Alice</i> address the lack of motivation for programming?	62
3.4.4.2	How does <i>Alice</i> address complex syntax and semantics?.....	62
3.4.4.3	How does <i>Alice</i> address the need for immediate feedback and identifying the results of computation as the created program runs?.....	62
3.4.4.4	How does <i>Alice</i> address the difficulties in understanding compound logic and the application of algorithmic problem-solving skills?.....	63
3.4.5	Shortcomings of <i>Alice</i>	63
3.4.6	Previous case studies: The use of <i>Alice</i> in programming courses	64
3.5	Summary and conclusion	68
Chapter 4:	Research Design and Methodology	69
4.1	Introduction.....	69
4.2	Research questions and where they are addressed	69
4.3	Research design and methodology	72
4.3.1	Philosophical worldviews	73

4.3.1.1	The advocacy and participatory worldview	73
4.3.1.2	The pragmatic worldview	73
4.3.2	Strategy of inquiry: Mixed-methods	74
4.3.2.1	The case study research approach	75
4.3.2.2	Case Study 1	77
4.3.2.3	Case Study 2	78
4.3.2.4	Addressing the primary research question	80
4.3.2.5	Triangulation	80
4.3.3	Research methods	81
4.3.3.1	Questionnaires	81
4.3.3.2	Interviews	82
4.3.3.3	Test and exam learner data.....	83
4.4	Population and sample	83
4.5	Design of data collection instruments.....	84
4.5.1	The questionnaire.....	85
4.5.2	Synthesis of evaluation criteria for the questionnaire.....	86
4.5.2.1	Nielsen’s ten heuristics used to assess the usability of <i>Alice</i>	87
4.5.2.2	Teaching and learning programming, challenges faced by learners of OOP, and techniques used to address these challenges	90
4.5.2.3	The researcher’s experience	92
4.5.2.4	Open-ended questions	94
4.5.3	The interview	94
4.6	The pilot study.....	95
4.7	Reliability and validity.....	96
4.8	Data analysis process	98
4.8.1	Quantitative data analysis.....	98
4.8.1.1	SPSS.....	99
4.8.1.2	Viscovery SOMine – Data mining software tool	101
4.8.2	Qualitative data analysis	103
4.8.2.1	The definition of commonly used qualitative terms	104
4.8.2.2	Planning and preparing the analysis	104
4.8.2.3	Themes and codes.....	107
4.9	Ethical considerations	109
4.10	Summary and conclusion	111
Chapter 5: Data Collection and Analysis, Case Study 1		112
5.1	Introduction	112
5.2	Analysis framework.....	113
5.3	Reliability of the questionnaire	115
5.4	Quantitative data analysis: Closed-ended questions.....	115
5.4.1	Data collection and preparation: Closed-ended questions	117
5.4.2	Data analysis and interpretation: Closed-ended questions.....	119
5.4.2.1	Usability of <i>Alice</i> in relation to Nielsen’s general interface design heuristics... ..	120
5.4.2.2	The teaching and learning of programming	125
5.4.2.3	Challenges faced by learners in learning OOP.....	126
5.4.2.4	How to improve the teaching of OOP	130
5.4.2.5	Criteria based on the researcher’s experience	132
5.5	Qualitative data analysis: Open-ended questions	135
5.5.1	Data collection and preparation: Open-ended questions.....	136
5.5.2	Data analysis and interpretation: Open-ended questions	136
5.5.2.1	General overview comments	137
5.5.2.2	Spontaneous positive responses on usability of <i>Alice</i>	138

5.5.2.3	Spontaneous negative responses on usability of <i>Alice</i>	140
5.5.2.4	Challenges faced by learners in learning OOP.....	142
5.5.2.5	Techniques to improve the teaching of OOP	143
5.5.2.6	Impact of <i>Alice</i> on improving understanding of OOP	143
5.5.2.7	Impact of <i>Alice</i> in addressing challenges of OOP	145
5.6	Quantitative data analysis: Test and exam learner data.....	146
5.6.1	Data collection and preparation: Test and exam learner data	146
5.6.2	Data analysis and interpretation: Test and exam learner data	150
5.6.2.1	SPSS data analysis of test and exam learner data.....	150
5.6.2.2	Viscovery SOMine data mining analysis of test and exam learner data	152
5.7	Problem situations in Case Study 1, to be improved in Case Study 2.....	157
5.8	Summary and conclusion.....	158
Chapter 6:	Data Collection and Analysis, Case Study 2.....	159
6.1	Introduction.....	159
6.2	Recommendations implemented to address problems highlighted in Case Study 1	160
6.3	Reliability of the questionnaire	161
6.4	Quantitative data analysis: Closed-ended questions in the questionnaire	162
6.4.1	Data collection and preparation: Closed-ended questions in the questionnaire	163
6.4.2	Data analysis and interpretation: Closed-ended questions in the questionnaire.....	167
6.4.2.1	Usability of <i>Alice</i> in relation to Nielsen's general interface design heuristics... 168	
6.4.2.2	The teaching and learning of programming	173
6.4.2.3	Challenges faced by learners in learning OOP.....	174
6.4.2.4	How to improve the teaching of OOP	178
6.4.2.5	Criteria based on the researcher's experience	180
6.5	Qualitative data analysis: Open-ended questions in the questionnaire.....	183
6.5.1	Data collection and preparation: Open-ended questions in the questionnaire	184
6.5.2	Data analysis and interpretation: Open-ended questions in the questionnaire	184
6.5.2.1	General overview comments: Open-ended questions in the questionnaire	184
6.5.2.2	Spontaneous positive responses on usability of <i>Alice</i> : Open-ended questions in the questionnaire.....	186
6.5.2.3	Spontaneous negative responses on usability of <i>Alice</i> : Open-ended questions in the questionnaire.....	187
6.5.2.4	Challenges faced by learners in learning OOP.....	190
6.5.2.5	Techniques to improve the teaching of OOP	191
6.5.2.6	Impact of <i>Alice</i> on improving understanding of OOP	192
6.5.2.7	Impact of <i>Alice</i> in addressing challenges of OOP	194
6.6	Qualitative data analysis: Interviews	195
6.6.1	Data collection and preparation: Interviews.....	197
6.6.2	Data analysis and interpretation: Interviews	198
6.6.2.1	General overview comments: Interviews.....	198
6.6.2.2	Spontaneous positive responses on usability of <i>Alice</i> : Interviews	200
6.6.2.3	Spontaneous negative responses on usability of <i>Alice</i> : Interviews	202
6.6.2.4	Challenges faced by learners in learning OOP: Interviews	204
6.6.2.5	Techniques to improve the teaching of OOP: Interviews	205
6.6.2.6	Impact of <i>Alice</i> on improving understanding of OOP: Interviews	206
6.6.2.7	Impact of <i>Alice</i> in addressing challenges of OOP: Interviews	208
6.6.2.8	Teaching and learning programming at DUT: Interviews	210
6.6.3	Wrap-up discussion on the interviews	216
6.7	Quantitative data analysis: Test and exam learner data.....	217
6.7.1	Data collection and preparation: Test and exam learner data	217
6.7.2	Data analysis and interpretation: Test and exam learner data.....	221

6.7.2.1	SPSS data analysis of test and exam learner data.....	221
6.7.2.2	Findings derived using Viscovery SOMine	224
6.8	Triangulation between the quantitative and qualitative findings in Case Study 2	228
6.8.1	Common findings	229
6.8.2	Varying findings	236
6.9	Summary and conclusion	239
Chapter 7:	Conclusion.....	241
7.1	Introduction	241
7.2	A visual representation of the study	242
7.3	Effectiveness of a mixed-methods approach.....	245
7.4	Re-visiting the research questions: Integration of empirical findings from Case Study 1 and Case Study 2.....	248
7.4.1	Sub-question 1: What is the effectiveness, as perceived by learners, of using the <i>Alice</i> visual programming environment in addressing the challenges facing novice programming learners within the object-oriented domain?	248
7.4.1.1	The teaching and learning of programming at DUT	248
7.4.1.2	Challenges faced by learners in learning OOP.....	249
7.4.1.3	Techniques to improve the teaching of OOP	250
7.4.1.4	Impact of <i>Alice</i> in addressing the challenges and improving the understanding of OOP	251
7.4.1.5	Criteria based on the researchers' experience	251
7.4.1.6	The answer to Sub-question 1	252
7.4.2	Sub-question 2: How do learners experience the usability of <i>Alice</i> ?	252
7.4.3	Sub-question 3: To what extent do the test and exam results of participating learners relate to those of similar learners who were not exposed to the <i>Alice</i> intervention?.	255
7.4.4	Primary research question: To what extent can the implementation of the <i>Alice</i> visual programming environment in a second-level programming course at the Durban University of Technology improve the performance and learning experience of learners?	256
7.5	Validity and reliability	257
7.6	Future research	258
7.7	Recommendations	259
7.8	Summary and conclusion	261
References	263
Appendix A – Ethical Clearance	272
A.1	Ethical Clearance – Durban University of Technology.....	272
A.2	Ethical Clearance – University of South Africa	273
Appendix B – Case Study 1 Documentation	274
B.1	Learner consent form – <i>Alice</i> workshop and questionnaire	274
B.2	Questionnaire	276
B.3	2011 <i>Alice</i> workshop:Poster 1.....	282
B.4	2011 <i>Alice</i> workshop:Poster 2.....	283
Appendix C – Case Study 2 Documentation	284
C.1	Learner consent form – <i>Alice</i> workshop and questionnaire	284
C.2	Learner consent form – Interview	286
C.3	Questionnaire	288
C.4	Interview	294

C.5 2012 <i>Alice</i> workshop:Poster 1	296
C.6 2012 <i>Alice</i> workshop:Poster 2.....	297
C.7 Example certificate of attendance	298
C.8 Example letter of achievement	299
Appendix D – Quantitative Analysis Tables	300
D.1 Data collection questionnaire instrument with results: Case Study 1	300
D.2 Data collection questionnaire instrument with results: Case Study 2	305
Appendix E – Qualitative Analysis Tables	310
E.1 Initial Codebook: Framework (adapted from Guest <i>et al.</i> , 2012:57)	310
Appendix F – DVD.....	319
F.1 DVD containing screen shots of the Viscovery SOMine cluster maps and component pictures for Case Study 1 and Case Study 2	319
Appendix G – (Final) Conference paper at IADIS IS Conference in Berlin, March 2012.....	320
G.1 Conference paper presented by Anniroot and de Villiers (2012) at the IADIS Information Systems 2012 Conference in Berlin (March) and published in the proceedings. (‘Anniroot’ is the maiden name of the researcher, now ‘Dwarika’).....	320

List of Figures

Figure 1.1	Pictorial insights into the <i>Alice</i> workshop (Conducted in the laboratory of the Department of Information Technology. Photographs used with permission)	5
Figure 1.2	A Framework for Design – The interconnection of worldviews, strategies of inquiry and research methods (adapted from Creswell (2009:5))	11
Figure 1.3	Research process (adapted from Oates (2006:33)).....	12
Figure 1.4	Interrelationships between the chapters	20
Figure 3.1	The <i>Alice</i> interface representing a sample animation	56
Figure 3.2	The five areas of the <i>Alice</i> interface	57
Figure 3.3	The object tree with subparts for the ‘bunny’ object.....	58
Figure 3.4	An initial scene in an <i>Alice</i> world during program execution.....	60
Figure 4.1	The road map that binds together the entire study	71
Figure 4.2	A Framework for Design – The interconnection of worldviews, strategies of inquiry and research methods (adapted from Creswell (2009:5))	72
Figure 4.3	Research process (adapted from Oates (2006:33)).....	75
Figure 4.4	Detailed research processes of Case Study 1 (adapted from Oates (2006:33) and Creswell (2009:209-210)).....	77
Figure 4.5	Detailed research processes of Case Study 2 (adapted from Oates (2006:33) and Creswell (2009:209-210)).....	79
Figure 4.6	Participants completing questionnaires during the 2012 <i>Alice</i> workshop (Photographs used with permission)	82
Figure 4.7	Qualitative and quantitative data analyses (adapted from Guest <i>et al.</i> (2012:6))	99
Figure 4.8	A sample cluster map generated from Viscovery SOMine	103
Figure 5.1	Detailed research processes of Case Study 1, highlighting the quantitative data analysis of the closed-ended responses to the questionnaire	116
Figure 5.2	Detailed research processes of Case Study 1, highlighting a quantification of qualitative data derived from the open-ended responses to the questionnaire.....	136
Figure 5.3	Detailed research processes of Case Study 1, highlighting quantitative data analysis of learner data from the tests and exam	146
Figure 5.4	Codes for results, listed against the associated description	147
Figure 5.5	Bar chart representing the mean score comparison between the experimental and comparison groups.....	151
Figure 5.6	The relationship between a cluster map, a cluster and a node	153

Figure 5.7	Cluster map indicating student numbers within a selected node	153
Figure 5.8	Component pictures for student number, gender and race in the experimental group.....	155
Figure 5.9	Experimental group – 2011.....	156
Figure 5.10	Comparison group – 2011.....	156
Figure 6.1	Detailed research processes of Case Study 2, highlighting the quantitative data analysis of the closed-ended responses to the questionnaire	162
Figure 6.2	Detailed research processes of Case Study 2, highlighting a quantification of qualitative data derived from the open-ended responses to the questionnaire.....	183
Figure 6.3	Detailed research processes of Case Study 2, highlighting a quantification of qualitative data derived from the responses to the semi-structured interviews	197
Figure 6.4	Detailed research processes of Case Study 2, highlighting quantitative data analysis of learner data from the tests and exam	217
Figure 6.5	Codes for results, listed against the associated description	218
Figure 6.6	Frequency plot for the experimental group vs. the comparison group	222
Figure 6.7	Component pictures for student number, age, race and gender in the experimental group.....	226
Figure 6.8	Experimental group – 2012.....	227
Figure 6.9	Comparison group – 2012.....	227
Figure 7.1	Road map that binds the entire study together	243
Figure 7.2	The research process model	244

List of Tables

Table 1.1	Overview of the data collection and analysis process	15
Table 2.1	Levels of Bloom’s Taxonomy with sample questions based on the OOP concept of inheritance (adapted from Starr, Manaris and Stavley (2008:262))	29
Table 4.1	Research sub-questions with corresponding locations in dissertation	70
Table 4.2	Overall breakdown of items for the questionnaire, with corresponding domain area .	86
Table 4.3	Jakob Nielsen’s ten heuristics in relation to the <i>Alice</i> visual programming environment	89
Table 4.4	Synthesis of evaluation principles/criteria mapped to reference and location.....	90
Table 4.5	Criteria derived from the researcher’s experience and relevance to the study	93
Table 4.6	Open-ended questions from the questionnaire with corresponding domain area	94
Table 4.7	Core questions in the interview protocol and corresponding domain area.....	95
Table 4.8	Definition of basic terms in textual qualitative analysis (adapted from Guest <i>et al.</i> (2012:50))	104
Table 4.9	The analysis plan for Study 1 and Study 2 (adapted from Guest <i>et al.</i> (2012:35))....	106
Table 4.10	Structural coding: The interview and questionnaire guide (adapted from Guest <i>et al.</i> (2012:56))	108
Table 5.1	Case processing summary.....	115
Table 5.2	Reliability statistics	115
Table 5.3	A segment of the data set of responses to closed-ended questions in the questionnaire ..	118
Table 5.4	Participant profile table, including bivariate race * gender cross-tabulation.....	119
Table 5.5	Percentage distribution of participant responses for usability of <i>Alice</i>	121
Table 5.6	Ranking of heuristics indicating usability of <i>Alice</i>	123
Table 5.7	Percentage distribution of participant responses for teaching and learning of programming	125
Table 5.8	Percentage distribution of participant responses for challenges facing learners of OOP	127
Table 5.9	Percentage distribution of participant responses for techniques to improve teaching of OOP	130
Table 5.10	Percentage distribution of participant responses for criteria based on researcher’s experience	132
Table 5.11	Positive overview comments on the usability of <i>Alice</i>	137
Table 5.12	Negative overview comments on the usability of <i>Alice</i>	138

Table 5.13	Spontaneous positive responses on usability of <i>Alice</i>	139
Table 5.14	Spontaneous negative responses on usability of <i>Alice</i>	140
Table 5.15	Challenges faced by learners in learning OOP	142
Table 5.16	Techniques to improve teaching of OOP	143
Table 5.17	Impact of <i>Alice</i> on improving understanding of OOP	144
Table 5.18	Impact of <i>Alice</i> in addressing challenges of OOP	145
Table 5.19	Data set 1 - Experimental group.....	148
Table 5.20	Data set 2 - Comparison group.....	149
Table 5.21	t-test for equality of means, mean scores and standard deviation for the experimental group and the comparison group.....	150
Table 6.1	Case processing summary.....	161
Table 6.2	Reliability statistics	161
Table 6.3	A segment of the data set of responses to closed-ended questions in the questionnaire	164
Table 6.4	Participant profile table, including bivariate race * gender cross-tabulation.....	165
Table 6.5	Participant profile table, including bivariate age * gender cross-tabulation.....	166
Table 6.6	Participant profile table, including bivariate age * race cross-tabulation.....	167
Table 6.7	Percentage distribution of participant responses for usability of <i>Alice</i>	169
Table 6.8	Ranking of heuristics indicating usability of <i>Alice</i>	171
Table 6.9	Percentage distribution of participant responses for teaching and learning of programming	173
Table 6.10	Percentage distribution of participant responses for challenges facing learners of OOP	175
Table 6.11	Percentage distribution of participant responses for techniques to improve teaching of OOP	178
Table 6.12	Percentage distribution of participant responses for criteria based on researcher's experience	180
Table 6.13	Positive overview comments on the usability <i>Alice</i> – Open-ended questions in the questionnaire	185
Table 6.14	Negative overview comments on the usability of <i>Alice</i> – Open-ended questions in the questionnaire	185
Table 6.15	Spontaneous positive responses on usability of <i>Alice</i>	186
Table 6.16	Spontaneous negative responses on usability of <i>Alice</i>	188
Table 6.17	Challenges faced by learners in learning OOP	190
Table 6.18	Techniques to improve teaching of OOP.....	191

Table 6.19	Impact of <i>Alice</i> on improving understanding of OOP	193
Table 6.20	Impact of <i>Alice</i> in addressing challenges of OOP	194
Table 6.21	Core questions in the interview protocol	196
Table 6.22	Positive overview comments on the ease of using <i>Alice</i> - Interviews.....	199
Table 6.23	Spontaneous positive responses on usability of <i>Alice</i> - Interviews.....	200
Table 6.24	Spontaneous negative responses on usability of <i>Alice</i> - Interviews	202
Table 6.25	Challenges faced by learners in learning OOP - Interviews	204
Table 6.26	Techniques to improve teaching of OOP - Interviews	205
Table 6.27	Impact of <i>Alice</i> on improving understanding of OOP - Interviews	207
Table 6.28	Impact of <i>Alice</i> in addressing challenges of OOP - Interviews	209
Table 6.29	General overview comments on the teaching and learning of programming at DUT.....	210
Table 6.30	Spontaneous positive responses on the teaching and learning of programming at DUT	211
Table 6.31	Spontaneous negative responses on the teaching and learning of programming at DUT	214
Table 6.32	Data set 1 - Experimental group.....	219
Table 6.33	Data set 2 - Comparison group.....	220
Table 6.34	Summary statistics.....	221
Table 6.35	t-test for equality of means, mean scores and standard deviation for the experimental group and the comparison group.....	224
Table 6.36	Common findings between quantitative and qualitative data for Case Study 2	230
Table 6.37	Varying findings between quantitative and qualitative data for Case Study 2	236

Acronyms

Acronym	Description
2D	Two-Dimensional
3D	Three-Dimensional
ACM	Association for Computing Machinery
API	Application Programming Interface
ATA	Applied Thematic Analysis
CAI	Computer-Assisted Instruction
CPD	Continuous Professional Development
CS1	Computer Science 1
CSET	College of Science, Engineering and Technology
CREC	College of Research and Ethics Committee
CVE	Collaborative Virtual Environment
DS1	Development Software 1
DS2	Development Software 2
DS3	Development Software 3
DUT	Durban University of Technology
EDU	Enterprise Development Unit
GUI	Graphical User Interface
IDE	Integrated Development Environment
IT	Information Technology
ITS	Integrated Tertiary Software
LCD	Liquid Crystal Display
LECGO	Learning Environment for programming using C using Geometrical Objects
LSL	Linden Scripting Language
MUPPETS	Multi-User Programming Pedagogy for Enhancing Traditional Study
ND: IT	National Diploma: Information Technology
NSC	National Senior Certificate
NN	Neural Network
OOP	Object-Oriented Programming
SL	Second Life
SOM	Self-Organising Maps
SPSS	Statistical Package for the Social Sciences
SQL	Structured Query Language
UML	Unified Modeling Language
UNISA	University of South Africa
VPE	Visual Programming Environment

Chapter 1: Introduction

1.1 Introduction

Learning to construct computer programs is considered hard for novices. Learners often struggle to develop the competencies and skills required to code programs that execute correctly. Hence, it is important to understand what makes learning how to program so difficult and how students learn (Matthews, Hin and Choo, 2009). It is considered difficult because “it requires learning about programming concepts and the language of programming at the same time” (Herbert, 2011:3). Furthermore, program execution is a dynamic process, and it is complex to mentally grasp and track how variables change during program execution. Learners have problems visualising all the changes that occur as a computer program runs. Programming involves understanding the task on hand, choosing appropriate methods, coding, debugging and testing an emerging program (Brooks, 1999).

Furthermore, programming courses traditionally emphasise theoretical understanding of programming concepts, as well as application. The theory is reinforced through practical hands-on experience. Learners without prior programming experience are likely to be overwhelmed by the breadth and depth of material, thus contributing to attrition (Moskal, Lurie and Cooper, 2004).

One of the core challenges experienced by programming lecturers is developing and sustaining a high level of learner interest and motivation to learn programming. To develop good programming skills, learners are typically required to do considerable intensive practice on programming exercises and to gain experience in debugging, which they cannot sustain unless they are adequately motivated (Law, Lee and Yu, 2010).

The teaching of programming therefore becomes as complex as learning how to program. Extensive research efforts have been invested in developing techniques to assist in teaching programming and learning programming. During this process, programming has evolved considerably from the traditional imperative (or procedural) programming languages and techniques. The evolution has resulted in a greater emphasis in object-oriented design and implementation (Phelps, Egert and Bierre, 2005). Writing object-oriented programs involves writing code in highly complex programming languages.

The goal of this study was to improve learners' understanding of programming within the domain of Object-Oriented Programming (OOP). This was done through the use of a visual programming environment called *Alice*, which was designed at Carnegie Mellon University to assist novice programmers in learning the concepts of OOP, whilst creating animated movies and video games. The present study further set out to engage learners and motivate them to learn. The intention was to draw their attention to new and exciting ways of programming that can enhance their skills and nurture higher-order critical thinking. The ultimate aim of this study was to see an improvement in learner performance and learning experience.

Based on this introduction, Section 1.2 presents the background to this study, while Section 1.3 addresses the real-world problem statement. The research goals and objectives are identified in Section 1.4. The rationale, namely Section 1.5, provides a motivation for the study. The research question and associated sub-questions are identified in Section 1.6. A detailed research design and methodology is discussed in Section 1.7. The ethical considerations adhered to during this study are explained in Section 1.8. Section 1.9, on the scope of the study, outlines the domain of the research and associated assumptions, as well as its limitations. Finally, Section 1.10 provides the structure of the dissertation before the chapter is concluded in Section 1.11.

1.2 Background

Learners engaged in acquiring programming-skills in South African universities face the challenges mentioned above, particularly the complexity of learning OOP. This further motivates a need for this study. Hadjerrouit (1999) proposed that one of the main reasons why an OOP approach is difficult for most novice learners is because it is more abstract than the procedural approach. This, along with other associated problems, will be further investigated in this study.

Furthermore, Sajaniemi, Kuittinen and Tikansalo (2008) posit that students learning OOP experience problems not only in developing the required skills for writing programs, comprehension, and debugging, but also in understanding the basic object-oriented concepts and terminologies which can be difficult to comprehend. Learners are often daunted by concepts such as objects, classes, abstraction, encapsulation, inheritance, polymorphism, dynamic binding, and modularity, to name a few. These concepts are used to understand problem situations, to design object-oriented models, and to choose appropriate means of implementation (Hadjerrouit, 1999).

The foundation for this study is based on the rich findings of other researchers within the domain of teaching and learning of programming. Several literature sources address the core problem areas that learners struggle with when learning OOP. To alleviate these problems, educators have proposed techniques that assist in the teaching of OOP. This study highlights certain prominent challenges faced by learners of OOP, notably the following:

- (a) Lack of motivation for programming (Esteves, Fonseca, Morgado and Martins, 2008; García-Mateos and Fernández-Alemán, 2009; Dann, Cooper and Pausch, 2009);
- (b) Complex syntax, logic and semantics (Winslow, 1996; Gomes and Mendes, 2007; Dann *et al.*, 2009);
- (c) The need for immediate feedback and identifying results of computation as a created program runs (Wright and Cockburn, 2000; Gálvez, Guzmán and Conejo, 2009; Dann *et al.*, 2009); and
- (d) Difficulties in understanding compound logic and the application of algorithmic problem-solving skills (Gomes and Mendes, 2007; Esteves *et al.*, 2008; Dann *et al.*, 2009).

These challenges are addressed in Section 2.3.2, while Section 2.4 suggests possible solutions in the form of characteristics and methods that can support the teaching of OOP.

The next section presents the real-world problem statement.

1.3 Problem statement

Concern over learner attrition, the lack of learner motivation and high failure rates in programming courses has elicited a drive to use creative approaches to make undergraduate courses more attractive to learners and to contribute towards higher success rates. Furthermore, learners do not easily apply their minds and exercise critical thinking capabilities with regard to problem-solving. Such concerns are prevalent in object-oriented programming worldwide, as articulated in the previous section (Hadjerrouit, 1999; Sajaniemi *et al.*, 2008; Dann *et al.*, 2009), and also occur amongst students at the Department of Information Technology (IT) of the Durban University of Technology (DUT), where the researcher is based and where the research was undertaken.

This study investigates the teaching and learning of computer programming, with the purpose of improving problem-solving skills and academic performance amongst second-year learners of OOP at DUT in Kwa-Zulu Natal, South Africa. As stated above, novice programmers face various challenges and difficulties in learning OOP, in particular: demotivation; the complex syntax and

semantics of an OOP language; the need for immediate feedback; and difficulties in understanding compound logic as well as the application of algorithmic problem-solving skills.

The researcher is a lecturer from the Department of IT at DUT, who has investigated the use of a 3D visual programming environment called *Alice*, with the aim of improving the understanding of fundamental programming concepts and imparting OOP skills. *Alice* is an open source teaching tool, designed to provide first-time exposure to learners on the basics of OOP. It allows them to learn fundamental programming concepts whilst creating 3D animated movies and basic video games, thus providing an engaging interactive environment (Carnegie Mellon University, 2012; Herbert, 2011).

The *Alice* system is a breakthrough in teaching OOP, one of its major strengths being that it can make abstract concepts concrete in the eyes of novice programmers (Dann *et al.*, 2009). For instance, “objects take on the form of physical entities, such as people, buildings, animals and cars. When an objects’ methods are called, they cause the object to perform actions that can be observed” (Gaddis, 2011:xiv).

One of the greatest challenges experienced by learners of programming skills, is the issue of adapting to the syntax and semantics of programming languages. This must be mastered before they can write programs that solve problems. *Alice* presents learners with interesting, motivational scenarios to solve and as a learning platform, achieves success in relieving learners from having to deal with complex syntax mechanics in the early stages of learning to program (Johnsgard and McDonald, 2008). Furthermore, human beings are known to understand concepts better through the power of visualisation. As such, *Alice* was designed to ensure the concrete visualisation of objects and inheritance, while logic errors become visually obvious.

The first study, Case Study 1, commenced during the first term of 2011 and was followed up in 2012 by Case Study 2. In both years an experimental group of participants was purposively extracted from the cohort of learners registered for the subject Development Software 2 (DS2) in the Department of Information Technology. These participants were hand-picked based on their first-year results and the fact that they were doing OOP for the first time. A supplementary *Alice* workshop was held during lunch hours over a two to three-week period, where these participants experienced hands-on interaction with the *Alice* software installed in the labs and also did collaborative projects. Figure 1.1 provides pictorial glimpses into the 2012 *Alice* workshop.



Figure 1.1a: Participants enjoying their first *Alice* lesson



Figure 1.1b: A full complement of 55 participants attended the first session of the second *Alice* workshop (i.e. the 2012 workshop)



Figure 1.1c: Participants engaging with the *Alice* visual programming environment

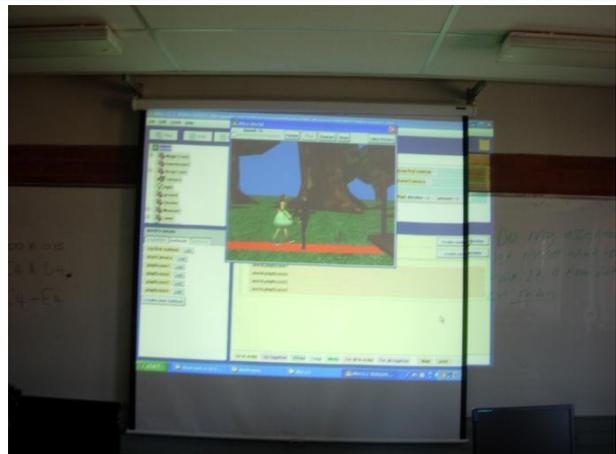


Figure 1.1d: Showcases of the nine *Alice* projects completed by enthusiastic participants, were presented to the entire class on the last day of the workshop, These participants received book prize awards and achievement letters (see Appendix C.8)



Figure 1.1e: All participants with full attendance received attendance certificates issued by the DUT Enterprise Development Unit (EDU) (see Appendix C.7 for an example)

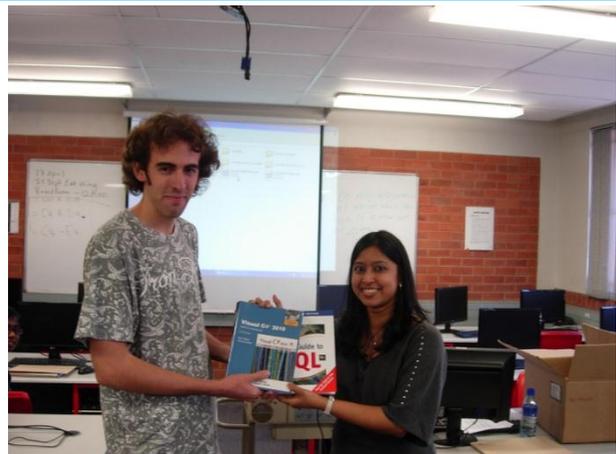


Figure 1.1f: Participant receiving 'First prize' for an outstanding *Alice* project, the prize being the prescribed textbook for DS2 (Visual C# .NET) and a (Structured Query Language) SQL Server textbook

Figure 1.1 Pictorial insights into the *Alice* workshop (Conducted in the laboratory of the Department of Information Technology. Photographs used with permission)

The purpose of the investigation was to determine the effectiveness of implementing the *Alice* visual programming environment, with a long-term view to improving the computer programming performance and learning experience of second-level Information Technology learners. The researcher conducted empirical research on the participants' use of *Alice* and their subsequent programming processes and performance, using research methods described in Section 1.7.3.

In both years the performance of the experimental group was measured against that of a comparison group with a similar success rate at first-year level. The learners in the comparison groups were selected from the other learners registered for DS2, who were taught OOP by conventional methods only. The comparison was achieved by analysing learner data from tests and examinations in both groups.

The next section discusses the research goals and objectives of the study.

1.4 Research goals and objectives

1.4.1 Research goals

This study employed two of the research goals identified by Reeves (2000), namely:

(a) *Theoretical goal* – this concerns the explanation of phenomena through logical analysis and synthesis of theories and principles, as well as the results of other research.

The present research involved an empirical study, with a strong foundation based on existing literature.

(b) *Action goal* – action goals focus on a particular program, product or method in an applied setting, for the purpose of estimating its effectiveness and worth or improving it.

The *Alice* visual programming environment was the object of focus for this study.

1.4.2 Field of investigation

The field of investigation was the domain of methods and tools that support the teaching and learning of OOP within higher education institutions. In particular, visual programming environments have been identified as tools that help improve the performance of learners.

The next section will discuss the potential beneficiaries of this study.

1.5 Rationale

Two groups of stakeholders should benefit from this study, namely learners and academics.

1.5.1 Learners as beneficiaries

The primary beneficiaries of the study are learners at DUT. Development Software is a major subject within the three-year National Diploma: Information Technology (ND: IT) programme. Learners must pass Development Software 1 (DS1) as a prerequisite to register for Development Software 2 (DS2). Likewise, DS2 is a prerequisite subject for Development Software 3 (DS3). As such, there is an increase in the complexity of the syllabus from the first year through to the third year. Approximately 300 learners register for DS2 each year in the first semester.

DS2 is aimed at teaching learners to design and produce software products and systems that meet specified needs, operate reliably, and are cost effective in their production and maintenance. The course attempts to equip them with the skills needed to develop Web Based applications using the Microsoft .NET framework and Microsoft SQL Server 2008 R2. The .NET platform is one of the most widely used development platforms in industry today. For the purposes of instruction in the DS2 syllabus, the C# .NET programming language is currently implemented. Microsoft SQL Server is used for storage and retrieval of data and data manipulation purposes. DS2 learners are therefore taught OOP using conventional methods only. No visualisation tools or supplementary programming support environments are currently used in the course.

DS2 adopts the following learning, teaching and assessment strategies:

(a) *Delivery of the subject*

Lectures are held in the laboratories of the IT Department, and are conducted hands-on, on a practical/tutorial basis. Teaching aids include the whiteboard, overhead projector and LCD projector. However, after concepts are taught and discussed, the learners are required to complete various exercises independently. Due to the nature of this module, it is vital that the learners treat exercises as self-study tutorials and it is strongly advised that they make productive use of the scheduled open lab sessions for this purpose.

(b) *Assessment*

Assessment comprises two tests, one assignment and one examination.

(c) *Activities to promote learning*

It is compulsory for all DS2 learners to register on an online Google forum, which provides a basis for learners to communicate, as well as to share ideas. The forum also provides notifications regarding assignments, test details and scopes, and other useful information.

In DS1, first-year learners are introduced to algorithmic problem-solving techniques, such as pseudocode, flowcharts and IPO tables, before being exposed to basic programming concepts such as selection, iteration, arrays, lists, etc. This is implemented using a procedural programming approach with Microsoft C# .NET as a conventional language. They had no exposure to OOP in their first year. The section on OOP is incorporated into the second year syllabus in DS2, and includes concepts such as inheritance, encapsulation, information hiding, etc. It is at this stage that many learners struggle to gain a solid understanding of OOP concepts.

The benefit of exposing learners to the *Alice* intervention is that they can closely associate the *Alice* software with the corresponding course work in the DS2 syllabus. The sections on OOP were taught using Microsoft C# .NET and, for the experimental groups in this study, were supplemented with knowledge gained from also learning OOP within the *Alice* environment. DS2 prepares the learners for the subject DS3, where the course work is divided between test assessments and an industry-related group project. Working with the *Alice* project can allow learners to interact collaboratively in groups, as is the requirement in DS3.

Although this study targets second-year learners, studies have been conducted elsewhere on using the *Alice* visual environment to enhance the learning of programming with first-year learners. This was considered appropriate due to the strength of *Alice* in making abstract concepts concrete in the eyes of first-time programmers. Such studies were conducted by researchers Cooper, Dann and Pausch (2003); Kelleher and Pausch (2006); Edwards, Gersting and Tangaro (2007); Cliburn (2008); and Salim, Hassan, Hamdi, Youssef, Adel, Khattab and El-Ramly (2010). Findings in these studies demonstrated improvement in learner performance. Some studies identified challenges, in that learners found it difficult to engage with *Alice*. For the purpose of this study, therefore, the participants who were exposed to the *Alice* intervention were second-year learners. Such an intervention at this level of study had not been previously investigated at DUT.

1.5.2 Academics as beneficiaries

The secondary beneficiaries of this study are the academics. It is suggested that teaching and learning should not be confined to traditional methods and norms. Gomes and Mendes (2007) suggest that the high failure rates and difficulties frequently experienced in programming disciplines, may indicate that traditional teaching approaches and study methods are ineffective for many learners. Salim *et al.* (2010) concur that the use of traditional methods to teach programming to novice learners, is a difficult task.

Use of the *Alice* environment in this study allowed an academic, namely, the researcher, to teach DS2 using a visual tool that can enhance the teaching and learning experience in the classroom. In the longer-term, there may be the prospect of future general adoption within the curriculum.

The next section presents the primary research question for this study as well as the associated sub-questions.

1.6 Research question and associated sub-questions

1.6.1 Primary research question

To what extent can the implementation of the Alice visual programming environment in a second-level programming course at the Durban University of Technology improve the performance and learning experience of learners?

1.6.2 Sub-questions

The primary research question gives rise to the following sub-questions:

1. What is the effectiveness, as perceived by learners, of using the *Alice* visual programming environment in addressing the challenges facing novice programming learners within the object-oriented domain?
2. How do learners experience the usability of *Alice*?
3. To what extent do the test and exam results of participating learners relate to those of similar learners who were not exposed to the *Alice* intervention?

These questions will be addressed in the course of this study and the answers will be briefly consolidated in Chapter 7, the Conclusion.

The next section addresses the research design and methodology used to conduct this study.

1.7 Research design and methodology

Every evaluation study has a research methodology and research criteria. Regarding the methodology, this study employed mixed-methods research, which according to Creswell and Plano Clark (2011), is a research design with philosophical assumptions as well as methods. As a method, it focuses on collecting, analysing and combining both quantitative and qualitative data in a single study or series of studies. This form of design helps to broaden understanding by incorporating both qualitative and quantitative research, or to use one approach to better understand, explain, or build on the results from the other (Creswell, 2009).

Section 1.7.1 describes the research model and design framework, adapted from Creswell's (2009) Framework for Design. The research process, which is based on the model proposed by Oates (2006:33), is presented in Section 1.7.2. Section 1.7.3 addresses the research methods employed in this study, while Section 1.7.4 provides an overview of the data collection and analysis process.

1.7.1 Research model: Framework for design

The research design dictated the approach used to conduct this study, which involves the aggregation of philosophy, strategies of enquiry, and specific research methods. These three components were adapted from Creswell's (2009:5) framework, as shown in Figure 1.2, and are considered below under points (a), (b) and (c) respectively. These concepts are covered more extensively in Chapter 4, the chapter on the Research Design and Methodology.

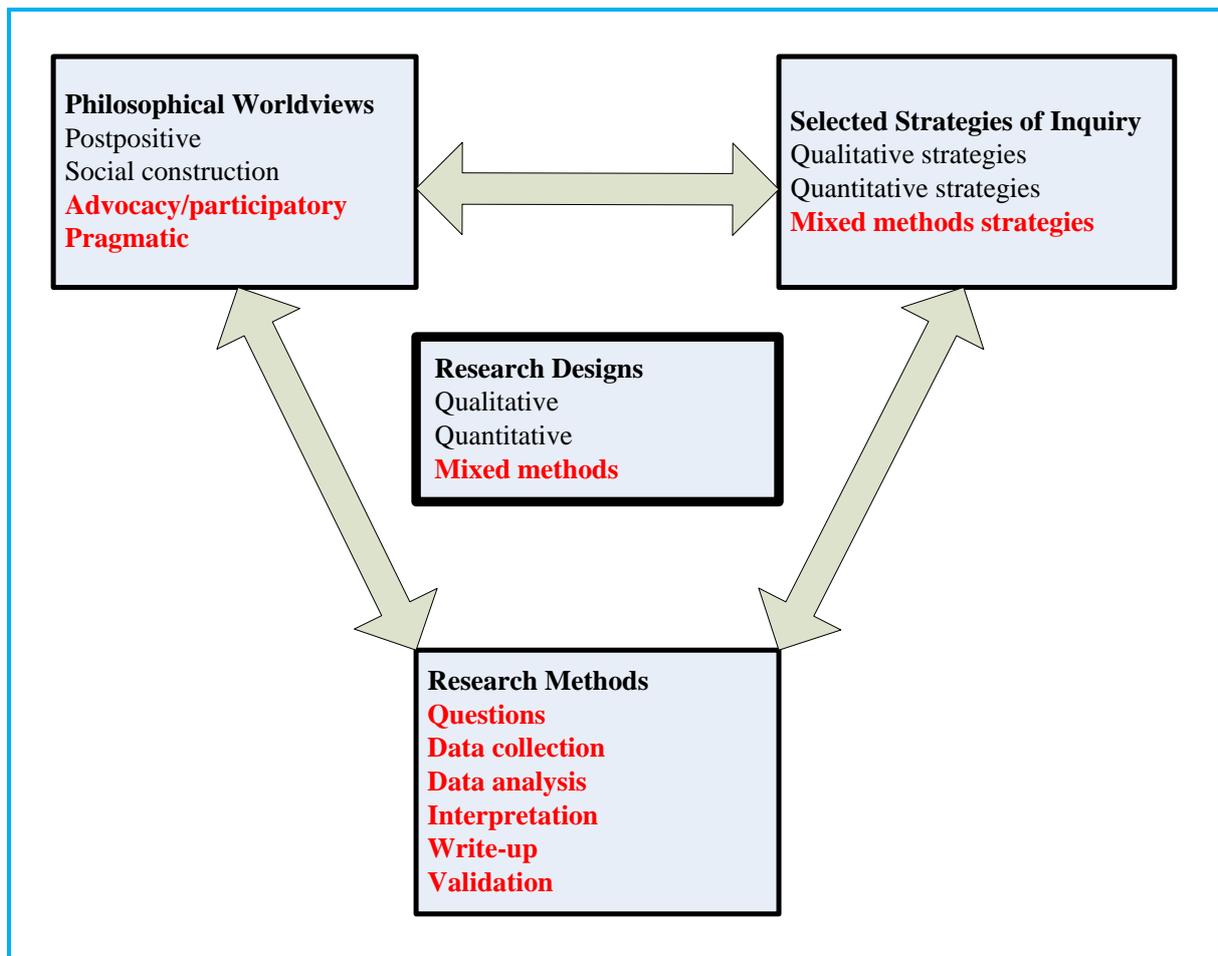


Figure 1.2 A Framework for Design – The interconnection of worldviews, strategies of inquiry and research methods (adapted from Creswell (2009:5))

The lists of approaches below each component are taken directly from Creswell, while the aspects applied in the current study are highlighted in red text in Figure 1.2.

- With regard to the philosophical worldview, this study adopted both an advocacy/participatory worldview as well as a pragmatic worldview;
- With regard to the strategies of inquiry, a mixed-methods approach was employed to answer the research questions. It entailed two case studies, involving both quantitative and qualitative data;
- The research methods involved progressing from the initial research questions to data collection via a questionnaire with closed-ended and open-ended questions, as well as qualitative data collection via interviews. Quantitative research was done on the results of learners' assessments to investigate their performance. Data analysis and interpretation were conducted, followed by the write-up and validation of the findings.

Under the advocacy/participatory worldview, a research study aims to introduce change. It includes action aimed at improving the lives of participants, changing practices in their institution, and

changing the life of the researcher. Pragmatism is also action-oriented, concerned with understanding a real-world problem and determining what works in solving it.

1.7.2 Research process

Figure 1.3 provides an overview of the research process, which is based on the model proposed by Oates (2006:33). The personal experience of the researcher and the motivation for this study, together with an investigation into existing literature, helped to define the research questions listed in Section 1.6. A case study approach was employed to answer the research questions. The data generation methods used within Case Study 1 included questionnaires, as well as learner data from tests and examinations. Case Study 2 used the same data collection instruments, supplemented with interviews to derive richer participant responses and enhance validity. This was done in the interest of triangulating data sources. The use of more than one data generation method is called *method triangulation*.

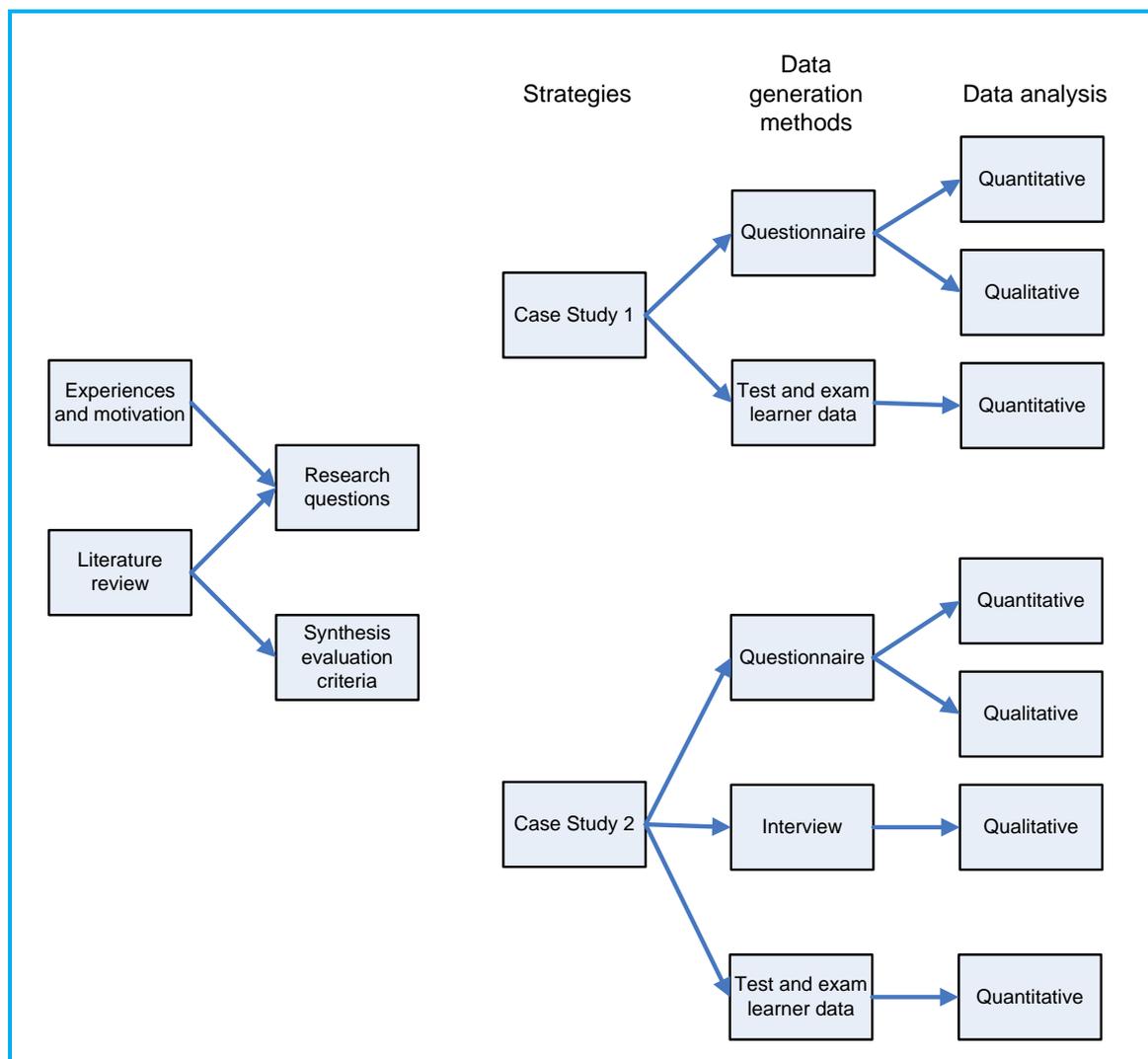


Figure 1.3 Research process (adapted from Oates (2006:33))

Quantitative statistical data analysis (reliability-, descriptive- and inferential analysis) was applied to the closed-ended questions of the questionnaires, as well as to the data from tests and exams, where the performances were compared of the experimental group and the comparison group in each of the two years, 2011 and 2012. Qualitative data analysis (thematic text and description) was applied to the open-ended responses from the questionnaire and the interview transcripts. A detailed discussion is provided in Chapter 4, the Research Design and Methodology chapter.

The next section overviews the mixed methods approach and data generation methods.

1.7.3 Research methods

The data collection instruments were questionnaires and interviews. Furthermore, test and exam data was extracted from DUT's Integrated Tertiary Software (ITS) system, with permission granted by the institution. These are outlined in this section and discussed further in Chapter 4.

1.7.3.1 Questionnaires

The questionnaire contained closed-ended and open-ended questions to elicit quantitative and qualitative responses respectively from the participants regarding the effectiveness of the *Alice* visual programming environment and their experiences with it. The questionnaire further enquired whether engaging with this tool helped them in the challenges of learning object-oriented programming. The questions in the survey emanated from three domains: some were grounded on *Nielsens' ten heuristics*; others emerged from an extensive *literature review on teaching and learning programming*; while yet others were based on the *experience of the researcher*. Each of these domains is briefly described below and will be elaborated in Chapter 4.

The criteria used for the first domain were the general *interface design heuristics*, based on Jakob Nielsens' ten principles for evaluation (Dix, Finlay, Abowd and Beale, 2004). These general principles (or criteria) were used to investigate the usability of the *Alice* visual programming environment, and will be addressed in Section 4.5.2.1 of the Research Design and Methodology chapter:

- (a) Visibility of system status;
- (b) Match between the system and the real world;
- (c) User control and freedom;
- (d) Consistency and standards;

- (e) Error prevention;
- (f) Recognition rather than recall;
- (g) Flexibility and efficiency of use;
- (h) Aesthetics and minimalism of design;
- (i) Recognition, diagnosis and recovery from errors; and
- (j) Help and documentation.

The second domain, namely *teaching and learning programming* focused on:

- (a) the issue of learner attrition, discussed in Sections 2.2 and 2.3.2.1 of the literature study;
- (b) commonly experienced challenges facing learners of OOP, as mentioned in Section 2.3.2 in a literature study chapter; and
- (c) the best teaching practices and techniques used to resolve these challenges, addressed in Section 2.4 of the literature study.

The third domain was based on the *experience of the researcher* and emerged from her personal ten-year involvement in teaching OOP to IT learners.

The questionnaires used for this study are available in Appendices B.2 and C.3.

1.7.3.2 Interviews

As a follow-up to the questionnaire evaluation, semi-structured interviews were conducted during Case Study 2 in 2012 with a selected subsample of the participants. This allowed the researcher to probe themes in more detail. Core questions were posed to interviewees, after which additional information was elicited by querying certain matters. The interviews were characterised by flexibility and informality, and interviewees were able to elaborate their experiences and opinions. The interview questions are available in Appendix C.4.

1.7.3.3 Test and exam learner data

The test and exam data was derived from DUT's Integrated Tertiary Software (ITS) system. This database holds all records of all learners registered at DUT and contains their personal details, together with subject results. This data was used with permission.

1.7.4 Overview of the data collection and analysis process

The studies were implemented over a period of two years, with successive cohorts of learners studying DS2 in 2011 and 2012 respectively. The research was therefore divided into Case Study 1 (2011) and Case Study 2 (2012), preceded by a pilot study in 2011. Reliability, descriptive and inferential statistical analysis was performed using Statistical Package for the Social Sciences (SPSS), and data mining was conducted using Viscovery SOMine. Detailed discussions of these analytical tools are provided in Chapter 4.

Table 1.1 presents an overview of the data collection and analysis processes for this study.

Table 1.1 Overview of the data collection and analysis process

	Pilot Study	Case Study 1	Case Study 2
Academic year	2011	2011	2012
Initial # of participants in experimental group	5	50	55
Final # of participants, due to attrition	5	21	55
Planning Tasks	Design questionnaire for survey evaluation.	Use findings from pilot study to prepare questionnaire for Study 1.	Revise and improve approach and questionnaire using findings from Study 1. ----- Design interview protocol.
Implementation Tasks	Perform pilot study to test survey documentation and process.	Prior to commencement, all participants sign consent form for <i>Alice</i> workshop and questionnaire. ----- Run an <i>Alice</i> workshop concurrently with traditional teaching of the OOP concepts using C# .NET. ----- Disseminate and collect questionnaires in the last two days of <i>Alice</i> workshop.	Prior to commencement, all participants sign consent form for <i>Alice</i> workshop and questionnaire. ----- Run an <i>Alice</i> workshop concurrently with traditional teaching of the OOP concepts using C# .NET. ----- Disseminate and collect questionnaires in the last two days of <i>Alice</i> workshop. ----- Prior to commencement, sample of participants sign consent form for interview session. ----- Conduct post-questionnaire interview sessions three weeks after <i>Alice</i> workshop.

	Pilot Study	Case Study 1	Case Study 2
Data collection instruments		Questionnaire survey ----- Test and exam learner data from ITS system	Questionnaire survey ----- Test and exam learner data from ITS system ----- Interviews
Quantitative data analysis to address appropriate sub-questions		SPSS analysis on responses to closed-ended questions in questionnaire (Addresses Sub-questions 1 & 2) ----- SPSS and Viscovery SOMine analysis on test and exam learner data (Addresses Sub-question 3)	SPSS analysis on responses to closed-ended questions in questionnaire (Addresses Sub-questions 1 & 2) ----- SPSS and Viscovery SOMine analysis on test and exam learner data (Addresses Sub-question 3)
Qualitative data analysis to address appropriate sub-questions		Applied thematic analysis on responses to open-ended questions in questionnaire (Addresses Sub-questions 1 & 2) ----- Quantitative frequency counts of responses to open-ended questions in questionnaire (Addresses Sub-questions 1 & 2)	Applied thematic analysis on responses to open-ended questions in questionnaire (Addresses Sub-questions 1 & 2) ----- Applied thematic analysis on responses to interviews (Addresses Sub-questions 1 & 2) ----- Quantitative frequency counts of responses to open-ended questions in questionnaire (Addresses Sub-questions 1 & 2) ----- Quantitative frequency counts of responses to interviews (Addresses Sub-questions 1 & 2)

Where possible, some of these steps were implemented concurrently. It was the role of the researcher to implement all steps whilst ensuring the validity and reliability of the findings, which is discussed in the Data Collection and Analysis chapters (Chapters 5 and 6), as well as in the Conclusion (Chapter 7).

The next section addresses the ethical considerations adhered to in this study.

1.8 Research ethics

Ethical clearance was granted from both the institution at which the research was conducted and the institution where the researcher is registered for a masters' degree. The Durban University of Technology authorised the researcher to conduct this research at DUT (see Appendix A.1). Furthermore, ethical clearance was granted by the College of Research and Ethics Committee (CREC) of the College of Science, Engineering and Technology (CSET) at the University of South

Africa (UNISA) (see Appendix A.2). The two samples of learners participating in the *Alice* workshops in 2011 and 2012 completed informed consent forms, in which they agreed to complete questionnaires on their experiences with *Alice* (see Appendix B.1 and C.1). The subsample of learners selected to participate in interviews in 2012 were required to complete a further informed consent form (see Appendix C.2) and were also asked to grant permission for the sessions to be tape recorded. The comparison groups in both years were anonymous sets of learners extracted by factors including their pass rates in first year subjects and first time exposure to OOP, and their names were irrelevant. Regarding the licensing of software, *Alice* is an open source teaching tool and is freely downloadable from www.alice.org. For the purpose of this study, the software was installed on the computers in the laboratories of the Department of Information Technology.

The next section discusses the scope of the study.

1.9 Scope of the study

This section addresses the domain of the study, together with the underlying assumptions and the limitations of the research.

1.9.1 Domain of the study

This study spans the areas of teaching and learning of programming; object-oriented programming; visual programming environments and Nielsen's (1994) ten design heuristics. Existing literature discussed in Chapter 2 provided background information on:

- (a) Programming pedagogy;
- (b) Learning theories;
- (c) Challenges facing learners in OOP; and
- (d) Techniques used to teach OOP.

A literature study on visual programming environments was addressed in Chapter 3, and in particular, literature relating to the *Alice* visual programming environment, formed a strong foundation for the study. This set the context and created a general frame of reference for the research processes, in which *Alice* was used extensively and investigated.

1.9.2 Assumptions

It was assumed that:

- (a) Participating learners had successfully completed the pre-requisite subject, Development Software 1, and thereby acquired a uniform level of understanding of the basic programming concepts and problem-solving skills taught during the first level of study.
- (b) Learners selected to participate in either of the case studies therefore had at least one year of IT studies at a higher education level, and were registered for their first attempt at Development Software 2. Learners who had failed DS2 at first attempt were excluded from this study.
- (c) For many of the participants, their vernacular is one of the nine official African languages and English is their second language. Nevertheless, most learners have an adequate command of English, which is the most common medium of instruction in the South African educational system. Furthermore, learners have adequate Mathematical capabilities. These two assumptions are based on the minimum entrance requirement into the ND: IT programme, which is a National Senior Certificate (NSC) with a pass in English and Mathematics.
- (d) The resources required to conduct this study would be available to conduct the workshops. This included laboratories containing operating computers with the open source *Alice* software installed. Furthermore, participants would have access to open lab time to practice their *Alice* programs. The workshops would be held during lunch hours when participants and the researcher were available to attend. Although cost and time considerations existed for the evaluations, these will not be investigated in this study.
- (e) The researcher is a qualified IT professional in the domain of programming and had the ability to learn the required content, so as to efficiently and independently run the *Alice* workshop in her individual capacity.
- (f) The expertise of a statistician would have to be acquired to perform data analysis using SPSS. The data mining software tool, Viscovery SOMine, would be used by the researcher independently and without assistance. Her prior knowledge and expertise in using this tool were adequate to successfully analyse the necessary data.

1.9.3 Limitations of the scope and research methods

The study had the following limitations:

- (a) The scope was limited to DS2 learners in the Department of IT at DUT.
- (b) For both case studies, the group of participants in the workshop, i.e. the experimental group, was restricted to a maximum of 55, mainly due to the capacity of the laboratories in the

Department of Information Technology. Furthermore, the researcher was the only academic involved in conducting *Alice* workshops. (In Case Study 1, attrition occurred, reducing the group of 50 to 21 – see Table 1.1).

- (c) For Case Study 2, all 55 participants in the experimental group were so engaged that they persevered to the end of the workshop. However, only 50 were used for the quantitative comparison of learners' test results and exam results. This was due to the availability of exactly 50 analogous learners in the comparison group who met the appropriate requirements with regard to first year subjects.
- (d) Methods of data collection are restricted to questionnaire evaluations and interviews, as well as learner performance data obtained from tests and the examinations.
- (e) Viscovery SOMine was used only to supplement the quantitative data analysis done by SPSS on the learner data extracted from the tests and exams.

The next section outlines the structure of the dissertation.

1.10 Structure of the dissertation

This study consists of seven chapters, as depicted in Figure 1.4, together with their interrelationships. Brief descriptions of the content of each chapter are presented after Figure 1.4.

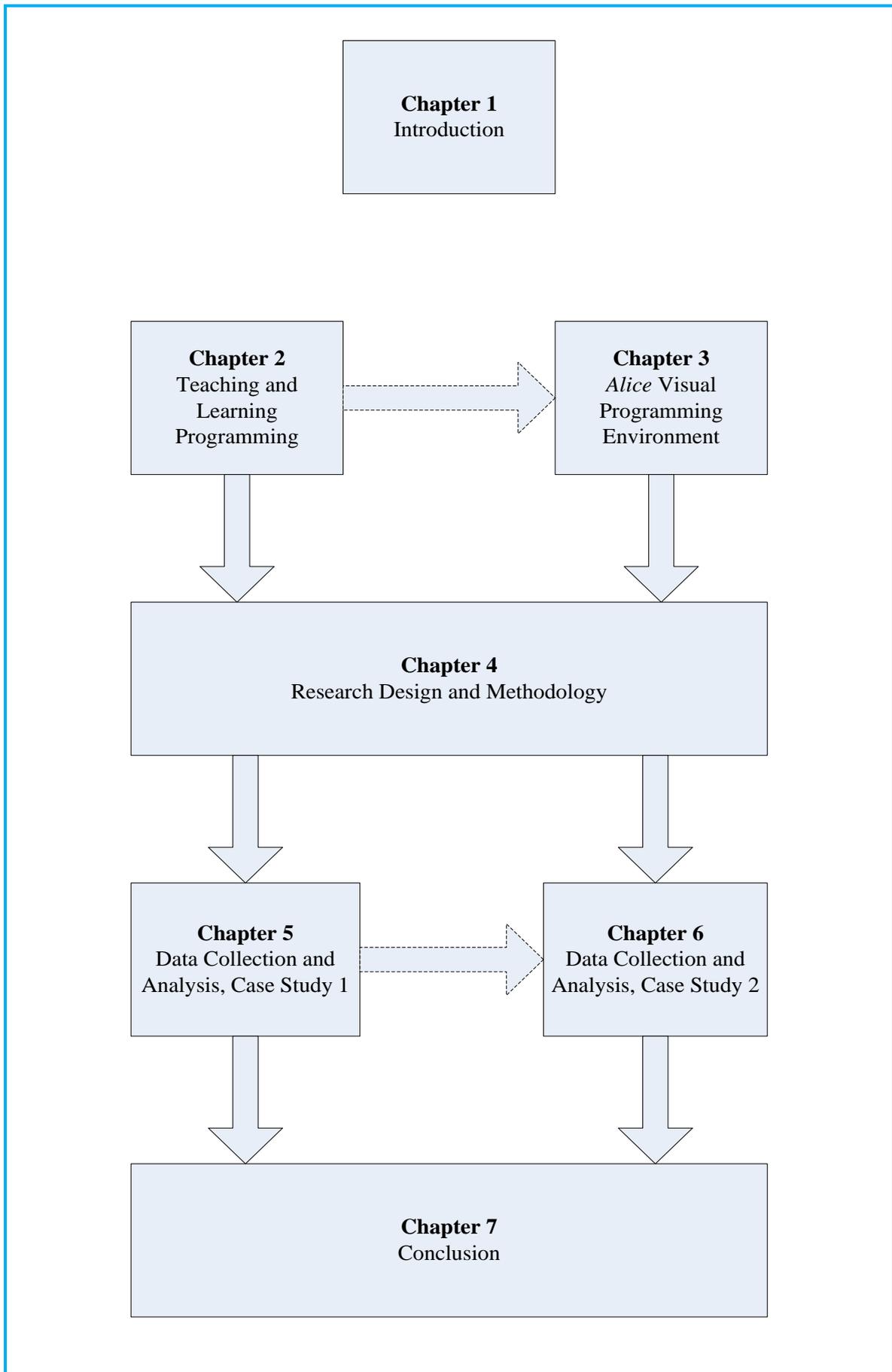


Figure 1.4 Interrelationships between the chapters

Chapter 1: Introduction

The introduction in Chapter 1 presents the background of the study and the real-world problem statement. The rationale explains the need for the study. The research goals and objectives are identified, as well as the research question and associated sub-questions. Chapter 1 also outlines the research design and methodology, the ethical considerations and the scope of the study.

Chapter 2: Teaching and Learning of Programming

The second chapter considers the difficulties experienced by learners when learning how to program. It presents a major literature review of programming pedagogy and the core learning theories. Moreover, problem-solving strategies pertaining to the teaching and learning of programming are highlighted, together with the importance of motivation when learning. An introduction is given to the concept of object-oriented programming (OOP), considering how learners learn OOP and discussing the prominent challenges facing them in this domain. The chapter goes on to discuss characteristics and methods used to facilitate the teaching of OOP languages.

Chapter 3: *Alice* Visual Programming Environment (VPE)

The chapter begins with a definition of a VPE, then goes on to describe a series of commonly-used visual environments developed over the years. The distinctive system features of these VPE's and how each contributed towards improving the teaching and learning of OOP, have been investigated by other researchers. These findings are addressed in Chapter 3. This is followed by a detailed discussion on the 3D *Alice* VPE, the effectiveness of which in facilitating the teaching and learning of OOP, is the subject of investigation in the present study.

Chapter 4: Research Design and Methodology

The fourth chapter sets out the research design and methodology. A brief overview lists each research sub-question against the section where it is addressed in the study. The selection criteria and processes used to identify participants are explained. A discussion follows of the design of each data collection instrument and how the pilot study was used to refine the questionnaire. The process adopted during analysis of the quantitative and qualitative data, as well as the reliability and validity, are described. Finally, the ethical practices employed during this study are outlined.

Chapter 5: Data Collection and Analysis, Case Study 1

Chapter 5 is concerned with recording and analysing the data collected, as well as presenting the main findings of Case Study 1. The analysis framework adopted for the data analysis process is presented. The reliability of the questionnaire is discussed, before the quantitative findings of the closed-ended questions in the questionnaire are given, followed by qualitative findings to the open-ended questions. The quantitative findings of the test and exam learner data, provide insight into the participants' performance after exposure to the *Alice* intervention. A description is given of problems identified in Case Study 1, which are to be improved in Case Study 2.

Chapter 6: Data Collection and Analysis, Case Study 2

The sixth chapter is concerned with recording and analysing the data collected, as well as presenting the main findings of Case Study 2. The chapter follows up on the problems highlighted in Case Study 1 and explains how these issues were resolved. The reliability of the questionnaire is addressed. Results of analysis of the data from both open-ended and closed-ended questions, are provided, before presenting the qualitative findings from the interviews. Learner data from tests and exams is quantitatively analysed and the findings thereof are discussed. Chapter 6 then provides a detailed comparison of the quantitative and qualitative findings in Case Study 2.

Chapter 7: Conclusion

Chapter 7 provides a conclusion to the research. A visual representation of the study is presented and discussed. The effectiveness of the mixed-methods approach employed in this study to answer the three research sub-questions, is evaluated. The research questions are answered by integrating the empirical findings from both Case Study 1 and Case Study 2. The reliability and validity of the findings across both case studies are addressed. Finally, direction and areas for future research are provided, before recommendations are presented.

The next section concludes Chapter 1 with an overall summary.

1.11 Summary and conclusion

This chapter clearly stated the intentions and importance of the study with a well-demarcated explanation of the content and structure. It set the foundation for the entire study, in particular for Chapters 5 and 6, which presents and discusses the results of data collection and analysis in Case Study 1 and Case Study 2, respectively. The research design and methodology discussed in this chapter is further elaborated in Chapter 4. Chapter 1 is also tightly connected to Chapter 7, the Conclusion, which consolidates the findings to the study by providing answers to the research questions.

A background to the study was provided in Section 1.2, while Section 1.3 addressed the real-world problem. The research goals and objectives were identified in Section 1.4. The rationale was discussed in Section 1.5 and motivated the need for the study. Section 1.6 identified the research question and associated sub-questions. Details of the research design and methodology were discussed in Section 1.7, highlighting the mixed-methods strategy of inquiry. The ethical considerations adhered to during this study were explained in Section 1.8. The scope of the study, including its domain and associated assumptions and limitations, was identified and discussed in Section 1.9. Finally, Section 1.10 provided the structure of the dissertation.

The next chapter addresses the domain of teaching and learning of programming.

Chapter 2: Teaching and Learning Programming

2.1 Introduction

Chapter 2 provides a literature study on the teaching and learning of programming. Programming has evolved considerably from traditional procedural programming to the languages and techniques of the object-oriented paradigm (Phelps *et al.*, 2005). This involves a different set of approaches to design and implementation. Learning OOP involves writing programs in a language with a high level of complexity (Carlisle, 2009). Novice programmers tend to find difficulties with the OOP approach, mainly because it is more abstract than the procedural style (Hadjerrouit, 1999).

According to Sajaniemi (2008), students learning OOP experience problems not only in developing the required skills for writing programs, understanding the relevant theory, and debugging, but also in grasping the underlying concepts of object-orientation. Object-orientation involves objects, classes, inheritance, encapsulation, polymorphism, abstraction, modularity and dynamic binding. These concepts are used to represent the problem situation, to design object-oriented models, and to decide on suitable means of implementation (Hadjerrouit, 1999).

This chapter highlights prominent challenges faced by learners of OOP, including the following:

- (a) Improving *learner motivation*, while simultaneously avoiding the extra cognitive implication usually associated with complex problem domains;
- (b) Using algorithmic *logic* instead of the *syntax and semantics* of a particular programming language;
- (c) Providing intrinsic visual *feedback* on learner knowledge assimilation;
- (d) Enforcing hands-on experience in *program problem-solving* and subsequently in the role of translation and reflection of this experience.

The teaching and learning of computer programming forms an integral part of this study, particularly in relation to object-oriented programming. Section 2.2 discusses programming pedagogy, learning theories, problem-solving strategies and motivation, while Section 2.3 highlights prominent challenges faced by learners of OOP. In Section 2.4, possible solutions are proposed in the form of characteristics and methods that can support the teaching of OOP. Finally, the chapter concludes with a summary, provided in Section 2.5.

2.2 Programming pedagogy, learning theories, problem-solving strategies and motivation

The high attrition in programming courses indicates that a large number of learners initially express interest in computing, but that the curriculum or pedagogical techniques tend to drive many of them away (Stiller, 2009). According to Quevedo-Torrero (2009), when learning theories are taught and applied effectively in higher education, it has a positive effect on the performance of novice learners. Learners should therefore be guided in ways of creating mental models that help them represent their knowledge. With a view to addressing the problems experienced by novice learners, Section 2.2 focuses on the programming pedagogy and learning theories that have been developed over the years, with the intention of improving the understanding of learners in the programming domain. Various problem-solving strategies are also addressed, as well as the matter of motivation.

Section 2.2.1 provides a general overview of the challenges facing educators and learners in the teaching and learning of computer programming.

2.2.1 Teaching and learning computer programming

The teaching and learning of programming can be complex (Jenkins, 2001). According to Yu and Yang (2010), many Computer Science learners feel that it is difficult to master a programming language. Several factors contribute to this learning problem, some of which will be briefly addressed.

One of the core problems experienced by many novice learners, as indicated by Gomes and Mendes (2007), is a basic lack of problem-solving abilities among learners. They encounter difficulties in using basic concepts, such as control structures, to develop algorithms that provide solutions to concrete problems. The need for learners to acquire problem-solving skills and the need to learn how to elaborate algorithms, makes it difficult to learn the underlying process of computer programming (Esteves *et al.*, 2008). According to Yu and Yang (2010), the use of tutoring systems or tools on their own cannot replace time spent on practicing and acquiring the range of problem-solving strategies that support effective programming skills.

Many scenarios used in programming problems originate in real-world domains. Learners lacking adequate domain background are likely to experience difficulties when attempting to understand these authentic problems and solve them effectively. Albeit that many learners understand the logic behind problem solving, their challenge lies in expressing solutions to problems in computer

programming syntax (Winslow, 1996). In this regard, Winslow suggests that programming can be divided into four steps:

- (a) Develop an understanding of the current problem;
- (b) Define a solution for the problem. The initial attempt can be in any format, e.g. text-based or a mathematical model, followed by a computer-compatible format such as pseudocode or flow-charts;
- (c) Translate the designed solution into the programming code of a specific language; and
- (d) Test the program and debug it.

For the educator, it can be a difficult task to teach a programming course, and get learners to master the skills being taught (Yu and Yang, 2010). High failure rates and common difficulties expressed among programming learners indicate that traditional teaching methods and studying techniques are far from optimal (Gomes and Mendes, 2007). Salim *et al.* (2010) indicate that the use of traditional methods to teach programming to novice learners is a difficult task.

Wulf (2005) suggests that the two primary goals of teaching computer programming are to ensure that:

- (a) learners have the ability to code programs; and
- (b) as far as possible, the curriculum and the instruction emulate professional practice and culture, so as to produce Information Technology (IT) graduates who become competent IT practitioners.

Tuition in programming should employ strategies that are applied in the real-world workplace, such as team work and pair-programming, as well as non-competitive assessment to encourage learners to work collaboratively as they absorb the material and practice the concepts. The domain of programming is so volatile that practitioners, too, should have access to life-long skills training, so as to remain current within their profession (Wulf, 2005).

To support the teaching and learning of programming, many systems aim to reduce the time required for learners to acquire programming skills and improve their performance (Yu and Yang, 2010). Chapter 3 provides a detailed discussion of several such supportive systems that have been widely adopted and that have had considerable impacts on teaching practices, with particular attention being paid to the *Alice* visual programming environment.

An overview of the theory of programming behaviour is discussed in Section 2.2.2.

2.2.2 Theory of programming behaviour

A programmer can facilitate the writing of a new program or a reader can support his/her understanding of a program by supporting the process with a sound underlying *conceptual organisation*. In the initial process of understanding a problem, the programmer should grasp the basic elements of the real-world problem scenario, including:

- (a) the objects of concern in the problem;
- (b) the properties and relationships of these objects;
- (c) the initial and final state of each object; and
- (d) the operations that are available to implement a transition from the initial state to the final state.

A second type of programming behaviour is called *method-finding*, where a method is a plan or outline of the program to be constructed. A method comprises a set of specifications regarding:

- (a) how real-world information should be represented in the program, i.e. specifying the data structures; and
- (b) how the operations should be performed on these representations to achieve the desired effects, i.e. writing the algorithms.

These methods should be hierarchically organised, with smaller methods forming components of the larger ones (Brooks, 1999).

Following this introduction of the need to apply learning theories and create models, Sections 2.2.3 to 2.2.5 present discussions of each of three major learning theories, namely the *cognitive*, *constructivist* and *behavioural* approaches to teaching and learning. It is pointed out that the *Alice* VPE used in this study promotes constructivist learning, a matter that will be addressed in Section 7.4.1.4 of the Conclusion chapter.

2.2.3 Cognitive approach

Cognitive psychology emphasises knowing over doing and often relates to non-observable outcomes. Cognitive psychologists have established numerous understandings of the nature of knowing, including *schemas*, *information processing* and *semantic networks* (de Villiers, 2005). Computer programming involves cognitive skills more than it involves the acquisition of a body of knowledge (Hadjerrouit, 2008).

“Various activities should occur during programming to meaningfully construct, explicitly reflect on, and critically select appropriate knowledge, skills and strategies to understand, design, code and

test high quality programs” (Havenga, Mentz and de Villiers, 2008:7). This involves the carefully balanced integration of cognitive, metacognitive and problem-solving techniques. Havenga, de Villiers and Mentz (2011) further point out that high performers in programming employ a variety of mental activities and supportive strategies. In this way they use a range of cognitive, psychological and reflective activities in the programming process.

According to Quevedo-Torrero (2009), information processing utilises memory manipulation and information retrieval to facilitate the processing of information. The three types of memory, namely sensory, short-term and long-term, and their respective roles are considered in cognitive information processing theory. When a novice learner receives new information, it reaches the short-term memory. For long-term memory, a learner must possess prior knowledge to use in the assimilation of the new information. The learner actively uses prior knowledge and integrates the new material with it (and Trollip, 2001).

“A schema is a cognitive construct that organises the elements of information according to the manner with which they will be dealt” (Sweller, 1994:296). Semantic networks refer to pieces of information, or nodes, that connect to many other pieces of information to form interrelated information and meaning (Alessi and Trollip, 2001).

The components of cognitive learning focus on activities such as critical thinking and analytical reasoning. This paradigm emphasises an understanding of concepts and their associated relationships. It is suggested that for understanding to improve, learners need to comprehend relationships between concepts, deconstruct information and rebuild it with logical connections (Hadjerrouit, 2008). For novice programming learners, a cognitive understanding of the actual programming problem is a fundamental baseline for program comprehension and writing. A strong grasp of the basic programming concepts is essential for learners to be able to write any good program (Matthews *et al.*, 2009).

Phelps *et al.* (2005) indicate that certain computer educators utilise Bloom’s Taxonomy to effectively assess cognitive learning. According to Starr, Manaris and Stalvey (2008), Bloom’s taxonomy was motivated partly by the fact that many exam questions required only rote memorisation and regurgitation of knowledge on the part of learners. Such questions cannot ascertain how well a learner has, in fact, mastered the concepts being tested. Bloom’s taxonomy identifies six levels of cognitive learning objectives, namely knowledge, comprehension, application, analysis, synthesis and evaluation (Alessi and Trollip, 2001). Based on Starr *et al.*

(2008), Table 2.1 provides a brief explanation of the levels that comprise Bloom’s taxonomy, and contextualises it with sample questions for the OOP concept of inheritance.

Table 2.1 Levels of Bloom’s Taxonomy with sample questions based on the OOP concept of inheritance (adapted from Starr, Manaris and Stavley (2008:262))

Level	Explanation	Sample Question
Recall of knowledge	The learner’s ability to recite memorised information about the concept.	What is inheritance?
Comprehension	The learner’s ability to explain the concept using his or her own words.	How is inheritance similar to the relationship between a father and son?
Application	The learner’s ability to apply the concept to a particular situation.	Write a method in the <i>Sailboat</i> subclass to calculate the cost of building a sailboat. Use the boat dimensions inherited from the <i>Boat</i> superclass.
Analysis	The learner’s ability to separate the material or concept into components so that their organisational structure is better understood.	Draw a UML (Unified Modeling Language) class diagram to depict the relationship between the <i>Boat</i> superclass and <i>Sailboat</i> subclass. Write down the required fields, properties and methods.
Synthesis	The learner’s ability to assemble parts together to form a whole, placing emphasis on creating a new meaning or structure.	Expand the <i>Boat Designers</i> business process to accommodate other modes of transport, such as aeroplanes.
Evaluation	The learner’s ability to judge the value of materials or ideas.	Given two programs that demonstrate the use of inheritance, which one is better and why?

Furthermore, Starr *et al.* (2008) suggest that Bloom’s taxonomy can consist of three meta-levels, through which learners must proceed when introduced to a new concept, namely:

- (a) Beginner level – Recall and Comprehension;
- (b) Intermediate level – Application and Analysis;
- (c) Expert level – Synthesis and Evaluation.

2.2.4 Constructivist approach

Constructivism is a learning theory which claims that learners actively construct knowledge, rather than merely receiving and storing knowledge transmitted by the teacher (Ben-ari, 1998). It is a learner-centered pedagogy, which typically entails experiential discovery learning through exploration (Wulf, 2005). This approach to learning requires that learners demonstrate their competency and ability to construct, interpret and solve real world problems using their own acquired knowledge. The focus of this approach is a shift from instructor-driven interventions to learner-centred activities, as it is assumed that learners achieve higher success when they are forced to discover and learn things for themselves (Hadjerrouit, 2008).

A study conducted by Ben-ari (1998) on constructivism suggests that knowledge is acquired recursively. This occurs when sensory data and prior knowledge are combined to form new cognitive structures which consequently provide the basis for further construction. Ben-ari states that the process of learning is active, whereby learners construct knowledge with guidance from teachers as well as by considering responses from other learners. Radical constructivism refers to an individual learner's construction of knowledge, whereas social constructivism refers to groups of learners.

2.2.5 Behavioural approach

According to behavioural psychology, learning can be defined as observable changes in the learner. This is known to occur in response to changes in the environment (Alessi & Trollip, 2001). Behaviourism concentrates on shaping the learner's behaviour (de Villiers, 2005). Research conducted by Thorndike in the first half of the 20th century concluded that the use of rewards and punishments can modify behaviour. This approach is termed *operant conditioning*. B.F. Skinner refined the work of Thorndike, which gave rise to the behavioural school of psychology and learning. Skinner enforced the principle of *intermittent reinforcement*. It was observed that certain behaviours increase rapidly in frequency when consistently rewarded. Conversely, behaviours tend to be extinguished rapidly when the reward ceases. In between the above extremes, lie behaviours that are rewarded intermittently. Albeit these behaviours increase in frequency at a slower pace, they are longer lasting or resistant to extinction (Alessi & Trollip, 2001).

Iran-Nejad and Homaifar (2005) recognise the efforts of behaviourists during that period, in achieving landmark successes in the application of the principle of *systematic observation* to highly controlled learning situations.

When teachers acknowledge the accomplishments of learners in achieving a desired learning outcome, learners often repeat the behaviours that contributed to their success. As such, teachers ought to clearly define their expectations for each learning outcome. It must be borne in mind that reinforcing good learning practices helps increase learner achievement (Quevedo-Torrero, 2009).

Some of the problem-solving strategies relevant to providing support for learners in a programming domain are explained in Section 2.2.6. These include the top-down, bottom-up, trial-and-error, hill climbing, systematic and as-needed problem-solving strategies.

2.2.6 Problem-solving strategies

According to Kiesmüller (2009:17:1), “one possible approach to improving both the teaching and the learning process is to specify the knowledge concerning the learners’ individual problem-solving strategies, their solutions, and their respective quality”. Previous research conducted on problem-solving, as cited by Kiesmüller, defines a problem as comprising an unrequested initial state and a requested goal state, with intermediate problem states building the problem space between them. The problem-solving process entails finding a correct path through the problem space from the initial state to the goal state. Some of the major problem-solving strategies will be further addressed, bearing in mind that, for each, the learner aims to reduce the difference between the initial state and the goal state.

Of the the problem-solving strategies considered below, the researcher found the bricolage (trial-and-error) approach to be the most prevalent amongst the learners who participated in this study. Further mention of the bricolage strategy will occur in Chapters 4, 5, 6 and 7.

2.2.6.1 Top-down approach

The top-down approach addresses the global scope of the problem, by further decomposing it into smaller sub-problems (Storey, 2006). The success of this strategy depends on the ability to identify all intermediate states before starting the solution, i.e. before finding the correct way through the problem space. This process employs the ‘divide and conquer’ strategy of computer science (Kiesmüller, 2009). Learners are able to edit the aforementioned states to a problem that is structured as a whole (Kiesmueller, Sossalla, Brinda and Riedhammer, 2010).

2.2.6.2 Bottom-up approach

Conversely, the bottom-up approach is described as a strategy that requires programmers to focus initially on details of the individual subproblems that are combined into higher-level abstractions (Storey, 2006). Learners are expected to solve every single subproblem as it is identified, even before searching for other subgoals. The solution of the problem is therefore complete after the final subproblem has been solved. This success of this strategy depends on subgoals not being intertwined in any way (Kiesmüller, 2009).

2.2.6.3 Bricolage (trial-and-error) strategy

Kiesmüller describes this strategy as a learner trying to find the correct way through the problem space (sometimes aimlessly) by attempting different possibilities one by one. The bricolage problem-solver (a bricoleur) is described as an individual who uses readily available materials in a trial-and-error manner to construct a solution to the problem at hand. The bricolage approach consists of two phases. In the hypothesis formulation and testing phase, learners gain familiarity with a provided program. In the problem-solving phase, learners try to adapt the given program to achieve a new behaviour (Stiller, 2009).

Hadjerrouit (2008) states that bricolage is the most common approach to programming among novice learners. In this approach, learners develop their programs incrementally, working directly on the computer, with the result that analysis and design are neglected. The gradual development of the program by testing it with various inputs creates cognitive obstacles. Hadjerrouit expresses reservations regarding this practice, stating that novice programmers develop misconceptions about computing. This in turn makes it difficult for the learner to understand the actual functioning of a program or the construction of an algorithm.

Although Ben-ari (1998) believes that concrete thinking and bricolage can support learners in learning how to program, he argues that bricolage is not an effective epistemology for professional programming. Consequently, a learner who makes exclusive use of such techniques does not qualify to work on embedded software and operating systems, which demand the ability to create and test abstract hypotheses.

2.2.6.4 Hill climbing strategy

This is a forward-thinking strategy where learners remain focused on a single step of the program. Learners have an idea of how the program should work in general. They subsequently start the program execution and actively stop the execution when they realise that the currently tackled sub-problem is either correct or incorrect. Mistakes are corrected, if necessary, before proceeding to the next step. This process is repeatedly performed until the final solution is found (Kiesmueller *et al.*, 2010).

2.2.6.5 Systematic and as-needed strategies for maintenance

Studies conducted by Brade, Guzdial, Steckel and Soloway (1992) identified two basic macro-strategies to globally categorise the use of documentation when conducting software maintenance, namely, a systematic strategy or an as-needed strategy.

Programmers who employ the systematic strategy, use multiple forms of simulation to trace the flow of the entire program, usually commencing at the start of the program and documentation (Brade *et al.*, 1992). In order to attain a comprehensive understanding of the program prior to code modification, this strategy utilises extensive symbolic execution of the data and control flow between subroutines (Koenemann and Robertson, 1991).

Programmers who use the as-needed strategy, choose to read and study only the segments of the program code and documentation that they envisage would be valuable for constructing their enhancement (Brade *et al.*, 1992). The as-needed strategy thereby minimises understanding of the program (Koenemann and Robertson, 1991).

When comparing both strategies, Brade *et al.* (1992) observes that the systematic strategy advantageously has minimal support needs, due to its tendency to lead to a correct enhancement. Conversely, programmers who employ the as-needed strategy require support in locating delocalised plans, so as to allow them to make a correct enhancement. This is mainly due to programmers restricting their grasp to those segments of code that they believe are useful for the task. Consequently, their understanding of the program is only partial and this can result in misconceptions or errors (Koenemann and Robertson, 1991).

Section 2.2.7 addresses the importance of learner motivation in achieving success.

2.2.7 Motivation

Motivation is considered to be a pivotal element in successful learning. This is even more pertinent in a practical discipline like computer programming where learners must possess a degree of self motivation to spend time practicing programming. Learner motivation must be sustained, even when explicit assessment credit is absent (Jenkins, 2001).

Jenkins identifies four types of motivation:

- (a) *Extrinsic* – the successful completion of a course is rewarded by the value in one's career and other associated benefits that will follow;

- (b) *Intrinsic* – the programmer possesses a genuine and deep interest in the subject of computer programming;
- (c) *Social* – the need for recognition from a third party whose impressions are valued; and
- (d) *Achievement* – the degree of personal satisfaction in achieving success.

Learners who are intrinsically motivated to learn have the capacity to reach a stage of achievement, depending on their ability to overcome the challenges they face. There is a need for some level of success to be attained so that intrinsic motivation can be sustained, otherwise learners might focus their attention merely on assessment results or other external signs of achievement (Carbone, Hurst, Mitchell and Gunstone, 2009).

A close connection exists between creativity and intrinsic motivation, as well as learner engagement. It is recognised that both external affective aspects (initial ability to assimilate knowledge) and internal affective aspects (ongoing attitude and endurance) are influenced by learners' emotions and values. Instructional methods that promote creativity endeavour to adopt both forms of affective aspects within learning. This can be achieved by seeking to apply concomitant innovative instructional techniques to engage learners. In doing so, the learner's cognitive-affective bonds are strengthened (de Villiers, 2005).

It is challenging to acquire the skill of programming, and considerable practice is required to attain a level of expertise. For effective learning, some of this practice will have to be self-directed. A skillful instructor would encourage and motivate learners to do this, since adequately motivated learners engage more effectively (Jenkins, 2001). Inadequate programming skills have a negative effect on motivation while, conversely, the presence of personal programming skills increases motivation. It is the role of the instructor to nurture, exemplify and facilitate the desires of the learners so as to steer their motivation in the right direction (Carbone *et al.*, 2009).

According to Law *et al.* (2010:220) “individual attitude and expectation”, “clear direction” and “reward and recognition” have the greatest motivating effect on learning. Based on this, a challenge to the educator is how the teaching and learning activities can be organised to effectively strengthen these motivating factors, so as to enhance the learning efficacy of the learners.

The next section addresses the learning of OOP, highlighting the core challenges facing learners of OOP.

2.3 Learning object-oriented programming

Programming languages and techniques have undergone considerable evolution from imperative languages to those which adopt object-oriented design principles and implementation (Phelps *et al.*, 2005). In most cases, object-oriented programming entails learning a language with a greater degree of complexity (Carlisle, 2009). The object-oriented approach is considered to be more abstract than the procedural approach. Consequently, most novice learners experience difficulty with learning this approach (Hadjerrouit, 1999). Section 2.3.1 follows with an explanation of object-oriented programming.

2.3.1 What is object-oriented programming?

“The essence of object-oriented programming is to model systems of real entities with the goal of separating their internal structure from their external, visible interactions” (Ralston, Reilly and Hemmendinger, 2000:1279). As mentioned in Section 2.1, object-orientation involves the concepts of objects, classes, inheritance, encapsulation, polymorphism, abstraction, modularity and dynamic binding. These concepts are used to:

- (a) represent the problem situation;
- (b) design object-oriented models; and
- (c) decide on suitable means of implementation (Hadjerrouit, 1999).

OOP is more comprehensive and specific than structured programming in its approach to solving a problem. OOP relates to a specific ‘form of structure’ that is associated with objects, while structured programming is concerned with ‘structure’ in general (Ralston *et al.*, 2000). “Objects are collections of operations that share a state” (Reilly, 2004:567). “An object is a thing that contains data and instructions for acting on that data” (George, Batra, Valacich and Hoffer, 2004:45). The authors further describe an object as something perceptible by the mind or intelligible. It contains properties that define its characteristics and behaviours that are indicative of its actions. For example, a ‘door’ object contains properties such as ‘width’, ‘length’ and ‘colour’, and behaviours such as ‘open’ and ‘shut’. Research reveals that learners learning OOP experience difficulties not only in developing their skills to write programs, understanding the underlying theory, and being able to debug programs, but also in comprehending the basic concepts of object-orientation (Sajaniemi *et al.*, 2008).

According to de Villiers (2005), one of the requirements of OOP is the construction of objects. Operations based on the occurrence of an event, can be implemented to trigger the execution of

these objects. A sequential approach is not mandatory and systems are open-ended. These practices have a notable impact on the ways in which users behave within those environments. Rich multimedia environments such as these promote flexibility of use that is congruent to constructivist and cognitive learning theory models. The initiative taken by learners elicits a corresponding response from the environment and thus suitably lends itself to problem-based learning and inquiry-based learning.

As mentioned in Section 2.1, novice programming learners are faced by challenges with respect to object-oriented programming. These challenges are discussed in detail in Section 2.3.2.

2.3.2 Challenges faced by learners of object-oriented programming

According to Salim *et al.* (2010), the domain of programming would appear less confusing to novice programming learners if the most important generic concepts of programming were clearly explained and understood before learners encounter the intricacies of actual programming languages and environments.

Immediate exposure to the terminology of programming syntax, such as do-while, if-then, switch-case, and for-next, may intimidate learners and result in poor performance, as well as learner attrition. The abstract concepts behind these words need to be articulated in tangible ways that learners can visualise and relate to. Learners should be exposed to testing all of these concepts separately and with sufficient time to grasp the meaning and effect of each one individually. This should improve the use and understanding of programming constructs, as well as the learner's ability to think abstractly. Salim *et al.* (2010) suggest that many novice programming learners lack the ability to express their creative thinking in terms of programming abstractions, because they do not grasp programming concepts on an abstract level. This may lead to learners writing code without really understanding each and every line in their code.

A study was conducted by Sajaniemi *et al.* (2008) on learners' grasp of OOP execution by requesting them to illustrate the program state at a given moment during execution time. The results indicated that learners lacked an explicit understanding of the dynamic aspects of object-oriented programs. However, positive differences were found in learners' representations after they had used different program visualisation tools.

Wright and Cockburn (2000:168) believe that inadequate emphasis has been placed on investigating the three fundamental programming concepts involved in learning, namely:

- (a) Reading – “the process of reviewing the static representation of a previously recorded program”;
- (b) Writing – “the process of recording or encoding a series of program actions”; and
- (c) Watching – “the process of mapping between the learners’ understanding of the program and the observed dynamic behaviour of the running program”.

Wright and Cockburn further suggest that such activities tends to support and promote learning naturally. Moreover, they expedite the transition from rudimentary to more advanced programming concepts. Computers, by their nature, are designed to provide an interactive experimental platform for the learner. The behaviour of a program expressed by a learner in written form, is a dynamic watchable form of the readable program.

The three activities of reading, writing and watching have been explicitly distinguished. This chapter will further expound the learning challenges that learners are faced with when mapping between their grasp of programming concepts and the mechanisms provided by the programming environment. Some of these challenges were mentioned in Section 2.1. As a means to overcome these challenges, a study conducted by Kordaki (2010) proposed using an open problem-solving computer learning environment called the Learning environment for programming using C using Geometrical Objects (*LECGO*). The tool was designed on constructivist and social perspectives with regard to knowledge construction, while emphasising the role of the challenges highlighted in Section 2.1. These challenges are repeated below, with references to relevant literature sources.

- (a) Lack of motivation for programming (Esteves *et al.*, 2008; García-Mateos and Fernández-Alemán, 2009; Dann *et al.*, 2009);
- (b) Complex syntax and semantics (Winslow, 1996; Gomes and Mendes, 2007; Dann *et al.*, 2009);
- (c) Immediate feedback and identifying the results of computation as the created program runs (Wright and Cockburn, 2000; Gálvez *et al.*, 2009; Dann *et al.*, 2009); and
- (d) Difficulties in understanding compound logic and the application of algorithmic problem-solving skills (Gomes and Mendes, 2007; Esteves *et al.*, 2008; Dann *et al.*, 2009).

A discussion of each of these challenges faced by learners in object-oriented programming follows in Sections 2.3.2.1 to 2.3.2.4.

2.3.2.1 Lack of motivation for programming

Motivation has been discussed in Section 2.2.7 as a key factor for success in learning how to program. The lack of motivation for learning object-oriented programming in particular, can be seen as a challenge. This section addresses these concerns and highlights some suggestions made by other researchers with a view to improving motivation.

There are two main causes of attrition in Computer Science studies, namely the implicit complexity of the subject matter and a lack of motivation amongst learners (García-Mateos and Fernández-Alemán, 2009; Moskal *et al.*, 2004). Learners who adopt a traditional approach to studying tend to feel demotivated as they cannot see the need for learning how to write code (Esteves *et al.*, 2008).

Wang, Mei, Lin, Chiu and Lin (2009) indicate that the text-based mode of writing object-oriented programs and the corresponding text-based output generated by these programs, tend to create an incorrect learner perception that programming is dull and uninteresting. Concern over declining enrolments in computer science and the need to improve learner success rates have led educators to develop approaches that make OOP more attractive to potential learners (Cliburn, 2008).

These long-standing challenges have drawn the attention of researchers throughout the years. Studies conducted in 2000 by Wright and Cockburn on support for the reading, writing and watching that can be offered by an educational programming environment, emphasise the importance of presenting learners with engaging, motivating and familiar programming domains.

A more recent study conducted by Kiesmüller in 2009 expresses that learners are motivated by visual learning environments and can gain competence in completing even complex tasks after only a few lessons, because the visualisations support ease of learning. Wang *et al.* (2009) point out that the lack of learner motivation has led some instructors to adopt the use of alternative programming languages that foster easier writing of programs, without detracting from the core programming concepts. This approach can also provide more attractive output. *Alice*, which is discussed in Chapter 3, was considered to be such an alternative.

2.3.2.2 Complex syntax, logic and semantics

Research on the psychology of programming reports that one of the challenges faced by novice programmers in learning a new programming language, is that they must learn new syntax and semantics (Winslow, 1996; Herbert, 2011). In OOP education, the learners' ability to understand the abstract model of a programming language is deemed more important than learning the syntax

of the language. Learners often spend considerable time focusing on the superficial syntax of a text-based programming language, and debugging the syntactical errors within their programs. Consequently, the learners' motivation to program decreases, and they are distracted from comprehending the essence of programming (Nagaoka, Osawa, Mochizuki, Takahashi, Nishina and Saito, 2000). Novice programmers often experience difficulty in translating their intentions into syntactically correct statements that the computer can interpret (Kelleher and Pausch, 2005).

Gomes and Mendes (2007) indicate that programming languages have complex syntactic characteristics promoting their use in a professional context. They were not designed with motivational instruction in mind. In most cases, programming courses focus on coding, syntax and specific characteristics of the respective programming languages. Learners concentrate on these ancillary details rather than on the fundamental skills required to develop the underlying algorithms. This focus on implementation details means that many learners fail to comprehend the essential algorithmic model that transcends particular programming languages (Joint Task Force on Computing Curricula, IEEE, 2001). Moreover, due to time spent dealing with syntactic complexity and detailed coding issues, learners spend insufficient time grasping the underlying concepts of object-orientation or problem-solving.

Furthermore, the learning style of most learners does not fit well into the textual nature of the majority of programming environments (Carlisle, 2009). Wright and Cockburn (2000) believe that a 'gulf of expression' exists between programmers' internal mental models of the expected program outcome and the symbols and syntax of the programming language. This gulf results in a compromise in language design between the level of abstraction of the language's symbols and its potential generality.

2.3.2.3 The need for immediate feedback and identifying results of computation as a created program runs

Mitrovic, Martin and Mayo (cited by Gálvez *et al.*, 2009) point out that feedback obtained immediately after an error, is the most effective pedagogical action. Feedback consists of pieces of knowledge that help learners to eliminate misconceptions or to learn concepts they have not grasped previously (Gálvez *et al.*, 2009). The ability to see an immediate display of the results of an action allows learners to quickly gauge the correctness of their idea. An incorrect idea will require the learner to correct and rethink their solutions, thus honing their critical thinking capacity (Esteves *et al.*, 2008).

According to Garcia-Mateos and Fernandez-Aleman (2009), there is a positive trend in that current educational tendencies are centred on the learner’s point of view, rather than on that of the educator. The intention is to create independent, reflective and life-long learners. The new methods should stimulate learners’ interest and offer appealing material, fair assessment and relevant feedback.

Wright and Cockburn (2000) indicate that a ‘gulf of visualisation’ occurs when programmers experience difficulties mapping their mental model against the observed behaviour of the program when it executes. The representation of the program acts as a substitute for the users’ mental mode of the program. Not only do learners who are novice programmers experience this rift in visualisation, but it is also a common problem for expert programmers who make use of debuggers and other tools to counteract the challenges of visualising the dynamic internal behaviour of the program.

According to Wright and Cockburn (2000), there is an inadequacy in *Alice*, in that the environment offers insufficient support for watching programs run. The entire program or a call to a procedure runs as a single step, thus hindering the programmer from being able to trace the correlation between the system state and each program instruction.

2.3.2.4 Difficulties in understanding compound logic and the application of algorithmic problem-solving skills

In writing programs, a programmer generates algorithms, which are sets of steps that must be executed to perform tasks. The programmer must then translate the algorithms into a computer programming language (Gaddis, 2011). Research conducted on programming psychology reveals a further challenge experienced by novice programmers, namely that in programming a solution to the problem at hand, the learner has to convert the paper-based solution into a computer program. At times the novice is capable of solving a problem manually and cognitively, but is incapable of writing a program to solve the same problem. An example of this occurs in solving quadratic equations (Winslow, 1996).

Learners seem to better grasp the importance of understanding basic problem-solving techniques when the relevance of finding patterns and the sequential development of algorithms are emphasised. They are able to apply such techniques in a systematic way to a wide set of problems. This approach can be effective in presenting language syntax to learners when the need for a

particular programming statement and its associated semantics has been sufficiently motivated beforehand (Winslow, 1977).

In some cases, the main reason why learners struggle to create algorithms, is that they are unable to solve common problems, quite apart from programming such solutions. Programming requires a sound grasp of both practical problem-solving techniques and the computer programming language (Gomes and Mendes, 2007). When learners lack fundamental problem-solving abilities, they are incapable of decomposing a complex and large programming task (Carbone *et al.*, 2009).

According to Carlisle (2009), previous studies indicated that learners have a preference for visual representations. Their success in learning how to program is greater when introduced to programming concepts by an iconic or flowchart environment. Carlisle consequently conducted a study using an iconic programming environment, named *RAPTOR*, which was designed particularly to help learners visualise the concept of a class and to reduce complex syntax. The environment allows learners to combine basic graphical symbols in order to generate algorithms. Learners use a UML designer to create their class hierarchy, following which flowcharts are used to represent method bodies. The end-result are programs that can then be executed in the environment, either in continuous-play mode or step-by-step. During this process, the environment provides a visual display of the location of the currently executing symbol, together with the values of all variables. The system also features a simple graphics library. *RAPTOR* is an open source tool that supports OOP, and includes the basic concepts of encapsulation, inheritance and polymorphism.

The section that follows, discusses the teaching of OOP with particular attention to the educational techniques used to address the above-mentioned challenges.

2.4 Techniques to help teach object-oriented programming

According to Dann *et al.* (2009), there have been relatively few innovations in the teaching of programming in the last 30 years, even though many programming courses are frustrating to learners. By contrast, Salim *et al.* (2010) argue that many research efforts have been directed to improve the teaching methodologies used in introductory programming courses.

Sections 2.4.1 to 2.4.4 are possible solutions to counteracting the challenges identified in Section 2.3.2, in the form of characteristics and methods that can support the teaching of OOP.

2.4.1 Algorithmic thinking and expression

“Algorithmic thinking and expression involves the ability to read and write in a formal language” (Dann *et al.*, 2009:13). For the majority of learners, the textual nature of conventional programming languages provides a non-intuitive framework for learning about the object-oriented paradigm and algorithmic thinking (Carlisle, 2009). According to Piaget (1973), as cited by Wright and Cockburn (2000), educators have made efforts to comprehend the worth of teaching programming as a tool for promoting constructivist learning and structured thinking.

2.4.2 Abstraction

Programming demands a high level of abstraction (Gomes and Mendes, 2007). Abstraction entails learning how to communicate complex ideas simply and to decompose problems logically (Dann *et al.*, 2009). “One thing all programmers do, whether they realise it or not, is use something called abstraction. Abstraction happens when we view something in general terms without focusing on its concrete details” (Vickers, 2009:12).

Bennedsen and Caspersen (2008) indicate that the majority of computer scientists view abstraction as a defining characteristic of their discipline. It is considered essential for an individual who strives to achieve success in Computer Science to demonstrate competence in performing abstraction. Clearly, abstract thinking and abstraction are core concepts in computing and vital components of learning it. For Computer Science education, it is compulsory to develop the learners’ abstractive skills. In their study, Bennedsen and Caspersen proposed that the ability to perform general abstraction is a measure of success for learning Computer Science. Their hypothesis in support of this finding, was confirmed.

2.4.3 Objects-first strategy

According to Cooper *et al.* (2003), the report of the Joint Task Force on Computing Curricula, IEEE (2001) summarised four different approaches to teaching introductory Computer Science. It was noted that the ‘programming-first’ approach was widely adopted in North America. The report discusses three implementation strategies required to achieve a programming-first approach, namely objects-first, functional-first, and imperative-first.

Objects-first emphasises the most fundamental principle of OOP and design. The strategy introduces the concepts of inheritance and objects at an early stage before addressing the traditional control structures. However, the object-oriented design remains the overarching focal point.

Cooper *et al.* (2003) further state that, due to the gradual transition between simple and advanced programming projects and examples, a resulting gentle learning curve provides more time for learners to build or assimilate knowledge incrementally.

The challenge of the objects-first strategy

An objects-first teaching strategy is intended to involve learners in working with objects. This means that they must start working immediately with objects and classes, encapsulation (private and public data) and methods (modifiers, constructors, helpers, accessors). Consequently, many learners are faced with mastering the basic concepts of variables, data types, values and references, at the same time as dealing with details of syntax. Learners often find this to be frustrating. The implementation of event-driven concepts to support interactivity with graphical user interfaces adds further complexity to the learning process.

Through various studies, Phelps *et al.* (2005) identified concerns with the objects-first approach, namely:

- (a) Apart from being difficult to teach, course materials based on the objects-first approach are difficult to develop;
- (b) Educators have demonstrated a lack of experience in relation to objects-first techniques, which is a cause for concern;
- (c) Object-oriented thinking may be unsuited to the type of problems usually addressed in introductory programming courses;
- (d) Novice learners do not possess the mental framework required to grasp OOP;
- (e) The pace of the transition from a program that initially does not exist to a realised application, with independent relationships and objects, is a concern;
- (f) The term ‘objects-first’ within the context of programming is not properly understood;
- (g) The need for instruction in imperative techniques to coincide with the development of the objects-first techniques, has proved highly complex for learners; and
- (h) It is overwhelming for novice learners to deal with additional complex libraries and graphical interfaces.

Success in deploying an objects-first approach

With a view to addressing these concerns, Phelps *et al.* (2005) identified the following guidelines relating to success with the objects-first approach:

- (a) Novice learners have frequently been exposed to gaming systems as their initial encounter with computing devices. Consequently, they experience difficulty relating to examples, even simple ones, that are textual in nature;
- (b) It is recommended that learners should first interact with design concepts before being required to code syntax;
- (c) It may be more effective for learners initially to make minor changes to existing complex environments and to recognise those changes, rather than having to create objects and complete programs in the early stages of programming;
- (d) Learners respond well to immediate feedback. Early feedback helps to internalise cause and effect in the development of programs;
- (e) Learners should gain early exposure to object-oriented concepts, such as inheritance, encapsulation and polymorphism, together with an understanding of relationships between classes, such as dependency, association and composition; and
- (f) Being introduced to design patterns at an early stage, facilitates the consideration of choices in design when constructing programs.

Providing learners with an interactive, collaborative, and rich virtual environment that encourages their programming experience, is a sound way to minimise the negative concerns and maximise the positive observations associated with the object-oriented paradigm and objects-first approach to programming. This approach is explored further in Chapter 3.

2.4.4 Three-dimensional animation authoring tools and visualisation

Visualisation is considered as a mechanism that concretises teaching and provides learners with visual feedback to reinforce their understanding (Montero, Díaz, Díez and Aedo, 2010). Visualisation and animations have been utilised to improve learners' grasp of algorithms (Sajaniemi *et al.*, 2008). Learners are able to comprehend programming concepts better when provided with a visual representation (Carlisle, 2009). According to Gomes and Mendes (2007), many tools for solving programming complexities use animation and simulation techniques, aiming to take advantage of the potential of the human visual system. Gomes and Mendes point out that, in comparison to static formats, animations can positively influence the understanding of the inherently dynamic concepts used in programming.

Developments in technology have raised fresh opportunities in the educational domain, such as the 3D virtual world. These innovations provide instructors with an accessible means of creating compelling and rich 3D contexts for situating learning, as well as communication tools to aid collaboration and discourse (Dickey, 2003). Three-dimensional animation provides a stronger object visualisation and a meaningful, flexible context for assisting learners to perceive object-oriented concepts (Cooper *et al.*, 2003). “Our goal is to engineer authoring systems for interactive 3D graphics that will allow a broader audience of end-users to create 3D interactive content without specialised 3D graphics training” (Conway, Audia, Burnette, Cosgrove and Christiansen, 2000:486).

Three-dimensionality provides a sense of reality for objects. In the 3D world, learners may write new methods to make objects perform animated tasks. Animations can create a meaningful context for understanding classes, objects, methods, and events. Three-dimensional animation meets the challenge of an objects-first approach by:

- (a) Decreasing the complexity of details that the novice programmer has to overcome;
- (b) Providing a design-first strategy to objects; and
- (c) Visualising objects in a meaningful context.

There are many visualisation systems, some focusing on algorithms, while others support the creation of operational programs. Some concentrate on low-level details, e.g. displaying data structures and their evolution during a program. Others incorporate a high-level of detail, showing component relations, program behaviours and methodologies. Some systems simply animate predefined data structures and programs. Conversely, other systems take in the learners’ own programs, affording them a chance to observe how they work and, finally, applying the necessary corrections (Gomes and Mendes, 2007).

Wright and Cockburn (2000) argue that within education-based environments, the ‘gulf of visualisation’ offers an opportunity to scaffold the learners’ grasp of the program. Systems can permit learners to manipulate the visualised behaviour of the environment in several ways, namely:

- (a) Change and control the time taken to execute instructions (e.g. step forwards and backwards through a series of statements); and
- (b) Reveal the internal structures that are usually hidden (e.g. the state of the runtime stack).

Wright and Cockburn indicate that a programming environment can support transfer effects between basic and more advanced programming concepts. This is achieved by providing controllable insights into the state of the machine, through appropriate metaphors.

The next section concludes Chapter 2 with an overall summary.

2.5 Summary and conclusion

Section 2.2 of this literature study chapter, presented a detailed discussion of the teaching and learning of computer programming by considering the theory of programming behaviour. The discussion led to an overview of the learning theories, namely the cognitive, constructivist and behaviourist approaches to teaching and learning. Furthermore, some of the problem-solving strategies relevant to providing support for learners in a programming domain, were explained. Finally, the importance of learner motivation in achieving learner success was discussed.

Section 2.3 focused particularly on learning within the domain of OOP, and commenced by defining this concept. There are several challenges that face novice learners on their initial exposure to OOP, and some of the most prominent challenges were identified and discussed. These included the lack of motivation for programming, difficulties in grasping language syntax, the need for immediate feedback on program execution and the difficulties in understanding compound logic.

Section 2.4 focused on innovative methods and techniques used to teach OOP. Algorithmic thinking and expression, abstraction, and the employment of an objects-first approach, have been identified as vital concepts that facilitate the teaching and learning of programming. Furthermore, it has been established that 3D animation authoring tools, as well as visualisation, can be effective methods of engaging and improving learners' experiences in learning OOP.

The *Alice* visual programming environment has been identified in the literature as a good rapid prototyping tool for 3D object behaviour. Chapter 3 elaborates on *Alice*.

Chapter 3: *Alice* Visual Programming Environment

3.1 Introduction

Visual programming environments, which are the focus of the literature study in this chapter, are tools used to improve the teaching and learning of programming. Such environments aim at:

- (a) increasing the motivation and interest of learners in programming;
- (b) promoting the understanding of program structures and execution status; and
- (c) enhancing efficiency of programming education (Nagaoka *et al.*, 2000).

Section 3.2 defines the concept of a visual programming environment (VPE), while Section 3.3 provides an evaluation of several VPEs that have been designed and implemented over the past years. The *Alice* visual programming environment is then discussed extensively in Section 3.4, highlighting how the tool addresses each of the challenges presented in Section 2.3.2 of this dissertation. Section 3.4 also explains why *Alice* was selected for this particular study, rather than any of the other ten systems mentioned in Section 3.3. Section 3.5 provides a summary to conclude the chapter.

The next section explains the concept of a visual programming environment.

3.2 What is a visual programming environment?

Glinert (1990) indicates that visual technology plays an important role within the programming processes of this type of environment. Combined with an integrated editor and interpreter, an appropriate programming language is required to develop such a programming environment. These environments may be developed to facilitate the functionality for viewing and/or manipulating either the dynamic or static variations of the program.

Giordano and Carlisle (2006) identified the following two categories of visualisation tools:

- (a) Tools that highlight traditional computer programming and expose the learner to some form of code or pseudocode; and
- (b) Tools that do not expose the learner to the code, due to it being hidden.

The *Alice* visual programming environment is the choice of software for this study, pioneered by the late, renowned, Professor Randy Pausch, together with his colleagues, Wanda P. Dann and Stephen Cooper, as well as the *Alice* development team. It falls within the second category

mentioned above, namely it does not expose the code, yet with the premise that learners will still be encouraged to construct statements that form methods and functions. Three-dimensional characters, animations and objects form the basis of *Alice*. Giordano and Carlisle (2006) further state that, although *Alice* objects demand a degree of syntactic understanding, learners are not compelled to type any text that demonstrates their logical thought processes.

The next section provides an evaluation of visual programming environments that have been developed over the years to improve the teaching and learning of programming.

3.3 Related works: Visual programming environments

Several software tools and visual programming environments have been developed to address the challenges that occur in teaching and learning introductory programming and aim, consequently, to improve the performance and learning experience of learners. Sections 3.3.1 to 3.3.10 introduce software tools that have a strong visual and graphical component. These tools were developed to help novice learners develop good intuition about OOP.

The environments discussed below, provide strong motivations for aspects such as the need for visual feedback, and the implementation of an ‘objects-first’ approach to programming. As previously mentioned in Section 2.4.3, the objects-first strategy emphasises the most fundamental principle of OOP and design. This strategy exposes the learner to the concept of objects and inheritance before introducing them to other traditional control structures.

The VPEs addressed in this section were designed taking into consideration the importance of learner motivation and engagement. Essentially, the environments aim towards helping learners to better understand the basic concepts of OOP. Furthermore, the VPEs were designed using graphical objects, thus contributing to avoidance of the syntactical errors that novice programmers often find frustrating and challenging.

3.3.1 *Logo*

Seymour Papert’s *Logo* is a popular microworld that uses turtle graphics to engage learners in exploring their creative line drawings. Devised in 1980, *Logo* was one of the early examples that applied a constructivist view to support learners in mathematical concepts and problem solving. The visual computer-controlled robot was called a ‘turtle’ and responded to program commands like ‘rotate’, ‘move forward’ and ‘draw’. The environment provides users with immediate feedback

about their programs, which facilitates the ease of identifying and correcting errors (Esteves *et al.*, 2008; Phelps *et al.*, 2005; Folk, 1981; and Alessi and Trollip, 2001).

3.3.2 *Karel the robot*

Karel the robot, originated by Rich Pattis in 1994, is a microworld that uses software simulation to assist learners with learning the processes of algorithms, such as assignment, selection and repetition. *Karel* also supports learning about objects. *Karel* exists in a 2D grid, within which methods may be invoked to manipulate the movements of the robots. These movements are rather limited, and are implemented by a small vocabulary, which allows for expansion. Learners are able to adapt easily to programming languages that resemble *Karel* (Cooper *et al.*, 2003; Larason, 1995). The world and the actions of the robots inside it are displayed visually on the computer screen, hence the software engages learners, whilst providing an interesting introduction to basic programming concepts. As a result of the visual representation of the robots, the animation provides learners with visual feedback on the correctness of algorithms. Merely by watching the animation, they are able to see where their programs go wrong. *Karel the robot* implements an objects-first approach, i.e. objects are instantiated, methods are invoked, and existing classes can be extended right from the beginning. The procedural concepts, such as selection, iteration and procedural decomposition are naturally blended with the object-oriented fundamentals (Becker, 2001).

3.3.3 *Kara*

In contrast to the features provided by *Karel the robot*, *Kara* is an educational software system, which allows learners to program the system in a purely graphical manner. A virtual ladybug is controlled by the learner through the use of finite state machines. The system was developed specially for novice programmers. It supports learning of the basic control structures such as command, sequence, conditional branch, and iteration. *Kara* exists on a grid-like chessboard in a world with fixed obstacles such as trees and movable objects such as cloverleaves. The ladybug is assigned typical tasks, for example, navigating through a forest of trees and collecting leaves. This requires a learner to identify the states needed and specify the transitions with the help of sensors and commands (Kiesmüller, 2009).

3.3.4 *BlueJ*

BlueJ is an Integrated Development Environment (IDE) designed using the Java programming language and intended for teaching introductory programming. The three goals considered during

design of this software tool are interactivity, simplicity and visualisation. Furthermore, it uses an objects-first approach. UML-like class diagrams are used to graphically represent the project structure. Instances of objects can be interactively created from any existing class in the software project, after which they become visible to the user. Learners can then interactively invoke any of its public methods by selecting them from a pop-up menu. Dialogue windows are used to enter and present parameters and method results. This environment aims to reduce the time learners take to familiarise themselves with the environment. Instead, it supports them in concentrating on the programming task at hand (Kölling and Rosenberg, 2001).

3.3.5 *Greenfoot*

Greenfoot is based on the aforementioned *BlueJ* platform. Java programs are created by combining the Java IDE with a framework, after which the learner can visualise the program using a two-dimensional grid. The framework uses a world environment as a basis and allows learners to visualise how objects contained within the world interact with each other and the world itself. (Montero *et al.*, 2010).

3.3.6 *Second Life*

Second Life (SL), originated by Philip Rosedale in 1991, is an online 3D social virtual world used to teach introductory programming courses to novice learners. The functionality of *SL* allows for multiple user connections in a virtual environment that supports interaction and collaboration. *SL* prompts users to create objects via avatars and then code their behaviours. A C-style script language called Linden Scripting Language (LSL) is used for *SL*. Learners engage with other learners as the system allows multiple avatars to simultaneously share the same code and edit the same object. *SL* intends to reduce the complexity of the development environment and to help users avoid compiler errors. It also aims at providing immediate perceptual feedback (Esteves *et al.*, 2008).

3.3.7 *Muppets*

Muppets (Multi-User Programming Pedagogy for Enhancing Traditional Study) is a Collaborative Virtual Environment (CVE) that teaches introductory-level programming through the creation of avatars and objects. The system allows learners to engage themselves in a complex, interactive world where advanced code generation methods form a comprehensive, integrated development environment. *Muppets* deploys an objects-first methodology, so learners are immediately exposed to polymorphism, encapsulation and inheritance. These three concepts are central to the system.

Muppets also provides support for class relationships, including composition and association. This environment also visually differentiates between classes and objects. Immediate feedback allows learners to generate multiple instances of classes and to interrogate each interactively and independently. The *Muppets* system fosters communication between learners by using a shared virtual world to share objects across various course sections and different programming levels. The system is therefore able to bridge the gap between senior learners and novice learners, in that an advanced object created by a senior learner can inspire and serve as a source of ideas for first-level learners (Phelps *et al.*, 2005).

3.3.8 *Scratch*

Designed for the educational sector, *Scratch* is a programming language and environment intended to improve the development of technological fluency at after-school centres in communities that are economically-disadvantaged (Maloney, Burd, Kafai, Rusk, Silverman and Resnick, 2004). Potential users of the software includes learners with varying levels of experience, age and background and allows them to explore the concepts of computer programming (Sandoval-Reyes, Galicia-Galicia and Gutierrez-Sanchez, 2011). The Lifelong Kindergarten Group, headed by Mitchel Resnick and his team, developed *Scratch* in 2007 at the Massachusetts Institute of Technology Media Laboratory (Sandoval-Reyes *et al.*, 2011, Kaučič and Asič, 2011).

Scratch is “targeted for interactive applications such as stories, animations, games, music, and art” (Kato, 2010:4). This is achieved by importing all the objects, graphics, sounds and scripts to a new program, thus allowing novice programmers to get immediate feedback and thereby improving motivation (Sandoval-Reyes *et al.*, 2011). *Scratch* has been designed considering the following core features:

- (a) building block programming; programmable manipulation of rich media;
- (b) deep shareability;
- (c) integration with the physical world; and
- (d) support for multiple languages (Maloney *et al.*, 2004).

Scratch is comprised of two key components, namely a community-based web interface and a visual programming language (Sandoval-Reyes *et al.*, 2011). “*Scratch* adds programmability to the media-rich and network-based activities that are most popular among youth” (Maloney *et al.*, 2004:104). For instance, version *Scratch* 2.0 was built to support Adobe Flash Player 10, thus

enabling *Scratch* to run on handheld devices such as Samsung and Nokia, with the exception of Apple (Sandoval-Reyes *et al.*, 2011).

3.3.9 Etoys

Etoys is a VPE designed with the needs of novice programmers in mind, particularly teachers and young children. A new program begins with the creation of a ‘Sketch’, which is a 2D graphical object. This is achieved in one of two ways, namely by importing an external image file or by using a built-in paint tool. The behaviour of a Sketch (such as sound, movement or appearance) can be controlled after its creation, by attaching an object called a ‘Script’ to it. Furthermore, *Etoys* supports the development of multimedia products such as games, digital storybooks and animations. Novice programmers are able to incorporate sound effects relatively easily to computer animation programs. *Etoys* supports “many built-in widgets, such as ‘Book’ and ‘Playfield’ that can be readily used in creating educational materials” (Lee, 2011:528).

3.3.10 Lego Mindstorms

Klassner (2002) states that the *Lego Inc. Mindstorms* robot development and programming kit, which was designed for middle-school learners, was released in 1996. Talking robots can be constructed from off-the-shelf components using *Lego Mindstorms’* robotics platform (Koller and Kruijff, 2004). They advocated *Lego* robots due to the affordability and potential for any dialogue researcher confronting challenges at the robot-dialogue interface. McNally, Goldweber, Fagin and Klassner (2006:61) describe *LegoMindstorms* as “an inexpensive robotics system consisting of a microprocessor brick, various sensors and motors, and numerous *Lego* pieces. It can be programmed in a variety of languages, including Java and C++”. Qidwai (2007) observed the system being implemented in numerous primary schools and universities.

Researchers believe that *Lego* robots has earned its popularity as a good teaching tool. This can be attributable to the following reasons:

- (a) *Lego* robots provide a fun learning experience;
- (b) Robot kits provide flexibility in relation to constructing and programming;
- (c) *Lego* robots are affordable; and
- (d) *Lego* robots can be combined with other similar components and systems.

The software boasts that learners require no prior knowledge of conventional programming languages when learning how to code the robot. Moreover, learners are able to logically connect drag-and-drop blocks located in the graphical libraries, thus alleviating them from writing lines of code.

Lui, Ng, Cheung and Gurung (2010) developed a course that uses *Lego NXT Mindstorms* robot kits as a tool to engage learners at various levels of learning independence and to promote self-directed learning skills. Agarwal, Harrington and Gusman (2012) posit that while the advancement of Android technology remains rapid, the built-in features of Android can be harnessed into robotics programming. An application was written to showcase the integration of an Android component and a *Lego Mindstorm NXT* robot. This was achieved by mounting an Android device onto the robot, whilst using the device's camera to capture live video. The captured footage was streamed to another Android device, which served as a controller.

The next section provides a detailed discussion on the *Alice* visual programming environment.

3.4 Alice 2.2 Environment

This study aims to investigate the effectiveness of using the *Alice* visual learning and programming environment as a tool to aid the teaching of object-oriented programming to novice learners. Section 3.4.1 provides a description of the software tool.

3.4.1 What is Alice?

Alice is an open source teaching tool, designed to provide learners with first-time exposure to the basics of OOP. Learners are encouraged to learn core programming concepts whilst creating 3D animated movies and basic video games, thus providing a fun, interactive environment (Carnegie Mellon University, 2012). *Alice* presents users with an easy to use front-end interface that facilitates the creation of multi-dimensional, readily populated scenes. These scenes include objects such as buildings, people, animals, and machinery. Users of *Alice* then animate the objects by creating written executable functions and methods (Zaccone, Cooper and Dann, 2003).

The keyword *Alice* is often considered to be an acronym, but this is not so. The explanation of how the name for this system was derived is as follows. The name '*Alice*' is in honour "of Charles Lutwidge Dodson, an English mathematician and logician who wrote under the pen name Lewis Carroll" (Dann *et al.*, 2009:34). He created the famous fictional character, Alice, the little girl who

ran down a rabbit hole and experienced amazing underground adventures. Carroll wrote *Alice's Adventures in Wonderland* and *Through the Looking Glass*. He appreciated the importance of making things simple and fascinating to a learner and was an inspiration to the creators of the *Alice* visual environment, who similarly aimed to simplify complexity.

Section 3.4.2 addresses the relevance of the *Alice* visual environment to the current study.

3.4.2 The relevance of *Alice* visual environment to the current study

The characteristics of *Alice* as a tool of learning embodies the methods discussed in Section 2.4 and attempt to address the challenges identified in Section 2.3.2. These characteristics include the following:

- (a) the concrete visualisation of concepts such as objects and basic inheritance;
- (b) motivation of learners by providing interesting problems for them to solve;
- (c) release from having to deal with complex syntax mechanics, while errors in logic become visually obvious; and
- (d) simplification of event-driven programming, which is interesting to explore in the *Alice* VPE (Johnsgard and McDonald, 2008).

The goal of *Alice's* innovative approach in teaching programming is also to aid educators in the instructional process and to allow for easier assimilation by learners of traditional program-creating concepts. The authors of *Alice* consider the system to represent a breakthrough in teaching object-oriented computing. Objects in *Alice* are reified as 3D humans, furniture and animals, thereby making them easily visible, concrete and real. Furthermore, the state of *Alice* objects can be changed by calling methods such as 'move forward one meter' or 'turn left a quarter turn'. Such object behaviours are intuitively and easily understood by learners. "One of *Alice's* real strengths is that it has been able to make abstract concepts concrete in the eyes of first-time programmers" (Dann *et al.*, 2009:11).

Alice has an interactive interface, in which learners use drag-and-drop graphic tiles to formulate coding statements during program creation. Learners are able to relate these instructions to standard statements in commonly used programming languages, such as C#, Java and C++. The VPE allows learners to instantly visualise the execution of their animation programs. Learners are thus enable to easily understand the relationship between the programming statements and the behaviour of the animated objects. Moreover, they are encouraged to manipulate the objects in their virtual world, whilst gaining experience in programming concepts such as loops, if statements, properties,

methods, functions, events etc. *Alice* thus exposes the learner to the basic programming constructs typically addressed in introductory programming courses (Carnegie Mellon University, 2012).

Section 3.4.3 provides an overview of *Alice*.

3.4.3 An overview of *Alice*

Section 3.4.3.1 provides the background information on *Alice* and Section 3.4.3.2 uses a sample scene and sample code to expound the explanation of the interface.

3.4.3.1 Background to *Alice*

As mentioned earlier, one of the core benefits afforded to learners using *Alice* is the ability to immediately see the effects of animation changes. Programs are coded in an object-oriented, interpreted language. During program execution, the current state of a program can be updated by the learner in one of two ways, namely, they can evaluate the fragments of program code or they can manipulate tools that are available in the *Alice* GUI (Graphical User Interface). Although *Alice* is extremely flexible at runtime, the system maintains highly interactive frame rates. This is done transparently through the use of *decoupling simulation* and *rendering* (Pausch, Burnette, Capehart, Conway, Cosgrove, Deline, Durbin, Gossweiler, Koga and White, 1995).

Alice is written using *Python*, which is an object-oriented, interpreted, high-level language. Its support of high-level data types such as lists and hash tables as primitives, was considered when choosing the language. Moreover, Python supports multiple inheritance, polymorphism and contains a widespread collection of run-time libraries. These include a large extent of the UNIX libraries usually accessed via the C language (Pausch *et al.*, 1995).

Versions of *Alice* have been made available for multiple operating systems. It was first released for Microsoft Windows and subsequently for Apple Mac and Linux.

3.4.3.2 Sample scene and code

Alice is a tool that learners use to animate 3-dimensional objects using programming statements. The *Alice* integrated development environment provides ease of use. Learners are able to use drag-and-drop tiles to create instructions that animate objects, thus providing them with simplified

syntax. An impression of the interface is represented in Figure 3.1 with an animation ‘Defending Naptime’, that tells a short story about a bunny whose sleep is interrupted by a cell phone. A discussion of the areas that comprise the *Alice* interface follows, highlighting details of how the animation is developed.

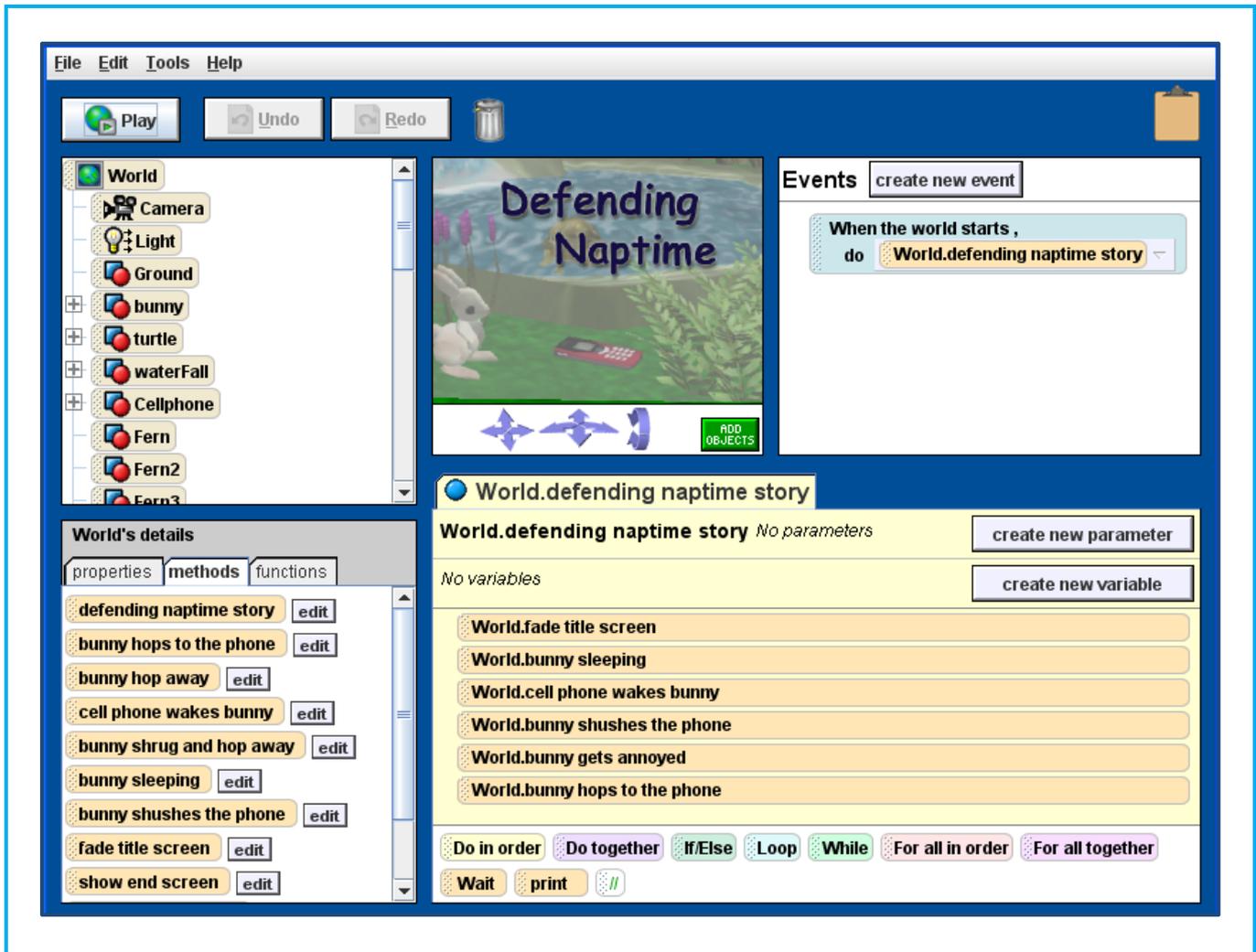


Figure 3.1 The *Alice* interface representing a sample animation

As depicted in Figure 3.2, the five distinct areas that encompass the main *Alice* interface are:

- Upper centre [World preview]: A picture of the current world;
- Upper left [Object tree]: A list of all objects that are in the world;
- Upper right [Events editor]: A list of world events;
- Lower left [Details pane]: The object’s properties, methods and functions; and
- Lower right [Method editor]: A method is a sequence of instructions that represent the animated behaviour of an object and is created using the drag-and-drop feature.

These five areas pertinent to the *Alice* interface will be further expounded.

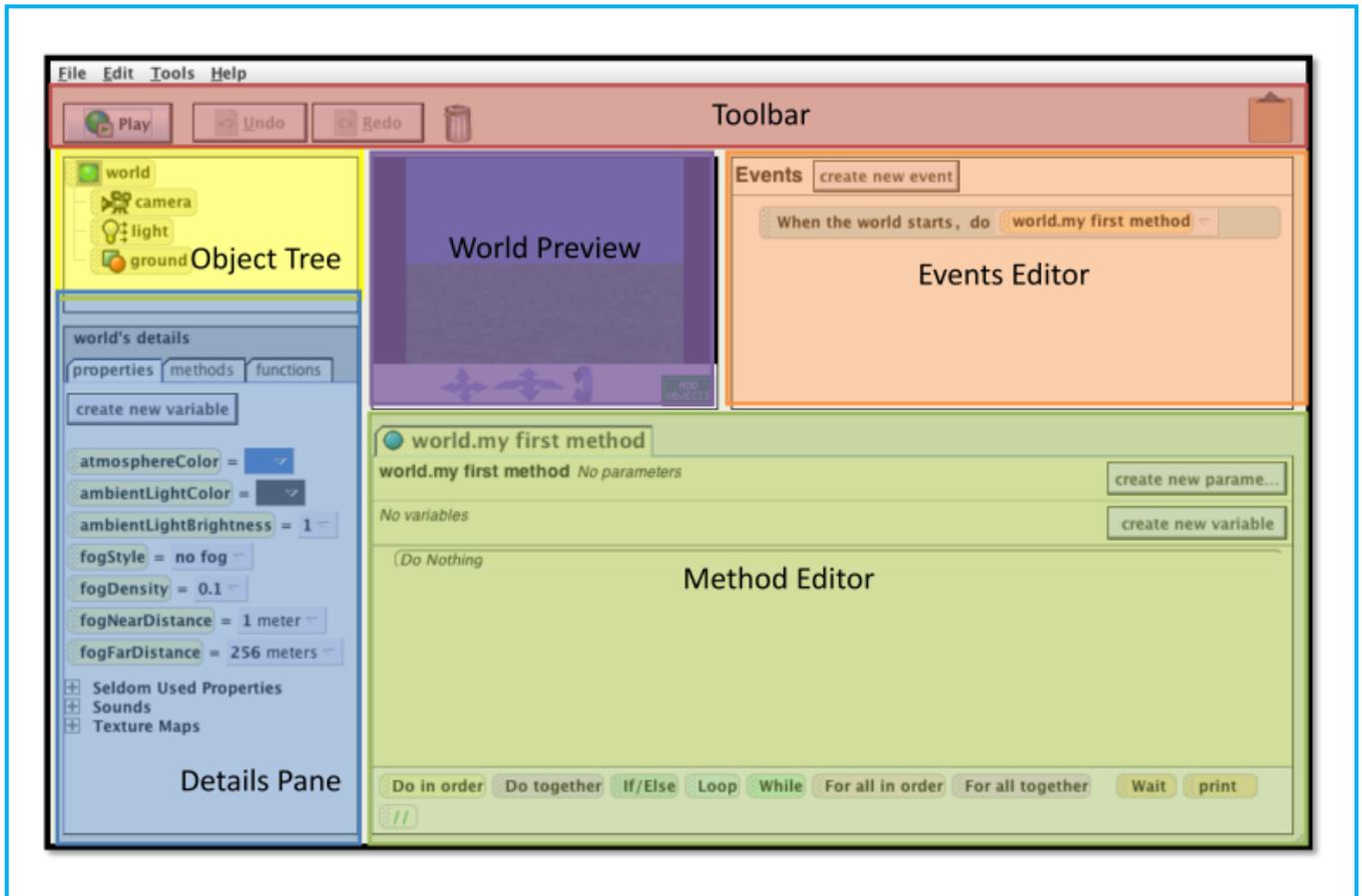


Figure 3.2 The five areas of the *Alice* interface

World preview

When developing an animation, a learner can add 3D objects from the gallery to a small virtual world. The gallery contains different types of objects that the learner can add to a world in *Alice*. It is organised into folders, such as ‘amusement park’, ‘animals’ and ‘beach’, which are listed in alphabetical order. The learner can then arrange the position of each object in the world as he/she pleases.

Object tree

All existing objects in the current world appear in the object tree and serves to reify the commonly used concept of an ‘object’ in traditional languages. Objects can be collapsed so as to reveal its subparts. For instance, the ‘bunny’ object representing the main character in the ‘Defending naptime’ animation depicted in Figure 3.1 contains the subparts ‘rightLeg’, ‘leftLeg’, ‘upperBody’ and ‘tail’. These subparts can be seen in Figure 3.3. Some of these subparts can be further subdivided into smaller parts, for instance, the ‘upperBody’ contains a ‘rightArm’, ‘leftArm’ and

‘head’. With respect to the level of depth of subparts contained within each *Alice* object, learners have greater flexibility and scope to animate the movement of objects with precision and detail.

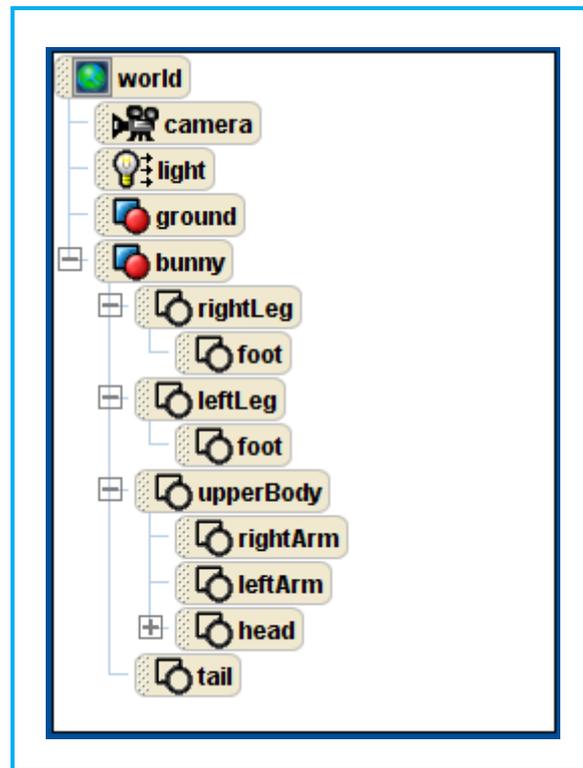


Figure 3.3 The object tree with subparts for the ‘bunny’ object

Events editor

Similar to the concept of an ‘event’ in conventional languages, the methods defined by a learner in *Alice* are dependant on an event trigger that occurs during program execution. The default event ‘When the world starts’ can be used to execute programmer-defined methods. Furthermore, new events can be created by the learner to trigger methods, such as with a keypress or mouse click.

Details pane

Through methods and functions, *Alice* objects can be rotated 360 degrees in multiple planes to provide a visual perception of width, height and depth, similar to that of a real-world object. This functionality lends virtual reality to the object (Zaccone *et al.*, 2003). The internal data structure of each *Alice* object, which contains its specific private properties such as width, height and location, is encapsulated. In addition, these objects have their own member methods. The properties, methods and functions of objects are available to the *Alice* user in the details pane.

Method editor

Once the virtual world is initialised and objects are added to the scene, programs are constructed to control the behaviour and appearance of these objects, in response to input from the keyboard and mouse. The program code is written using the method editor in *Alice*. With the simple click of a mouse, an object is dragged into the method editor and learners are allowed to select a primitive method from a drop-down menu that sends a message to the object. For example, with reference to the sample animation illustrated in Figure 3.1, calls are made to the following methods that belong to the ‘World’ object:

- (a) ‘World.fade title scene’;
- (b) ‘World.bunny sleeping’;
- (c) ‘World.cell phone wakes bunny’;
- (d) ‘World.bunny shushes the phone’;
- (e) ‘World.bunny gets annoyed’; and
- (f) ‘World.bunny hops to the phone’.

The animation makes use of the terms ‘shush’ and ‘nap’. These terms are pre-coded into the methods that are called to set the objects in motion. In addition, learners are able to write their own user-defined functions and methods, and these are automatically added to the drop-down menus for ease of use. For instance, user-defined methods can be coded to ‘smash the phone’ and ‘hop away’.

For ease of use and understanding of elementary programming structures, such as ‘If-Else’, ‘Loop’ and ‘While’, these constructs reside within the method editor in the form of drag-and-drop tiles. These tiles replace the equivalent complex syntax constructs used in languages, such as C++, Java and C#. By using the method editor, learners are spared from the tedious dealing with symbols such as brackets, semi-colons etc. Furthermore, they do not need to concern themselves with syntax errors.

In addition, the method editor helps the learner to position objects and statements in locations that are syntactically correct. For instance, bearing in mind that a loop condition can be either one of two states i.e. true or false, *Alice* allows for a construct of data type ‘Boolean’ to be dropped only into the condition placeholder of a ‘While’ loop. *Alice*’s method editor thereby allows the novice programmer to learn about syntax, while reducing the frustration of learning how to type syntax code (Zacccone *et al.*, 2003).

An incremental approach is used in the implementation and testing of written programmes. This incremental approach gives the learner the opportunity to test specific methods and commands to determine whether they perform as expected. The learners are thus able to visualise the relationship of the program construct to the animated action resulting from the execution of the method (Zaccone *et al.*, 2003). For example, Figure 3.4 depicts an initial scene of the ‘Defending naptime’ animation previously illustrated in Figure 3.1. This screenshot was taken during program execution, and gives the learner a visual representation of the status of their program code. Learners are able to pause, play, restart and stop the animation, toggle its speed, as well as take a picture at any given time.



Figure 3.4 An initial scene in an *Alice* world during program execution

A discussion is provided in Section 3.4.4 on how *Alice* is used to teach programming. Furthermore, each of the challenges identified in Section 2.3.2 are discussed in more detail in relation to how *Alice* attempts to address these challenges.

3.4.4 The approach used by *Alice* to teach programming

The approach used by *Alice* to teach programming, allows learners to creatively and enjoyably storyboard, direct and edit on-screen movies and games. In doing so, the concept of an ‘object’ is made tangible and visible. A 3-dimensional virtual world is populated with a collection of *Alice* objects. *Alice*’s drag-and-drop feature facilitates the creation of programs by allowing learners to combine program elements such as if-else statements, loops, variables etc. into a method editor. These program constructs come pre-packaged with the software, and are intended to reduce syntax errors.

The *Alice* visual programming environment is a modern tool that support variables, methods, functions, parameters, recursion, events and arrays. Such an environment is used to support an objects-first approach by introducing the learner to events at an early stage in a programming course (Joint Task Force on Computing Curricula, IEEE, 2001). Dann *et al.* (2009) posits that in *Alice*, learners can visibly see every object. Hence, the system promotes an objects-first approach to programming.

The generation of code is dependant on visual formatting and releases learners from concerning themselves with punctuation details, such as commas, parentheses and semicolons. In *Alice*, the focus of the learner shifts from syntax to concepts such as encapsulation and objects. Although the coding is visual, learners are encouraged to maintain good program structure, whilst enforcing the semantics of code statements (Cooper *et al.*, 2003).

The next sub-sections highlight how *Alice* addresses the challenges that learners are faced with in object-oriented programming, as discussed in Section 2.3. The developers of *Alice* identified four obstacles to learning introductory programming, namely:

- (a) lack of motivation for programming;
- (b) complex syntax and semantics;
- (c) the need for immediate feedback and identifying the results of computation as the created program runs; and
- (d) difficulties in understanding compound logic and the application of algorithmic problem-solving skills (Cooper *et al.*, 2003; Dann, Cooper and Pausch, 2012).

The *Alice* development team has proposed ways to address each of these four challenges, as will be discussed in Sections 3.4.4.1 to 3.4.4.4.

3.4.4.1 How does *Alice* address the lack of motivation for programming?

While external motivation can be created using rewards and punishments, such as the awarding of marks for a learners programming assignment, the *Alice* system uses a purer form of motivation, namely, intrinsic motivation. It bases programming in the activity of storytelling, which is appealing to many learners. Rather than writing programs that perform calculations and display text, learners use *Alice* to create rich animations and computer games. By using 3D graphics as the authoring medium, the *Alice* system speaks directly to a generation raised on video games and Pixar films. Storyboarding is considered to be one of the few design activities that can be immediately understood by a university learner (Dann *et al.*, 2009; Gaddis, 2011). Today's generation of learners are media-conscious and susceptible to the ease with which 3D graphical environments can be used (Moskal *et al.*, 2004).

3.4.4.2 How does *Alice* address complex syntax and semantics?

The *Alice* system provides a well-engineered drag-and-drop user interface that allows learners to move program components around the screen and ensures that they cannot make syntax errors. The editing environment is thus ideally suited for supporting novice programmers in developing an intuition for syntax (Dann *et al.*, 2009). The method editor prevents learners from making the kinds of syntax errors that are common amongst beginner programmers (Moskal *et al.*, 2004). Runtime errors can still occur when learners use incorrect instructions or place instructions out of sequence. However, because syntax is not an issue, learners can devote their time to developing and debugging algorithms (Gaddis, 2011).

3.4.4.3 How does *Alice* address the need for immediate feedback and identifying the results of computation as the created program runs?

Conventional programming environments provide for some degree of feedback, in the form of variable watchers and textual debuggers. Conversely, the *Alice* VPE makes the ongoing state of the program visually available, which relieves the cognitive load placed on learners in favour of providing input to their perceptual systems. It appears that learners find it easier to see backward movement of an object rather than forward movement of an object and to take notice of the fact that a 'sum' variable has been decremented, rather than incremented. *Alice* provides a platform for learners to watch how their animated methods and functions execute (Dann *et al.*, 2009). The visual nature of *Alice* and the immediate feedback produced during program execution, allow learners to

easily see the effect of a programming statement or group of statements. Thus, the debugging process is simplified (Moskal *et al.*, 2004).

3.4.4.4 How does *Alice* address the difficulties in understanding compound logic and the application of algorithmic problem-solving skills?

The creation of small functions and methods is promoted in the *Alice* environment. Movie and game generation are based on the concept of storyboards. This is integrated into the *Alice* environment and leads learners into a well-known and established movie-making procedure. The user-friendly screen captures and simplified sketches, aid and illustrate design techniques. The creation of textual storyboards evolving by refinement and the use of design with pseudocode, are encouraged (Dann *et al.*, 2009). The *Alice* gallery contains a range of 3D modelled classes, from which objects can be instantiated at the discretion of the learner. *Alice* objects represent real-life objects such as cars, trees and horses, thereby providing a concrete view of the concept of an object. This promotes an objects-first approach to learning programming (Moskal *et al.*, 2004).

Section 3.4.5 addresses the shortcomings of *Alice*.

3.4.5 Shortcomings of *Alice*

According to Baldwin (2007), *Alice* does not directly support inheritance. There is also no support for polymorphism in *Alice*. Johnsgard and McDonald (2008) noted that the creators of *Alice* assigned an arbitrary centre point for each 3-dimensional class in version 2.0. This prevents the user from changing the centre point and its associated orientation, resulting in some movement methods being inconsistent and further leading to a lack of true inheritance and polymorphism.

Due to these issues, the walking motions of every ‘animal’ or ‘human’ object must be programmed one by one. This could prove to be an arduous programming task for novices who desire realistic movements of objects. This shortcoming detracts from the essence of object-oriented programming concepts and may tend to inhibit learner creativity.

Other operational problems are known to occur in *Alice* and can be frustrating to work with at times. There are instances when the application freezes during garbage collection, and instances of the application crashing completely due to insufficient system memory. Such occurrences can cause the learner to lose all current work in progress.

Wright and Cockburn (2000) reveal that there is an inadequacy in *Alice*, in that the environment does not support users in watching the programs run in a stepwise manner. The entire program or a procedure call executes in a single step. This does not allow the programmer to watch and comprehend the relationship between each program instruction and the state of the system.

Johnsgard and McDonald (2008) express concern about gender-bias, which can occur when learners are given autonomy to choose the content of their *Alice* movies. Male learners are inclined to choose scenes that include explosions, violence or vehicles, whereas female learners are inclined to depict animals, humans and fairytale characters. Females are favourable to non-violent scenes that contain basic movement of objects, such as a flying pixie.

Research conducted by Johnsgard and McDonald (2008), revealed that the emphasis that *Alice* places on story-telling may have created difficulties for 'at-risk' learners when they must transition from using *Alice* to programming in more general-purpose OOP languages. A study by Cliburn (2008) confirmed that learners found it challenging to transition from *Alice* to traditional languages such as Java and C++. Furthermore, certain features prominent in *Alice*, such as the object model, can lead learners to develop misconceptions when they learn a subsequent language.

Moreover, the findings of Cliburn indicate that when learners are taught *Alice* in conjunction with another language within the same course, then it is possible that the goal of the course can become distorted to learners. An alternative approach would entail introducing *Alice* as the only language in a particular course, to ensure that learners devote their attention to the programming concepts enforced in *Alice*. When learners encounter a subsequent language, they can avoid confusion.

Section 3.4.6 highlights case studies on previous adaptations of the *Alice* visual tool in programming courses.

3.4.6 Previous case studies: The use of *Alice* in programming courses

Alice has been successfully used in programming courses at several educational institutions. Zaccone *et al.* (2003) reported that *Alice* was used as an effective tool to demonstrate fundamental programming concepts at an Engineering course seminar. The beneficial features of *Alice*, namely the visualisation of programs and a method editor, facilitated the introduction to basic OOP concepts, such as inheritance. This was achieved in only three weeks. Learners expressed that *Alice*

was an engaging tool for learning to program and they displayed an increased interest in Computer Science.

Studies conducted by Kelleher and Pausch (2006) aimed at presenting computer programming using storytelling to help motivate girls to learn to program. The girls that participated in the study were able to create their individual *Alice* programs based on a storyline. Many learners were inspired by the needs of their storylines and sought to use more complicated programming constructs, such as loops and parallelism. Once satisfied with the layout of the scenes, most girls were able to code and call multiple methods to organise animations that comprised multiple scenes.

A study conducted by Edwards *et al.* (2007) reflected on how the *Alice* VPE was successfully used to teach first-level learners the core concepts of OOP. Several opportunities and challenges presented themselves by including a cultural perspective into the teaching of an introductory *Alice* course. The incorporation of Hawai'ian mythology into the course, with its myths and legends, blended well with the narrative, story-based approach offered by *Alice*. Learner survey data has revealed that using this approach for instruction was efficient.

Not all findings are positive. For instance, a study conducted by Cliburn (2008) discussed the findings extracted from 84 learners who participated in an introductory programming course (Computer Science 1). The course incorporated *Alice* and Java concurrently in the same term. A fair percentage (59.5%) of the learners believed that prior exposure to *Alice* assisted them to grasp Java later in the term. Sixty-seven percent (66.7%) of learners were in favour of continued use of *Alice* in the course, whilst others would prefer a reduction of the time spent on *Alice*. Although these statistics would ordinarily be interpreted as a positive result, educators felt that a withdrawal of the *Alice* intervention from CS1 was warranted. The degree of uncertainty expressed by learners as to whether *Alice* helped improve an understanding of Java later in the term, was considered adequate to justify a discontinuation of *Alice* coverage in the course.

Wang *et al.* (2009) conducted a quasi-experiment to gauge the success or failure of implementing *Alice* as a programming intermediary in secondary education. *Alice* was used to teach two randomly selected high school classes. Simultaneously, a comparison group of two classes was being taught C++. This intervention was undertaken over a period of eight weeks. Learners were familiarised with programming constructs including selection structures, variables, arithmetic expressions, built-in functions and repetition structures. A subsequent analysis of learners' test results revealed a significantly better performance of the *Alice* group when compared to the C++ group. *Alice* was

thereby regarded as a more effective tool for facilitating learners' understanding of fundamental programming concepts. The findings of a questionnaire indicated that no significant difference existed between the two groups, with regards to improving learners' motivation in programming or their holistic learning experience. It was therefore implied that both languages were received equally well by the learners.

Salim *et al.* (2010) conducted an experiment to measure the effectiveness of *Alice* by using it to teach programming concepts to first-level learners. The study was conducted in 2008 and 2009 with learners from the Faculty of Computers and Information at Cairo University. The study was implemented prior to introducing them to a high-level language. A survey was used to measure the level of acceptance and benefit attained by the target learners. A notable result of this study was that learning with *Alice* was a beneficial experience for learners who were having a first-time exposure to computer programming. However, the learners who had a previous programming proficiency found *Alice* to be rather boring.

A study was conducted by Webb and Rosson (2011) with the target population being middle school girls attending a week-long summer camp. The central theme emphasised future career choices that rely on computer technology. The programme followed by the girls included:

- (a) visits to campus computer labs;
- (b) talks on computer technology addressed by guest speakers; and
- (c) sessions that encouraged the creation of original *Alice* projects.

The results revealed that while the *Alice* projects may not have been equally engaging for all girls, those who did find them appealing were more likely to gain a positive disposition towards a career in the field of computing.

A similar study by Mason, Cooper and Comber (2011) used high school promotional visits and campus workshops to encourage female junior high learners to consider a career in IT. The girls were divided into two equal groups with the first group doing a session on programming with *Alice* while the second group had an introduction to *Mindstorms NXT robots*. After a period of time, the two groups swapped activities. Prior to the commencement of the workshops, the girls anticipated that *Alice's* 3D storytelling approach would be easier to use than *Mindstorms's* mechanical approach. However, after experiencing each of the two environments, they found them both to have a moderately low level of difficulty. Moreover, the girls were comparatively more positive to *Mindstorms* after the workshops. Ultimately, the girls' perception of the difficulty of programming altered because of the workshops.

Six institutions involved in higher education, in collaboration with secondary education schools attached to these tertiary educational bodies, embarked on a four-year project. Cooper, Dann, Lewis, Lawhead, Rodger, Schep and Stalvey (2011) described the findings from the project. At its core, the project sought to provide middle and high school educators with professional development. The concurrent objective was to equip these educators with the competencies needed for them to create learning programmes that would enthuse their learners about computing. To realise these intentions, the project employed *Alice* as the tool to elicit a high degree of interest in animation, computer graphics, and storyboarding and thereby to help the learners grasp the concepts of object-based programming. Out of the 100 or more secondary school educators who participated in this project, approximately 80% reported that the learning style that they were exposed to, aided them in their own classrooms and subsequent teaching.

Jones, Kisthardt and Cooper (2011) attempted to develop a new approach for teaching introductory programming via creative writing. The authors state that creative writing begins by connecting the planning, organising and detailing of writing a story to the programming process. Creative concepts were taught first and then connected to an equivalent programming concept. Learners were required to work in pairs. This was achieved by permitting a learner with strong analytical abilities to pair with a learner studying in the field of humanities. The pairs applied the concepts and skills by creating stories, designing an animation, and implementing the animation using the *Alice* programming system.

Section 3.4.7 explains why the researcher chose *Alice* as a VPE for this study.

3.4.7 Rationale for the choice of *Alice* as a VPE for this research

Several factors were pertinent to the choice of *Alice* as the VPE for this research, rather than one of the other ten systems described in Sections 3.3.1 to 3.3.10:

- As mentioned in Section 3.4.1, *Alice* is an open source teaching tool, which made the software easily available at no cost. Most of the systems described in Section 3.3 have to be purchased and this requires capital outlay.
- Unlike similar systems which have remained static and that have not been upgraded since their introduction, the *Alice* VPE has evolved, and continues to evolve, over time. Several versions have been released and the latest version, *Alice 3*, contains explicit support for transitioning to

Java, which is a programming language taught within the ND: IT programme. The dynamic nature of this VPE is appealing as it takes into account constant developments in the IT sector.

- The case studies outlined in Section 3.4.6, revealed that *Alice* had been successfully used to teach programming to learners in secondary education classes and to provide similar instruction to first-year university students. This study followed on these successes achieved in international work by investigating the use of *Alice* to conduct workshops with learners registered for a second-year subject who already had exposure to basic programming.

The next section concludes Chapter 3 with an overall summary.

3.5 Summary and conclusion

Section 3.2 of this literature study chapter, defined the concept of a visual programming environment.

In Section 3.3, the functionality of various visual environments was demonstrated, prior to introducing the *Alice* visual programming tool. These visual learning and programming environments have been designed and implemented across various institutions in the world. Such systems have achieved success in improving the learning experience of novice programming learners.

Section 3.4 introduced and defined the *Alice* visual programming environment, following which the relevance of *Alice* to the current study was addressed. An overview of *Alice* was provided, beginning with a discussion of the background of the *Alice* software tool. Further, a sample scene and sample code were used to expound the details of the interface. The approach used by *Alice* to teach programming was considered, by elaborating on each of the challenges identified in Section 2.3.2 and explaining how *Alice* attempted to address these challenges. Finally, the shortcomings of *Alice* were addressed, following which the relevant findings were highlighted from previous cases where the tool was adopted in programming courses.

The next chapter provides an extensive discussion of the research design and methodology for this study, which was briefly introduced in Chapter 1.

Chapter 4: Research Design and Methodology

4.1 Introduction

This study investigates the effectiveness of implementing the *Alice* VPE in a second-level programming course at DUT, in terms of improving the performance of the learners and their learning experience. Chapter 4 explains the research design and the scientific methods used to collect data to answer the research sub-questions listed in Section 1.6, and repeated in Section 4.2.

Section 4.2 maps the research questions to the locations in the dissertation where they are addressed. The research design and methodology for this study are extensively discussed in Section 4.3. The targeted population and sample are considered in Section 4.4, while Section 4.5 presents a detailed explanation of the design of the data collection instruments administered to the selected samples of learners. A pilot study was conducted prior to Case Study 1, and is addressed in Section 4.6. Section 4.7 discusses reliability and validity issues. The tools and methodologies employed in the processes of quantitative and qualitative data analysis are explained in Section 4.8. Section 4.9 describes the ethical considerations of this study and the chapter is concluded with a summative overview in Section 4.10.

The section that follows indicates sections in the dissertation where the research questions are addressed.

4.2 Research questions and where they are addressed

As mentioned in Section 1.6.1, the primary research question enquires:

To what extent can the implementation of the Alice visual programming environment in a second-level programming course at the Durban University of Technology improve the performance and learning experience of learners?

The primary research question for this study gives rise to the sub-questions presented in Table 4.1, alongside the locations in the dissertation where they are addressed.

Table 4.1 Research sub-questions with corresponding locations in dissertation

Number	Sub-question	Location in dissertation	
1	What is the effectiveness, as perceived by learners, of using the <i>Alice</i> visual programming environment in addressing the challenges facing novice programming learners within the object-oriented domain?	Chapter 5	Section 5.4.2.2 to 5.4.2.5 Section 5.5.2.4 to 5.5.2.7
		Chapter 6	Section 6.4.2.2 to 6.4.2.5 Section 6.5.2.4 to 6.5.2.7 Section 6.6.2.4 to 6.6.2.8 Section 6.8.1 to 6.8.2
		Chapter 7	Section 7.4.1
2	How do learners experience the usability of <i>Alice</i> ?	Chapter 5	Section 5.4.2.1 Section 5.5.2.1 to 5.5.2.3
		Chapter 6	Section 6.4.2.1 Section 6.5.2.1 to 6.5.2.3 Section 6.6.2.1 to 6.6.2.3 Section 6.8.1 to 6.8.2
		Chapter 7	Section 7.4.2
3	To what extent do the test and exam results of participating learners relate to those of similar learners who were not exposed to the <i>Alice</i> intervention?	Chapter 5	Section 5.6.2.1 to 5.6.2.2
		Chapter 6	Section 6.7.2.1 to 6.7.2.2
		Chapter 7	Section 7.4.3

Figure 4.1 provides a visual representation of routes traversed in the formulation of this dissertation. It is presented in the form of a road map that binds the entire study together, and shows the flow of the research process from the inception of the primary question to finding its solution. The centrepiece of the diagram illustrates the primary research question, which is comprised of the three sub-questions shown in Table 4.1. The arrows emanating from Sub-question 1 and Sub-question 2 in Figure 4.1 merge and lead into both Case Study 1 and Case Study 2, the results of which contribute to the final answer. Similarly, Sub-question 3 is addressed in both case studies, contributing further to the final answer to the primary research question, which will be addressed in Chapter 7.

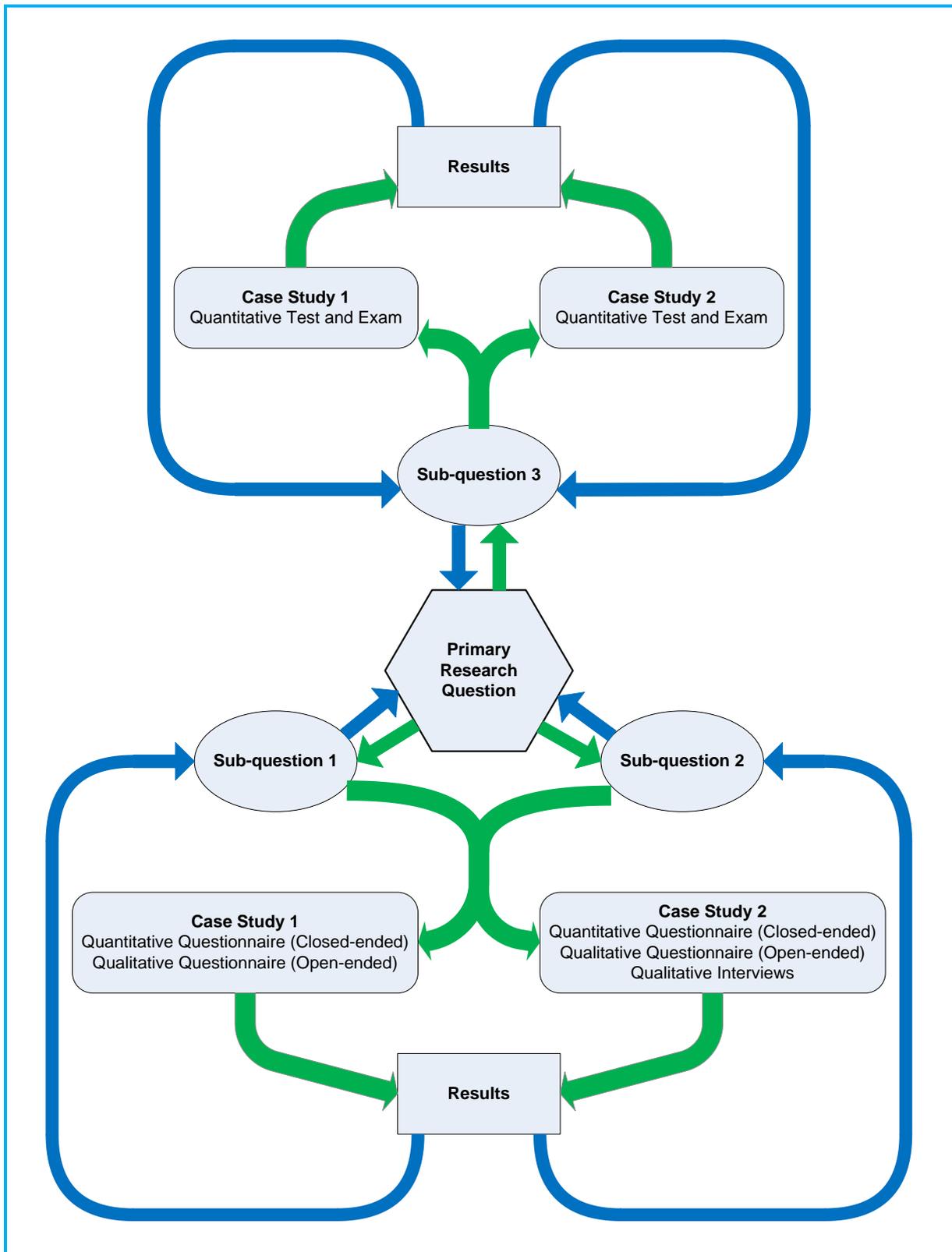


Figure 4.1 The road map that binds together the entire study

The next section provides a detailed discussion of the research design and methodology.

4.3 Research design and methodology

As mentioned in Section 1.7.1, the research design of this study is based on Creswell's (2009:5) *Framework for Design*. The three vertices of this framework, shown in Figure 4.2, represent the philosophical worldview proposed in a study, the selected strategies of enquiry, and the research methods used in the study. The aspects that relate to the current study are highlighted in red text in the figure. The same figure was originally presented in Chapter 1 as Figure 1.2, together with a brief initial discussion of the research design.

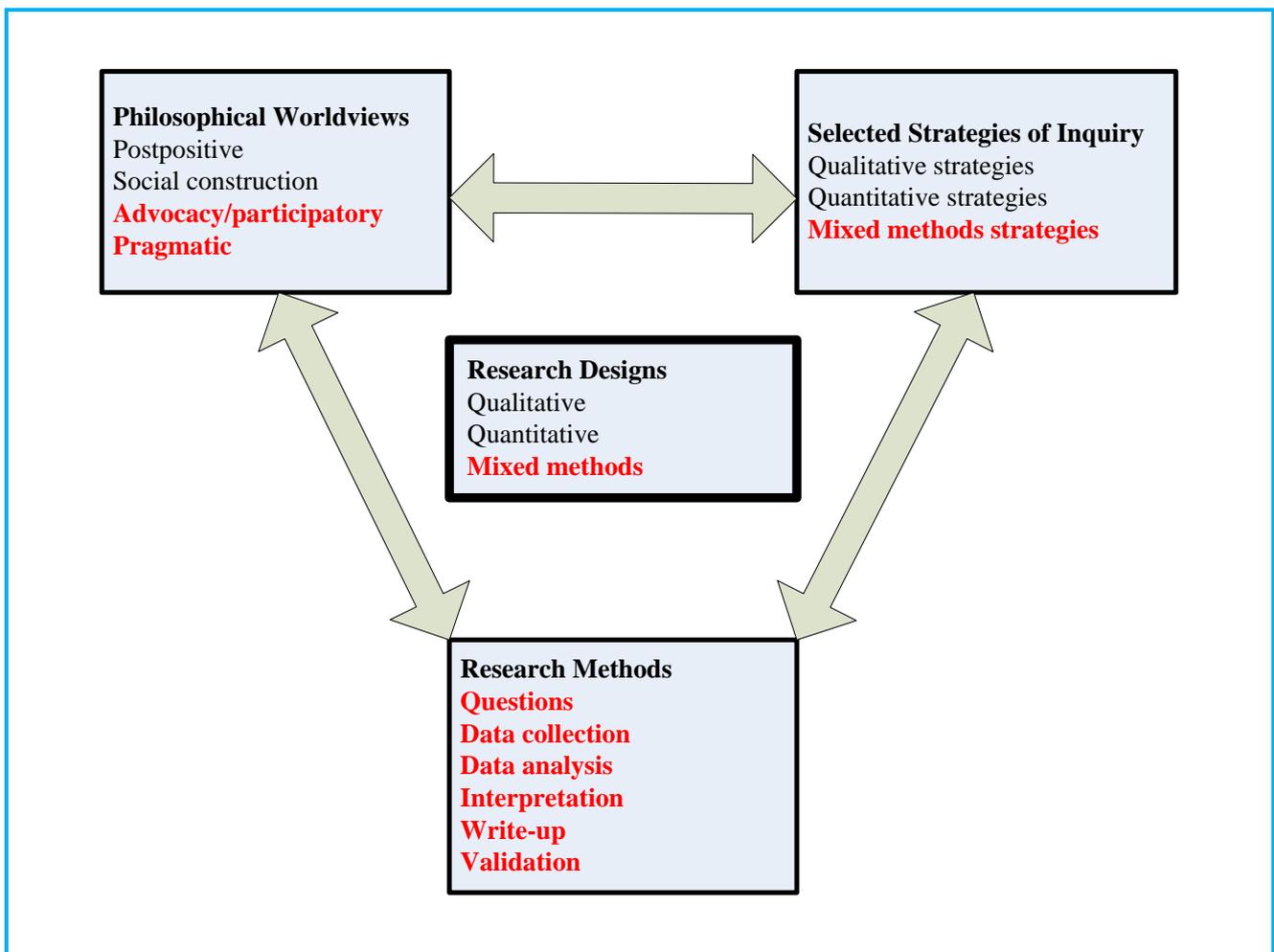


Figure 4.2 A Framework for Design – The interconnection of worldviews, strategies of inquiry and research methods (adapted from Creswell (2009:5))

Philosophical worldviews are regarded as basic sets of beliefs that guide one's actions and that comprise the underlying paradigms with which one approaches a venture. It is these paradigms and beliefs, shaped by researchers' personal experiences in their chosen field and incorporating elements of social behaviour, that subtly direct researchers in their choice of research strategy. The three strategies of inquiry are quantitative research and qualitative research on either end of the

continuum, respectively, and a mixed-methods approach residing in a central region. A mixed-methods strategy of inquiry was employed in this study to strengthen the quality of the research and to triangulate data collection. The research methods used in this study progress from the initial research questions through to data collection and analysis, followed by interpretation, write-up and validation.

The next sub-sections describe how each of the three components is implemented in this study. First, Section 4.3.1 addresses the philosophical worldviews.

4.3.1 Philosophical worldviews

The philosophical worldviews used in this study are advocacy/participatory and pragmatic. These are addressed in Sections 4.3.1.1 and 4.3.1.2 respectively.

4.3.1.1 The advocacy and participatory worldview

“An advocacy/participatory worldview holds that research inquiry needs to be intertwined with politics and a political agenda” (Creswell, 2009:9). This research indeed has the capacity to change the situations of the learners, the institution in which they study and the approaches adopted by academics. In the context of this study, advocacy research provides a voice for the learners. In enhancing their learning experience, it raises their consciousness and advances an agenda to improve their performance in academia and in the workplace, and hence holds potential for improvement in their lives.

4.3.1.2 The pragmatic worldview

There are different forms of the pragmatic worldview, but in general, this worldview arises out of practical actions, situations and consequences. Furthermore, “for the mixed methods researcher, pragmatism opens the door to multiple methods, different worldviews, and different assumptions, as well as different forms of data collection and analysis” (Creswell, 2009:11). Various data collection and analysis methods are used in this research in two consecutive cycles, as will be explained in the next two sections. The methods are pragmatically selected as being the most appropriate for the purposes they should serve.

The next section elaborates on mixed-methods research.

4.3.2 Strategy of inquiry: Mixed-methods

This study employed a mixed-methods strategy of inquiry, which according to Creswell and Plano Clark (2011), is a research design with philosophical assumptions as well as methods of inquiry. This approach can guide many phases of the research process and is outlined below. As a set of methods, it focuses on collecting, analysing and combining both *quantitative* and *qualitative data* in a single study or series of studies. By incorporating both qualitative and quantitative research, this form of design helps to broaden understanding, as well as using one approach to better understand, explain, or build on the results from the other (Creswell, 2009).

The research process was briefly addressed in Section 1.7.2, with reference to Figure 1.3. The same figure has been repeated in Figure 4.3, and is based on a model proposed by Oates (2006:33). The personal experience of the researcher and the motivation for this study (see Chapter 1), together with an investigation into existing literature (see Chapters 2 and 3), helped to define the research questions given previously in Section 4.2. The literature review also formed a strong foundation that contributed to synthesising the evaluation criteria used to design the questionnaire, as discussed later in Section 4.5.2. A case study approach was employed to answer the research questions, which is discussed in more detail in Section 4.3.2.1.

Sections 4.3.2.2 and 4.3.2.3 elaborate on the research processes used in Case Study 1 and Case Study 2 respectively. The mixed-method approach permeates the entire study and many sections relate to it. One of the advantages of mixed-methods research is its role in triangulation. The concept of method triangulation is addressed in Section 4.3.2.5 while more extensive explanations of the data generation methods are provided in Section 4.3.3. Finally, the data analysis, which adopts both quantitative and qualitative approaches, is discussed in Section 4.8. Quantitative data analysis in the form of reliability, descriptive and inferential statistical analysis, was applied to the closed-ended questions of the questionnaires, as well as to the learner data extracted from tests and examinations. Qualitative data analysis in the form of Applied Thematic Analysis (ATA), was applied to the open-ended responses from the questionnaires and interview transcripts.

As mentioned in the problem statement of Chapter 1 (see Section 1.3), both studies measured the performance of the ‘experimental’ group against that of a ‘comparison’ group with a similar success rate at first-year level. The experimental group contained participants chosen to attend the *Alice* workshop, and the comparison group comprised similar learners who were not exposed to the *Alice* intervention.

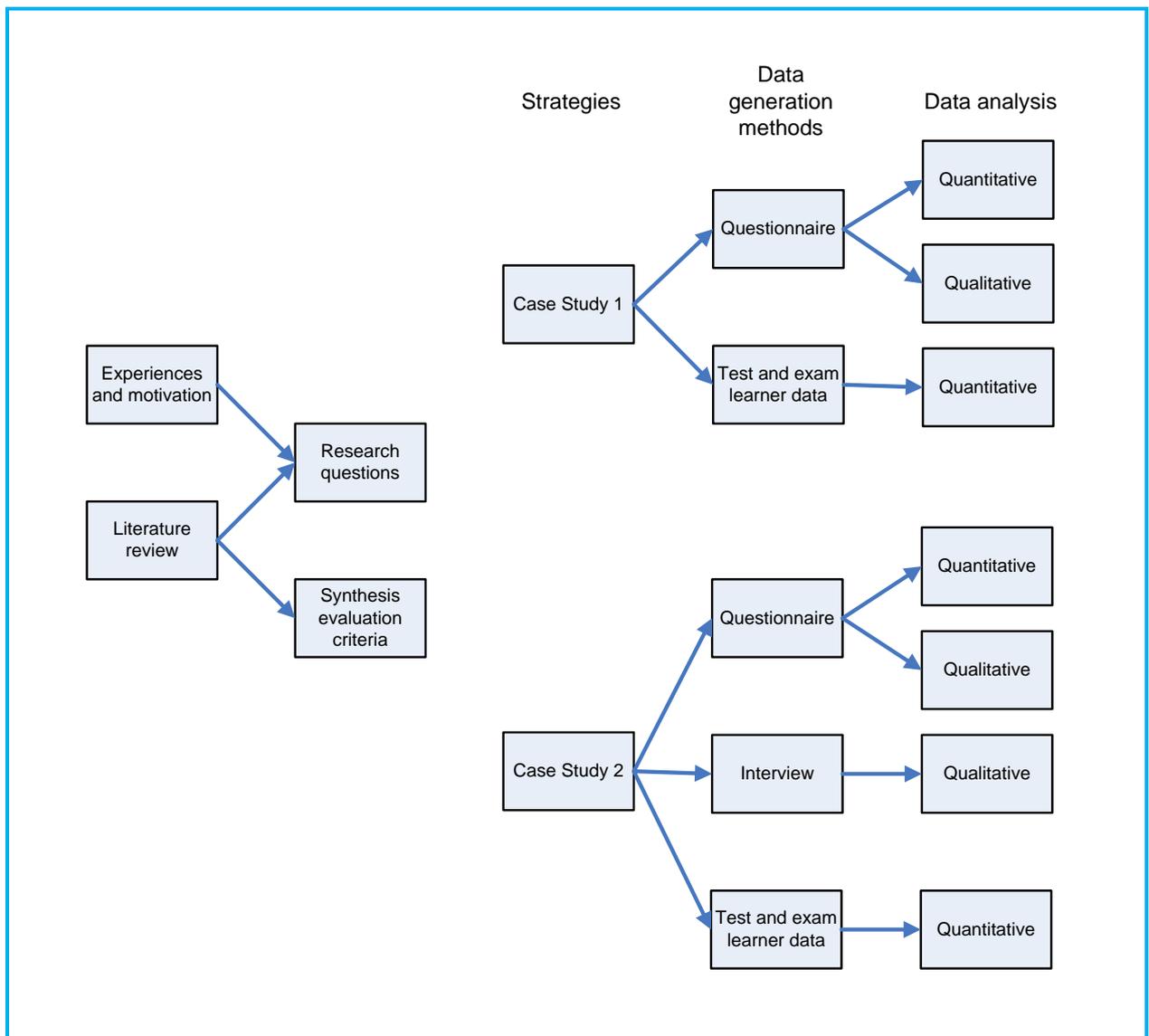


Figure 4.3 Research process (adapted from Oates (2006:33))

4.3.2.1 The case study research approach

Stake (cited by Creswell, 2009) provides a general definition of a case study as being a strategy of inquiry where a researcher investigates a program, an event, a process, an activity, or one or more persons in depth. Furthermore, cases are bounded by time and activities. In order to gather detailed information, researchers use various data collection methods over a period of time. Yin (2003) defines a case study as an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident. Case study research is used to analyse a limited number of units, as was the situation in this research. For instance, the number of learners is limited to samples of ND: IT learners from the Department of IT at DUT.

“Descriptive or ‘case-study’ research occurs where a specific situation is studied either to see if it gives rise to any general theories or to see if existing general theories are borne out by a specific situation” (Melville and Goddard, 1996:4). There was a strong case for using descriptive case study research in the present study since it supports the assessment of a *specific* VPE implemented within a *specific* institution. Oates (2006) describes a descriptive study as one that provides a rich, detailed analysis of a phenomenon within its context. The analysis “...tells a story, including a discussion of what occurred and how different people perceive what occurred” (Oates, 2006:143). This, too, was the case in the present research, as participants provided their perceptions in both quantitative and qualitative ways.

Rosnow and Rosenthal (1996) state that in order to learn about any behaviour, researchers should focus on at least two variables at the same time; that is, they must make two sets of observations that can be related to one another. The case study designs identified by Yin (2003) include *single-case* design and *multiple-case* design. Single-case designs are appropriate when dealing with specific cases, for example, an individual, a group such as a class, or an institution such as a university. Multiple-case designs facilitate comparisons between cases, and thus provide substantiation towards a conclusion, for example, several groups can also be investigated. In order to strengthen the study, to get richness of experiences, and to undertake an in-depth investigation, a longitudinal dual-case study method was followed in the present research, which is based on evidence collected from two major cases. The dual runs were conducted with cohorts of learners in 2011 and 2012 respectively. These studies are presented as Case Study 1 and Case Study 2, shown on the right-hand side of Figure 4.3, and will be addressed in more detail in the next two sub-sections respectively. Two sets of data were thus used to answer the three research questions.

Furthermore, Olivier (2004) identifies one of the challenges of a case study as being the ability to obtain knowledge that is useful, and not just interesting. Oates (2006) states that studying a particular instance can generate insight and knowledge that might also be relevant to other situations. In order to ensure the delivery of useful and insightful information, this is a short-term contemporary study, which examines what is occurring at the present moment. The present researcher thus observed what occurred, collected quantitative data, and also required participants to reflect in a qualitative way and explain the occurrences.

4.3.2.2 Case Study 1

In addition to the overall research process based on Oates (2006:33), and defined in Figure 4.3, the researcher designed an elaborated visual model, which is presented in Figure 4.4. It depicts a detailed breakdown of the research processes of Case Study 1, which is shown in the top right of Figure 4.3 and which incorporates some of Creswell's (2009) concepts of mixed-methods notation. The research processes transition from the data generation methods through to data analysis, followed by the interpretation of the findings, all of which are addressed in detail in Chapter 5. Following Figure 4.4 is an explanation of the sequence of steps implemented in Case Study 1, in line with the activities in the figure.

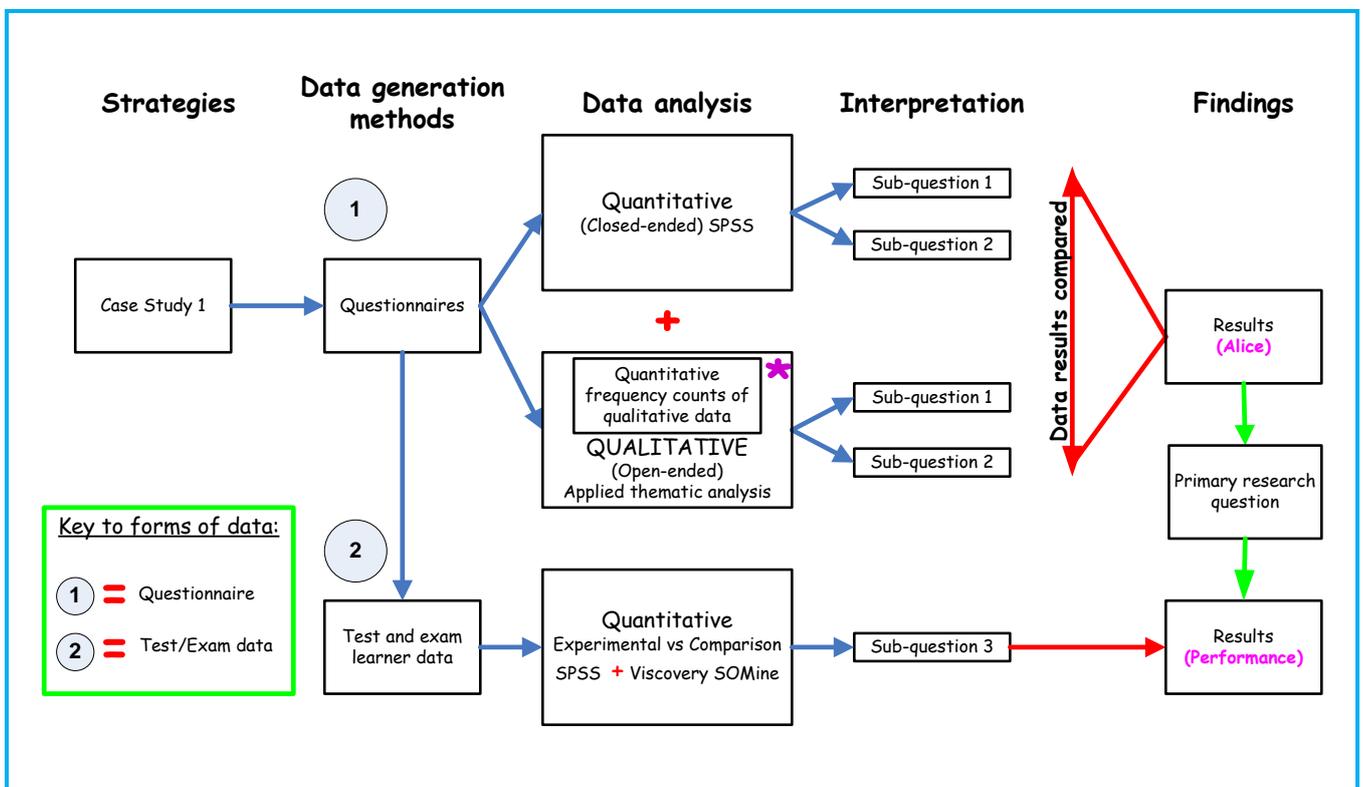


Figure 4.4 Detailed research processes of Case Study 1 (adapted from Oates (2006:33) and Creswell (2009:209-210))

1. Questionnaires were used to collect participants' responses, deriving information pertaining to the effectiveness and usability of the *Alice* VPE, as well as their experiences of the teaching and learning of OOP via *Alice*.
2. The test and exam data for the same participants who completed the questionnaire was extracted from the ITS system at DUT.

3. Quantitative data analysis was conducted, using SPSS on the responses to the closed-ended questions of the questionnaire, thus answering Sub-questions 1 and 2.
4. Concurrent qualitative data analysis was conducted, using applied thematic analysis on the responses to the open-ended questions in the questionnaire. This involved creating codes and themes qualitatively, then counting the number of times they occurred in the textual data. Creswell (2009) defines this process as data *transformation*, in which quantification of qualitative data enables the researcher to compare quantitative results with qualitative data. The quantification was done using frequency counts. This answers Sub-questions 1 and 2 and will be discussed further in Section 4.8.2. The capitalisation of 'QUALITATIVE' in the central block (depicted with a purple *) indicates prioritisation of the analysis of qualitative data. The QUALITATIVE/quantitative notation (Creswell, 2009:209-210) indicates that quantitative methods are embedded within the qualitative design. The emphasis placed on the qualitative analysis relates only to the open-ended section of the questionnaire.
5. Further quantitative data analysis was conducted, this time on the test and exam learner data of the experimental group in relation to the comparison group. This was achieved by using SPSS and thereafter doing data mining analysis with Viscovery SOMine, as a supplementary technique to the quantitative analysis, thus providing methodological triangulation. This answers Sub-question 3.

Note: Steps 3 to 5 can be conducted in parallel.

6. The fact that both quantitative and qualitative data analysis were performed, triangulated the findings and facilitated interpretation of the questionnaire results by providing an overall assessment of *Alice*.
7. The analysis of learner data obtained from tests and examinations, provided a comparison between the performances of learners in the experimental and comparison groups of Case Study 1.

4.3.2.3 Case Study 2

Similarly, the detailed research processes for Case Study 2 follow in Figure 4.5. Case Study 2 is shown at the bottom right of Figure 4.3. Figure 4.5 elaborates the sequence of steps implemented in Case Study 2 and includes aspects of the mixed-methods notation described by Creswell (2009:209-210). As was done with Case Study 1 in Figure 4.4, the detailed research processes in Figure 4.5 transition from data generation through to data analysis, followed by the interpretation of the findings, all of which are addressed in Chapter 6.

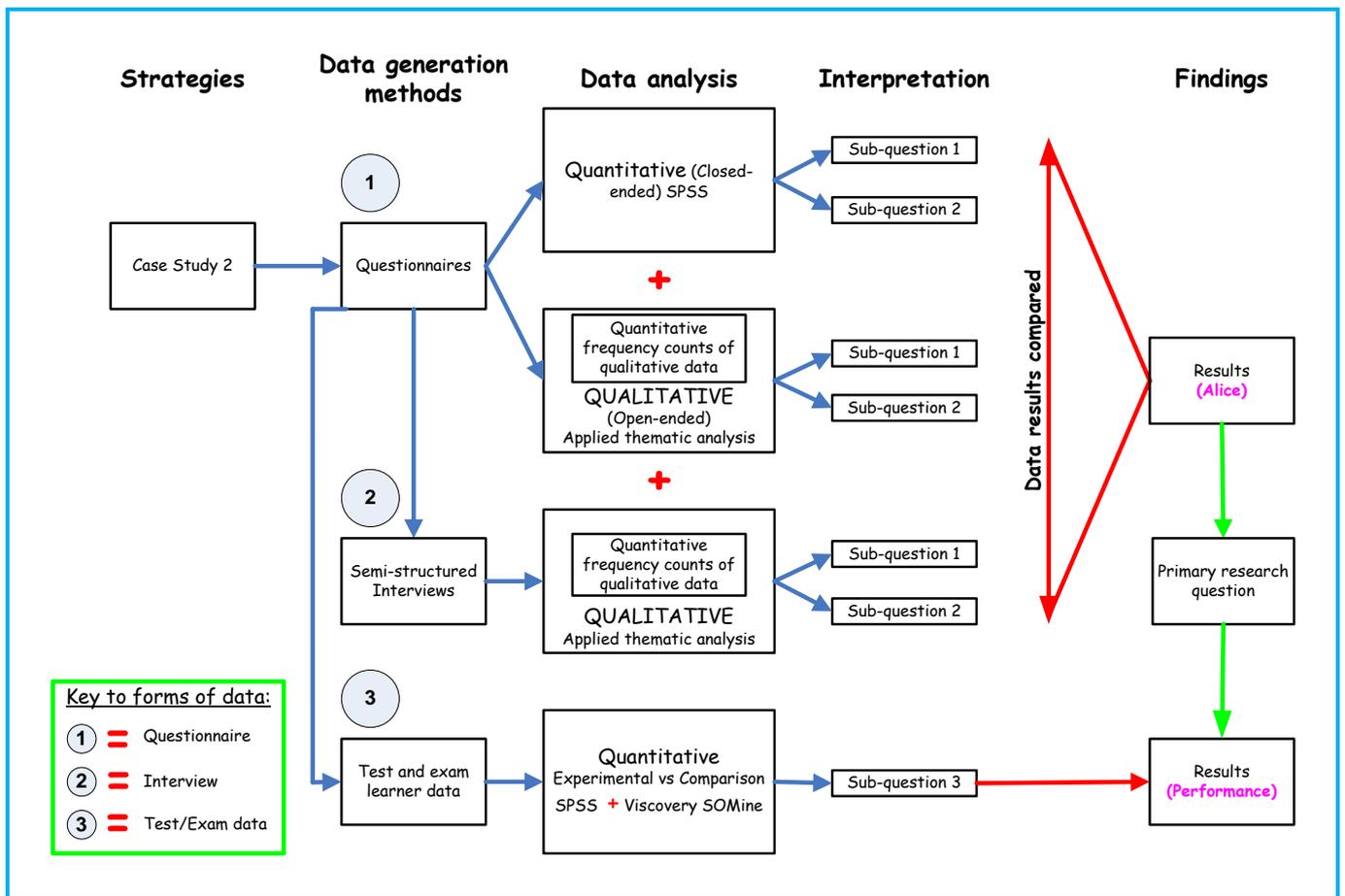


Figure 4.5 Detailed research processes of Case Study 2 (adapted from Oates (2006:33) and Creswell (2009:209-210))

Case Study 2 was the main empirical study. The problems encountered during Case Study 1, which will be discussed in Section 5.7, highlighted the need for a greater and sustained number of participants, as well as post-questionnaire interviews to strengthen the data. The seven steps of the research process pertaining to Case Study 1 are also applicable to Case Study 2. However, Case Study 2 includes further aspects, namely steps 8 and 9 listed below:

8. Semi-structured interviews with a sample of the participants took place three weeks after the questionnaire was completed.
9. ATA was used to qualitatively analyse the interview data. The quantification of the qualitative data was performed using frequency counts. This contributed to answering Sub-questions 1 and 2, with a broader scope of method triangulation.

4.3.2.4 Addressing the primary research question

Finally, the answer to the primary research question (see Section 4.2) was determined by establishing whether the implementation of the *Alice* VPE with DS2 learners at DUT had improved the subsequent performance and learning experience of participants. This was achieved by collectively understanding the findings from Case Study 1 and Case Study 2, and will be addressed in Chapter 7.

4.3.2.5 Triangulation

Cohen, Manion and Morrison (2011:195) define triangulation as “the use of two or more methods of data collection in the study of some aspect of human behaviour”. Triangulation provides an opportunity to identify inconsistent and contradictory evidence, as well as common findings, which researchers should analyse and interpret carefully (Mathison, 1988). Miles and Huberman (1994) state that triangulation should support findings by demonstrating that independent measures are consistent or, at least, do not contradict each other.

The types of triangulation applicable to the current study are *data/method triangulation* and *time triangulation*. When data obtained by different methods (this study uses questionnaires, interviews and assessment marks) and at different times, confirm each other, it leads to consistency of findings (Oates, 2006).

Data/Method triangulation

Cohen *et al.* (2011:196) state that methodological triangulation uses “the same method on different occasions or different methods on the same object of study”. Oates (2006:37) concurs that “the use of more than one data generation method to corroborate findings and enhance their validity is called method triangulation”. Guest, MacQueen and Namey (2012) posit that for any given study, if the same trends and themes emerge within data from different participant groups and data collection methods, then the validity of the findings is increased considerably.

Solving problems is likely to require the use of quantitative and qualitative approaches (Willemse, 2009). Concurrent collection of both quantitative and qualitative data, with the intention of comparing datasets to determine whether convergence occurs, or whether differences emerge, or if there is a combination of convergence and divergence, is known as the concurrent triangulation

approach (Creswell, 2009). Multiple forms of triangulation occurred in this study. As previously stated, both quantitative and qualitative research were conducted. In addition, data collection was triangulated by using questionnaires and interviews as instruments, as well as by extracting performance data from test and examination records. Furthermore, data analysis was triangulated by using both statistical analysis and applied thematic analysis.

Time triangulation

Time triangulation refers to stability over time which occurs when similarity emerges between different data gathered at the same time (Cohen *et al.*, 2011). According to Oates (2006:37), time triangulation occurs when “the study takes place at two or more points in time”. This study is longitudinal, employing time triangulation by observing the differences and similarities in findings from two cohorts of learners, over a dual-case study conducted in 2011 and 2012.

The next section explains the research methods employed in this study.

4.3.3 Research methods

The final vertex of Creswell’s (2009:5) Framework relates to the type of research methods employed. The methods used in this study progress from the initial research questions through to data collection and analysis, followed by interpretation, write-up and validation. Data collection entails identifying appropriate data sources and using them to gather relevant data details (Du, 2010). As explained in the previous sections, both Case Study 1 and Case Study 2 used questionnaires (Section 4.3.3.1) while Case Study 2 used interviews as well (Section 4.3.3.2) to collect data from the participants. Learners’ test and exam data (Section 4.3.3.3) was extracted from the ITS system with permission from DUT. The SPSS and Viscovery SOMine software packages were used to perform the data analysis and are discussed later in Section 4.8.1.

4.3.3.1 Questionnaires

Olivier (2004) defines the term *questionnaire* as a list of questions to be answered by the respondents themselves. Oates (2006) explains that a predefined set of questions are assembled in a pre-determined order. During the last week of the *Alice* workshop for both Study 1 and Study 2, participants were asked to provide informed consent and complete a questionnaire regarding their experiences with and opinions of *Alice*, as well as experiences and opinions regarding the teaching

and learning of OOP. Figure 4.6 shows some participants completing the questionnaires during the *Alice* workshop in 2012, that is, in Study 2.



Figure 4.6 Participants completing questionnaires during the 2012 *Alice* workshop (Photographs used with permission)

The questionnaire comprised closed-ended and open-ended questions to elicit responses from the participants. Closed-ended questions force the respondents to choose from a range of predefined answers, whereas open-ended questions leave the respondent to decide what unprompted answer to give (Oates, 2006). The data was processed by conducting quantitative analysis of the responses to closed-ended questions and qualitative analysis of responses to the open-ended questions, as addressed in Chapters 5 and 6. A detailed explanation of the design of the questionnaire is provided in Section 4.5.1, while an explanation of the analysis of responses is provided in Section 4.8.1.

4.3.3.2 Interviews

Oates (2006) describes an interview as a particular kind of conversation between two people, which has an unspoken set of assumptions that do not apply to regular conversations. The person undertaking the interview has the specific intention of obtaining information from the other. In this research, the researcher conducted post-questionnaire interviews in Study 2 with eighteen of the 55 participants who had engaged with the *Alice* visual environment. These semi-structured interviews were conducted three weeks after the workshop. Participants were required to complete a consent form prior to the interview and were asked for permission to have their session recorded (see Appendix C.2). Each session was audio-recorded and backed up with hand-written notes. The purpose of the hand-written notes was to reflect key ideas detected during the interview (Creswell, 2009).

The semi-structured interviews consisted of five open-ended questions on the main themes/domain areas. This led into open discussions, as each interviewee was probed further with more focused questions as the interview progressed. The interviewees were free to speak in detail on the issues raised, and also introduced issues of their own that were relevant to the theme at hand. The interviews elicited rich, unanticipated data. For the purpose of the present study, data analysis and interpretation in Chapter 6 is restricted to the five core questions, but further analysis can be conducted on other aspects in future research.

The interviews supplemented the responses to the questionnaire, by probing detailed aspects and obtaining additional rich information to confirm findings and strengthen the research. The design of the interview is explained in Section 4.5.3, while the use of ATA to observe patterns and trends in the resulting data is addressed in Section 4.8.2.

4.3.3.3 Test and exam learner data

Test and exam marks were used to compare the performances of the experimental group who participated in the *Alice* workshop with performances of the comparison' group who did not participate in the workshop. This comparison of learner performance was conducted in both Case Study 1 and Case Study 2. The findings will be addressed in Chapters 5 and 6.

The next section addresses the population and sample for this study.

4.4 Population and sample

The participants were second-year learners registered for DS2, within the ND: IT programme at DUT. Non-probability sampling was employed, whereby there is a specific choice in whom or what is selected (Steyn, Smit, Du Toit and Strasheim, 1994; Cohen, Manion and Morrison, 2007). Learners were selected based on criteria that related to pre-requisites. All the participants in both the experimental and comparison groups were doing OOP for the first time. The selection criteria differed slightly between Study 1 and Study 2 in terms of the number of subjects, as well as prerequisite subjects the learners should have passed in order to qualify to attend the *Alice* workshop.

Study 1: The learners chosen to participate were selected from those who had passed all four first-year subjects at first attempt, which contributed to similarity between participants and ensured that

they all had an average-to-good understanding of basic programming skills. The experimental group initially comprised 50 learners. As the study progressed, attrition occurred in the lunch hour *Alice* workshop, and only 21 respondents remained to the end of the intervention. Therefore, in the selection of learners for the comparison group, 21 learners with a similar success rate at first-year level were hand-picked from the remaining learners. Those in the comparison group had therefore been taught OOP by conventional teaching methods only. Furthermore, the process implemented stratified sampling, in that each stratum was homogeneous with respect to certain characteristics (Steyn *et al.*, 1994). For example, there was identical gender composition in each group.

Study 2: Participants were required to have passed both programming subjects (Development Software 1 and Technical Programming 1) together with one of the theoretical subjects (Information Systems 1 or Systems Software 1) at first attempt during their first year of registration in the ND: IT programme. This contributed to similarity between participants and ensured that they all had an average-to-good understanding of basic programming skills. The experimental group comprised 55 participants with the goal of 50 completing the intervention, but in fact all 55 remained and completed the questionnaire at the end of the *Alice* workshop. However, in the group of learners who were taught OOP by conventional teaching methods only, there were only 50 learners with a similar success rate at first-year level. To equalise numbers in the experimental and comparison groups, five participants of the 55 participants in the experimental group were therefore randomly chosen and omitted, resulting in 50 learners in both the experimental group and comparison group. This was done for the purpose of performing comparative quantitative data analysis on the learner data from tests and exams.

The section that follows, describes the design of the data collection instruments, that is, the questionnaire and interview.

4.5 Design of data collection instruments

Careful thought and planning was invested in the design of the questionnaire and interview protocols. Following the introductory explanations of the questionnaires and interviews provided in Sections 4.3.3.1 and 4.3.3.2 respectively, the next three sections, namely 4.5.1, 4.5.2 and 4.5.3, provide a detailed discussion on the design of the said protocols.

4.5.1 The questionnaire

The questionnaires in Case Study 1 and Case Study 2 were similar, apart from some improvements made to the questionnaire used for Case Study 1 before it was implemented in Case Study 2. The questionnaires are available in Sections B.2 and C.3 of the appendix respectively.

Section 1: The first section of the questionnaire requested the participants' profiles and demographic details. This included student number, surname, first name(s), gender, age, race, email address, contact telephone number and class group. In Case Study 1 participants were asked for details regarding subjects they had passed and subjects for which they were registered, but this was omitted from the Case Study 2 questionnaire, as these details were verified and checked before the study commenced. Minor refinements were made to the terminology of the actual questions in Case Study 2. However, the discussion that follows is applicable to both questionnaires.

Section 2: The second section of the questionnaire contained 25 closed-ended items, based on a 5-point Likert scale from 1 (Strongly Disagree) to 5 (Strongly Agree). Likert scales are used by respondents to indicate the degree to which various statements apply to themselves (Olivier, 2004). The questions related to varying aspects, namely:

- (a) Questions 1 to 10 investigated usability of the *Alice* VPE and were based on the interface design heuristics of Jakob Nielsen's ten general principles (or criteria) for evaluation (Dix *et al.*, 2004). This part of the questionnaire sought to answer Sub-question 2 of the study and is discussed further in Section 4.5.2.1.
- (b) Questions 11 to 19 emerged from the literature and findings of previous studies on the teaching and learning of programming, the challenges faced by OOP learners, and ways of improving the teaching of OOP. These questions were designed to answer Sub-question 1 and are discussed further in Section 4.5.2.2.
- (c) Questions 20 to 25 were based on criteria identified by the researcher. They emerged from her personal ten-year involvement in teaching OOP to IT learners. This part of the questionnaire also sought to answer Sub-question 1 of the study and is discussed further in Section 4.5.2.3.

Section 3: The third section of the questionnaire was Question 26, with six open-ended sub-questions. It elicited qualitative responses regarding the participants' experiences with the *Alice* environment, their consequent understanding of OOP, and improvements they would like to see in the teaching of OOP. Oates (2006) points out that qualitative data should be analysed in a quest for

relationships and themes. This study employed the applied thematic analysis (ATA) approach to examine participants’ responses to Questions 26.1 to 26.6. “The ATA approach is a rigorous, yet inductive, set of procedures designed to identify and examine themes from textual data in a way that is transparent and credible” (Guest *et al.*, 2012:15). This part of the questionnaire sought to answer both Sub-questions 1 and 2 of the study and will be discussed further in Section 4.5.2.4, with ATA elaborated in Section 4.8.2.

For Case Study 1, eight of the 70 questions were intentionally posed as negatively-phrased questions, so as to ascertain whether participants had read the questions carefully before answering. Similarly, eight of the 72 questions were negatively phrased for the questionnaire used in Case Study 2.

4.5.2 Synthesis of evaluation criteria for the questionnaire

An evaluation study requires an evaluation method or methods and also an appropriate set of evaluation criteria. This section presents the criteria used in the two case studies, classifying them into categories. Table 4.2 summarises each category and the domain it relates to. The theoretical criteria in this section form the basis of the evaluation statements in the questionnaires, although they were re-phrased to customise them for participants.

Table 4.2 Overall breakdown of items for the questionnaire, with corresponding domain area

Category	Domain area from existing literature
1 to 10	Nielsen’s ten heuristics used to assess the usability of <i>Alice</i>
11	The teaching and learning of programming
12 to 15	Challenges facing learners in learning OOP
16 to 19	Techniques to improve the teaching of OOP
20 to 25	The researcher’s experience
26	All of the above domains

A discussion of each category follows in Sections 4.5.2.1 to 4.5.2.4, with an explanation of how questions in the questionnaire were formulated.

4.5.2.1 Nielsen's ten heuristics used to assess the usability of *Alice*

Jakob Nielsen (1994), a distinguished engineer in 'human factors', which is the original term for human-computer interaction (HCI), presented ten general principles for user interface design. They are called 'heuristics' because they are more in the nature of rules of thumb than specific usability guidelines (Dix *et al.*, 2004). Nielsen's ten heuristics form the foundation for the first ten items of the questionnaire, thus allowing the usability of the *Alice* VPE to be evaluated with respect to a small set of classic usability principles.

Nielsen (2003) defines usability as a general concept that cannot be measured, but that is related to several usability parameters that are measurable. He posits that many aspects of usability can best be studied by simply asking the users. This can be done by questionnaires and interviews, which are useful methods for studying how users use systems and what features they particularly like or dislike. Apart from Nielsen's ten heuristics, there are other classic sets of heuristics such as Shneiderman's eight golden rules (Dix *et al.*, 2004), Norman's seven principles (Norman, 1990) and Powal's ten research-based heuristics (Gerhardt-Powals, 1996). If this research had been primarily a usability study, a set of criteria based on all the above would have been synthesised. However, the main focus of this research is the role of *Alice* in the teaching and learning of OOP, hence Nielsen's classic set is deemed adequate.

In addition, while addressing usability evaluation of *Alice*, the researcher paid attention to the three basic design principles that the developers of *Alice* had in mind when creating the software, as indicated in a paper by Conway *et al.* (2000) and the team of researchers from the University of Virginia and Carnegie Mellon University:

1. Choose a target audience and keep their needs in mind. In the context of the study conducted by Conway *et al.* (2000), the target audience included undergraduates who were not science or engineering learners.
2. Where possible, avoid mathematical and cryptic notation in the Application Programming Interface (API), for example, vectors and matrices. Furthermore, new terminology should be introduced only when needed.
3. Test the designs iteratively with real users, obtaining continual improvement in both learnability and usability of the system in the process.

The current study took cognisance of these three design principles within the chosen research methods explained in Section 4.3.3. Moreover, some of Nielsen's ten heuristics, which are

addressed by the first ten items of the questionnaire, are closely related to Conway's principles. The ways in which each of Conway's principles was implemented in this study, are as follows:

- (a) *Target audience and their needs*: As mentioned in Section 4.4, the researcher hand-picked the target audience for the *Alice* intervention. The audience comprised second-year IT learners registered for DS2 who met the pre-requisite subject criteria and were doing OOP for the first time. *Alice* was used to address their needs by providing additional teaching and learning of OOP concepts. Furthermore, the first principle mentioned by Conway *et al.* (2000) relates to Nielsen's second heuristic *Match between the system and the real world*. Ensuring the participants' ability to relate real-world objects to objects in the *Alice* system (see Sub-criterion 2.2 in Table 4.3), also focuses on the needs of the target group.
- (b) *Notation in the interface*: Conway's second principle relates to Heuristic 2 (see Sub-criterion 2.1 in Table 4.3), which investigates whether *Alice* avoids terminology that learners may find complex or unfamiliar.
- (c) *Iterative evaluation*: The two consecutive cycles in this study allow the *Alice* environment to be tested by two cohorts of learners, and consequently address Conway's third principle regarding iterative studies.

Items 1 to 10 in the questionnaire are based on the ten criteria, which are correspondingly founded on Nielsen's ten interface design heuristics, listed in Table 4.3. The first column contains the number of the criterion, e.g. 1, 6 or sub-criterion, e.g. 2.2, 3.1, etc. In the subsequent discussions of findings in Sections 5.4.2.1 and 6.4.2.1, the criteria are referred to by these numbers. The second column provides Nielsen's ten heuristics and, alongside each sub-criterion number, gives a brief explanation.

Note: The original heuristics refer to 'users' but the term has been changed to 'learners' to customise the heuristics for this research.

Table 4.3 Jakob Nielsen’s ten heuristics in relation to the *Alice* visual programming environment

General interface design heuristics, based on Nielsen (1994) and (Dix <i>et al.</i>, 2004), used to assess the usability of the <i>Alice</i> visual programming environment	
	Criterion
1	Visibility of the system status
1.1	The system always keeps the learner informed about what is going on through the use of appropriate, timeous feedback.
2	Match between the system and the real world
2.1	The system uses words, terms and phrases that are natural for learners to understand.
2.2	The system allows learners to relate to real-world objects used in their everyday experience.
2.3	The flow of information is arranged in a logical order.
3	User control and freedom
3.1	Learners control the system.
3.2	Learners can quickly and easily recover from mistakes by clicking on the undo or redo buttons, thus they can avoid serious errors.
4	Consistency and standards
4.1	Learners gain confidence due to the standard and consistent outcome of their actions.
4.2	The system avoids the use of words, situations or actions that have multiple meaning within different contexts.
4.3	The environment maintains a consistent look and feel.
5	Error prevention
5.1	The system prevents errors from occurring in the first place.
6	Recognition rather than recall
6.1	The required use of an object, the necessary actions to be taken, and the available options for selection are clear and visible at all times. Learners do not have to depend on remembering previous dialogues in order to proceed with the next step.
6.2	Instructions that facilitate the use of the system are visible and readily available at all times.
7	Flexibility and efficiency of use
7.1	The system caters for various levels of learners, from novice to expert.
7.2	The use of frequent actions, such as saving or opening a file, have the option to perform a task at a faster speed and improve interaction, i.e. through the use of shortcuts, combinations keys, toolbar icons etc.
8	Aesthetic and minimalist design
8.1	There is no irrelevant or rarely-needed information within dialogues. (Unnecessary information tends to distract the learner from focusing on more critical information at hand).
9	Learners are helped to recognise, diagnose and recover from errors
9.1	The occurrence of an error is followed with a clear, understandable and appropriate error message written in plain language.
9.2	A precise indication of the problem is provided.
9.3	The system proposes a constructive way to recover from the error.
10	Help and documentation
10.1	The help facility and documentation are easily accessible, easy to use, detailed to a specific task, and provide short and concrete steps to be carried out to achieve a certain task.

4.5.2.2 Teaching and learning programming, challenges faced by learners of OOP, and techniques used to address these challenges

Items 11 to 19 in the questionnaire are based on principles that were carefully formulated from the literature reviews in Chapters 2 and 3. These principles can be used as evaluation criteria and can be re-phrased as evaluation statements in questionnaires. The first column of Table 4.4, with the criterion numbers, e.g. 11.3, 14.2, is referred to during the discussion of interpretation of the findings in Sections 5.4.2.2 to 5.4.2.4 and 6.4.2.2 to 6.4.2.4. The second column lists the criteria that apply to this study on the effectiveness of using *Alice* at DUT. The third column provides evidence of their theoretical basis by citing the associated literature sources, while the fourth column indicates the location in the dissertation.

Table 4.4 Synthesis of evaluation principles/criteria mapped to reference and location

The teaching and learning of programming			
11	Learner attrition	References	Location in dissertation
11.1	Learners without prior programming experience are likely to be overwhelmed by the breadth and depth of material, thus contributing to learner attrition.	Moskal <i>et al.</i> (2004); Stiller (2009);	Sections 2.2 and 2.3.2.1
11.2	Attrition in programming courses suggests that a large number of learners initially express interest in computing, but the curriculum or pedagogical techniques tend to drive many of them away.	Quevedo-Torrero (2009); Wang <i>et al.</i> (2009);	
11.3	The text-based mode of writing object-oriented programs and the corresponding text-based output produced by the object-oriented programs gives many learners the impression that programming is dull and uninteresting.	Cliburn (2008)	
11.4	Concern over declining enrollments in Computer Science and the need to improve learner success rates leads instructors to develop approaches that make object-oriented programming more attractive to potential learners.		
Challenges faced by learners in learning object-oriented programming			
12	Lack of motivation for programming	References	Location in dissertation
12.1	The motivation of learners is a key issue if they are to learn.	Law <i>et al.</i> (2010); Jenkins (2001)	Section 2.2.7
12.2	Developing good programming skills typically requires learners to do intensive practice on programming exercises and to gain experience in debugging, which they cannot sustain unless they are adequately motivated.		

13	Fragile mechanics of program creation, particularly syntax	References	Location in dissertation
13.1	Learners often spend more time dealing with syntactical complexity and detailed issues of coding than on learning the underlying principles of object-orientation or solving the problem.	Salim <i>et al.</i> (2010); Carlisle (2009); Winslow (1996); Kelleher and Pausch (2005); Joint Task Force on Computing Curricula, IEEE (2001)	Sections 2.3.2 and 2.3.2.2
13.2	Immediate exposure to the terminology of programming syntax, such as do-while, if-then-else, switch-case, and for-next, may intimidate learners and result in poor performance, as well as learner attrition.		
13.3	The textual nature of most programming environments works against the learning style of learners.		
14	Identifying results of computation as the program runs	References	Location in dissertation
14.1	Psychological evidence indicates that feedback obtained immediately after an error is the most effective pedagogical action.	Gálvez <i>et al.</i> (2009); Gárcia-Mateos and Fernández-Alemán (2009); Wright and Cockburn (2000)	Section 2.3.2.3
14.2	Current educational tendencies are centred on the learner's point of view rather than on the instructor, with the intention of creating independent, reflective and life-long learners.		
14.3	A gulf of visualisation arises when a programmer has difficulty mapping between the observed behaviour of the running program and their mental model.		
15	Difficulty of understanding compound logic and learning design techniques	References	Location in dissertation
15.1	Core problems experienced by many novice learners are a basic lack of problem-solving abilities and difficulty in using basic concepts such as control structures, to create algorithms that solve concrete problems.	Gomes and Mendes (2007); Esteves <i>et al.</i> (2008); Yu and Yang (2010)	Section 2.2.1
How to improve the teaching of object-oriented programming			
16	Algorithmic thinking and expression	References	Location in dissertation
16.1	Algorithmic thinking and expression involves the ability to read and write in a formal language.	Dann <i>et al.</i> (2009)	Section 2.4.1
17	Abstraction	References	Location in dissertation
17.1	Many novice programming learners cannot express their creative thinking in terms of programming abstractions, because they do not grasp programming concepts on an abstract level. This may lead to learners writing code without really understanding each and every line in their code.	Salim <i>et al.</i> (2010); Dann <i>et al.</i> (2009); Vickers (2009); Bennedsen and Caspersen (2008)	Sections 2.3.2 and 2.4.2
17.2	Abstraction entails learning how to communicate complex ideas simply and to decompose problems logically.		
17.3	Abstraction occurs when we view something in general terms without focusing on its concrete details.		
17.4	The abstract concepts behind terminology of programming syntax need to be explained to learners in a tangible way that they can visualise and relate to.		

18	Objects-first strategy	References	Location in dissertation
18.1	Objects-first emphasises the most fundamental principle of object-oriented programming and design.	Joint Task Force on Computing Curricula, IEEE (2001); Phelps <i>et al.</i> (2005)	Sections 2.4.3 Section 3.4.4
18.2	The strategy presents the notions of objects and inheritance immediately and then goes on to introduce the more traditional control structures.		
18.3	One way to minimise the negative concerns and maximise the positive observations associated with object-oriented and objects-first programming is to provide learners with a rich, interactive, collaborative virtual environment that supports the programming experience.		
18.4	A firm sense of objects and a strong visual environment supports the above.		
19	3D animation authoring tools and visualisation	References	Location in dissertation
19.1	Visualisation and animation can make algorithms more understandable to learners.	Sajaniemi <i>et al.</i> (2008); Gomes and Mendes (2007); Carlisle (2009); Cooper <i>et al.</i> (2003)	Section 2.4.4
19.2	Learners understand programming concepts better when given a visual representation.		
19.3	Three-dimensional animation assists in providing stronger object visualisation and a flexible, meaningful context for helping learners to perceive object-oriented concepts.		
19.4	Three-dimensionality provides a sense of reality for objects.		
19.5	In the 3D world, learners may write methods from scratch to make objects perform animated tasks.		
19.6	Animations can provide a meaningful context for understanding classes, objects, methods, and events.		

4.5.2.3 The researcher's experience

Items 20 to 25 in the questionnaire are based on criteria derived from the researcher's personal experience in her capacity as a lecturer in the Department of IT and ten-year involvement in lecturing software development. Several of them are also encountered in the literature. The first column of Table 4.5, with the criterion numbers, e.g. 20.1 and 25.2, is referred to in discussing interpretations of the findings in Sections 5.4.2.5 and 6.4.2.5. The second column presents the pertinent criteria identified by the researcher, and the third column provides brief discussions of the relevance and importance of the criteria in the context of this study, relating them to concepts from the literature.

Table 4.5 Criteria derived from the researcher’s experience and relevance to the study

The researcher’s experience		
20	Appreciation of trial-and-error	Rationale
20.1	The bricolage problem-solver is an individual who uses readily available materials in a trial-and-error manner to construct a solution to the problem at hand.	The researcher found a prevalence of ‘bricolage problem-solvers’ amongst the learners in DS2, in line with Stillers’ (2009) argument concerning use of trial-and-error to solve a problem. See Chapter 2, Section 2.2.6.3.
20.2	Learners ‘try out’ individual animation instructions as they create user-defined methods. Learners can visibly see the effect that each new animation instruction has on the animation.	
21	Incremental construction approach	Rationale
21.1	Learners do not write the entire program first, but learn how to program incrementally. They naturally write one method at a time, testing and running each piece.	A common approach adopted by learners in DS2 involves writing a program in stages, testing and running each component incrementally. The incremental construction approach has been discussed by Zaccone <i>et al.</i> (2003). See Chapter 3, Section 3.4.3.2.
22	Improved understanding of OOP concepts	Rationale
22.1	Inheritance	These four core concepts form a foundation for discussion when learners are introduced to the section on OOP within the DS2 syllabus.
22.2	Methods – behaviour of objects	
22.3	Properties – characteristics of objects	
22.4	Functions	
23	Improved understanding of basic programming concepts	Rationale
23.1	Loops	Key concepts dealt with in DS1 include iteration, selection, variables and data types as well as event-driven programming.
23.2	If statements	
23.3	Data types	
23.4	Event-driven programming	
24	Ability to collaborate	Rationale
24.1	Learners are encouraged to combine their individual efforts and work in pairs.	Webb and Rosson (2011) state that learning occurs naturally when learners interact with each other in their environment. Hamer, Luxton-Reilly, Purchase and Sheard (2011) encourage learners to contribute to the learning of others and value the contribution of others. The ability of learners to collaborate is an integral element within the ND: IT programme. Learners are required to work in groups for the DS2 assignments in preparation for their DS3 industry-related group project.
25	Factors relating to the impact of <i>Alice</i> among DS2 learners	Rationale
25.1	The relevance of using <i>Alice</i> to learn OOP in relation to the current DS2 syllabus.	These aspects were intended to summarise the overall experience of the DS2 learners in terms of the impact <i>Alice</i> had made on their motivation to learn.
25.2	The keenness of learners to explore other VPEs.	
25.3	The interest of the learner in further interaction with <i>Alice</i> .	
25.4	To ascertain whether learners used the <i>Alice</i> software only during the workshop or whether they also used it at home.	
25.5	The impact of <i>Alice</i> on stimulating an interest in learning OOP.	

4.5.2.4 Open-ended questions

Item 26, with six open-ended sub-questions, elicited qualitative responses from the participants concerning their overall experience with the *Alice* VPE, as well as the teaching and learning of programming. The questions were followed by blank spaces for them to fill in as they saw fit, and were general enough for the participants to express their opinions freely in their own words. Table 4.6 presents the open-ended questions in the questionnaire, as well as the domain areas being addressed.

Table 4.6 Open-ended questions from the questionnaire with corresponding domain area

26	Open-ended questions	Domain area
26.1	Do you like <i>Alice</i> ? Explain your experience with using <i>Alice</i> .	Usability of the <i>Alice</i> VPE
26.2	Has working with <i>Alice</i> improved your understanding of OOP? Give reasons for your answer above.	Impact of <i>Alice</i> on the understanding of OOP
26.3	Name the problems you encountered in using <i>Alice</i> .	Usability of the <i>Alice</i> VPE
26.4	Briefly outline the challenges that you face in learning OOP.	Challenges faced by learners in learning of OOP
26.5	Do you agree that <i>Alice</i> as a VPE can help address some of these challenges, and why?	Impact of <i>Alice</i> on addressing the above-mentioned challenges
26.6	Explain what changes you would like to see in improving the teaching of OOP.	Useful ways to improve the teaching of OOP

The questions were assembled in an order that would provide the participants with a logical progression, starting with an easy Yes/No question and moving on to more detailed responses.

4.5.3 The interview

Semi-structured interviews (in Case Study 2) were conducted with a set of core questions directed to each interviewee on an individual basis. Based on responses to the core questions, further questions were raised and handled individually with each interviewee. The use of ‘Why’ and ‘Tell me more’ helped to elicit rich qualitative data.

Table 4.7 includes the core questions as well as the corresponding domain area that was addressed in the interview protocol. The basic interview protocol is in Section C.4 of the appendix.

Table 4.7 Core questions in the interview protocol and corresponding domain area

Core questions	Domain area
1. How easy is it to use <i>Alice</i> ? Why?	Usability of the <i>Alice</i> VPE
2. How would you describe your experience with the teaching and learning of programming at DUT? Tell me more.	Learners' personal experience with the teaching and learning of programming at DUT.
3. What are the challenges you are faced with in learning OOP?	Challenges faced by learners in learning of OOP
4. How would you like to see the teaching of OOP improve?	Useful ways to improve the teaching of OOP
5. How has <i>Alice</i> impacted on your understanding OOP and addressing the challenges you mentioned?	Impact of <i>Alice</i> on the understanding of OOP and addressing the above-mentioned challenges

The first question served as an ice-breaker and was followed by four other questions that were broad enough to elicit various experiences and ideas.

The next section describes the pilot study.

4.6 The pilot study

The prime purpose of a pilot study is to try out the intended approaches and research instruments. Cohen *et al.* (2011) state that the pretesting of questionnaires is crucial to their success. Moreover, “a pilot has several functions, principally to increase the reliability, validity and practicability of the questionnaire” (Cohen *et al.*, 2011:402). Olivier (2004) states that the best way to avoid problems such as misunderstanding of instructions and the issue of respondents struggling to answer questions due to lack of information, etc., is to conduct a pilot study before the main study.

As mentioned in Section 1.7.4, a pilot study was conducted prior to Case Study 1, with a randomly selected group of five learners from the population chosen to participate in the *Alice* workshop. Olivier further suggests that the selection of the group should include atypical, as well as stereotypical, members of the population. As such, the composition of learners for the pilot study included at least one learner from each of four race groups, so as to provide heterogeneity, whereas the composition of the student body comes mainly from one racial group. Furthermore, the composition included both male and female learners.

The questionnaire was administered to this group, with learners being encouraged to identify questions they misunderstood; ambiguities or vagueness; questions that were difficult to answer; any lack of clarity in the instructions; and places where insufficient space was provided. They were also asked whether the length of the questionnaire was reasonable for the one-hour duration of the lunch break.

The researcher was present during the pilot to answer queries. Learners were allowed to think out loud whilst completing the questionnaire. The researcher informally interviewed them after they had completed it, as is suggested by Oates (2006). This exercise proved to be fruitful, since several comments given by these five participants were used to make adaptations and improvements to the questionnaire before it was distributed in Case Study 1.

The next section addresses the reliability and validity of this study.

4.7 Reliability and validity

Creswell (2009) states that data analysis in mixed-methods research should consider dual facets, namely checking both the validity of the quantitative data and the accuracy of the qualitative findings. Cohen *et al.* (2011:179) define validity as “a demonstration that a particular instrument in fact measures what it purports to measure”. It is important in both quantitative and qualitative research. The principles of validity and reliability that this study adheres to, are summarised in Section 7.5 of the Conclusion chapter.

Quantitative reliability and validity

Reliability refers to “consistency and replicability over time, over instruments and over groups of respondents. It is concerned with precision and accuracy. For research to be reliable it must demonstrate that if it were to be carried out on a similar group of respondents in a similar context, then similar results would be found” (Cohen *et al.*, 2011:199).

With regard to validity of the quantitative data analysis, the use of more than one data analysis tool contributed towards confirming the findings pertaining to learner performance. The primary tool used for the quantitative analysis was SPSS and the secondary tool was Viscovery SOMine.

Qualitative reliability

“In qualitative research, reliability can be regarded as a fit between what researchers record as data and what actually occurs in the natural setting that is being researched” (Cohen *et al.*, 2011:202). To achieve qualitative reliability, the researcher’s approach should be consistent across different projects (Creswell, 2009). The reliability procedures followed for this study are as follows:

- (a) A qualitative codebook is a table or record of predetermined codes that the researcher uses to code data (Creswell, 2009). The present researcher developed a codebook and used it for analysing qualitative textual data, whilst ensuring that all codes had clear definitions. She consistently referred to the codebook during the coding process. A highly descriptive and precise codebook facilitates data comparison when using the same codes in a different study (Guest *et al.*, 2012), as is the case in this research, where the same codebook in Case Study 1 and Case Study 2 for qualitative analysis of data derived from the questionnaires and the interviews.
- (b) Prior to coding and analysis, each interview transcript in Case Study 2 was transcribed verbatim without losing the richness and accuracy of the participant responses, and was double-checked by the researcher.

Qualitative validity

“In qualitative data, validity might be addressed through the honesty, depth, richness and scope of the data achieved, participants approached, extent of triangulation or objectivity of the researcher” Cohen *et al.* (2011:179). To ensure qualitative validity, the researcher must check accuracy of the findings by using different validity strategies (Creswell, 2009). The validity procedures employed in this study are as follows:

- (a) The triangulation of open- and closed-ended questions, as well as the use of interview data and methods (see Figure 4.5 and Section 6.8) allowed the researcher to establish themes from multiple sources by compiling converging ideas and perspectives of the participants; thus increasing the validity of the findings. Guest *et al.* (2012) concur that the emergence of common themes from different groups of participants and different data collection methods, increases the validity substantially.

- (b) The use of rich, thick descriptions connected the findings of the literature as discussed in Chapters 2 and 3 to the empirical evaluation work discussed in Chapters 5 and 6, thereby contributing to the validity.
- (c) In cases where the opinion of the researcher was expressed, it was indicated that such statements were contributions made by the researcher.
- (d) Quotes made by participants were used verbatim to support themes and interpretations (see Section 7.3), thereby increasing the validity of the findings.
- (e) The dual-case study was conducted over a two-year period, allowing comparison of the findings from two different case studies and thus increasing validity.

The section following discusses the tools and methodologies employed during the data analysis processes in the present study.

4.8 Data analysis process

An introduction to the data analysis processes was provided in Section 1.7.4, explaining how it fits into the research processes depicted in Figure 4.4 and Figure 4.5. Data analysis, as explained by Creswell (2009), involves preparing the data for analysis; conducting different analyses; drawing a deeper understanding of the data; representing the data and making an interpretation of the larger meaning of the data. The next sub-sections will outline the tools and methodologies used to perform analysis in this research. Section 4.8.1 discusses how quantitative data analysis was conducted in this study, while Section 4.8.2 addresses qualitative data analysis.

4.8.1 Quantitative data analysis

Quantitative data analysis was performed using SPSS on participant demographic data, namely gender, race and age, as well as on closed-ended responses to the questionnaire. The tool was also used to perform primary statistical and mathematical analysis on learner data from tests and examinations. The implementation of SPSS for analysing numeric data falls within the bottom right quadrant (quadrant D) of the model proposed by Guest *et al.* (2012:6), shown in Figure 4.7. Furthermore, the results from SPSS are presented in the form of tables, graphs, cross tabulations and other figures, which forms the interpretation of patterns in numeric data, as recommended in quadrant B of Figure 4.7.

A data mining technique was employed for further analysis of the test and exam data, using Viscovery SOMine, with results presented in the form of graphical component maps. This secondary analysis approach also provides statistical processing of numbers (quadrant D), with qualitative interpretation of patterns that emerge from the data (quadrant B).

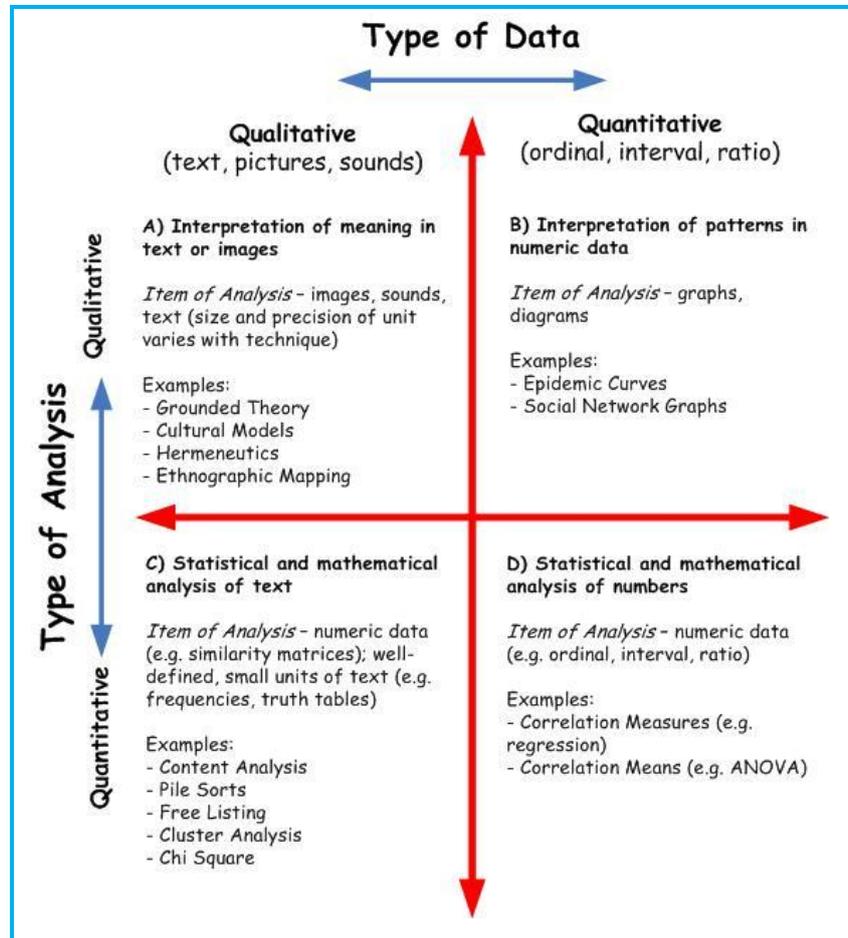


Figure 4.7 Qualitative and quantitative data analyses (adapted from Guest *et al.* (2012:6))

These software tools used for quantitative data analysis are discussed in the next sub-sections.

4.8.1.1 SPSS

The Statistical Package for the Social Sciences (SPSS) offers a comprehensive set of tools and solutions for flexible statistical analysis and data management. SPSS can be used to process various types of data and files, and has facilities to generate tabulated reports, charts, plots of distributions and trends, and descriptive statistics (Data analytics and reporting with IBM SPSS, no date). According to the Harvard-MIT Data Center, SPSS has been in use since the late 1960s (Guide to

SPSS, no date). It has been used in this study to generate reliability statistics as well as descriptive and inferential statistics. These various types are discussed in this section.

Reliability Statistics

Reliability refers to the property of a measurement instrument that causes it to give similar results for similar inputs. Cronbach's alpha is a measure of reliability. More specifically, alpha is a lower bound for the true reliability of the survey. Mathematically, reliability is defined as the proportion of response variability that is a consequence of differences between respondents. This implies that answers to a reliable survey differ, not because the survey is confusing or has multiple interpretations, but because respondents have varying opinions (SPSS Software, 2008).

Descriptive statistics

Descriptive statistics describe the organisation and summarisation of quantitative data. This study uses both univariate and bivariate analysis, which are most appropriate for descriptive statistics. Univariate analysis is concerned with measures of central tendency and measures of dispersion. The most appropriate measure of central tendency for interval data is the mean and the most appropriate measure of dispersion for interval data is the standard deviation. Bivariate analysis concerns the measurement of two variables at a time (Lind, Marchal and Mason, 2002).

Descriptive statistics are useful as they summarise results for an experiment, thereby also allowing for more constructive research after detailed analysis. Furthermore, descriptive data analysis aims to describe the data by investigating the distribution of scores on each variable, and by determining whether the scores on different variables relate to each other.

Inferential statistics

Inferential statistical analysis is concerned with the testing of hypothesis and allows the researcher to draw conclusions from sample data that relate to populations (Lind *et al.*, 2002). The t-test is used to compare the values of the means from two different samples and to test whether it is likely that the samples come from populations having different mean values (Kerr, Hall and Kozub, 2002).

4.8.1.2 Viscovery SOMine – Data mining software tool

The Viscovery SOMine tool is introduced in considerable detail, since it may not be familiar to the readers. This study uses data mining, which is based on the concept and algorithm of Self-Organising Maps (SOM), an unsupervised Neural Network (NN) method.

What is data mining?

The concept of data mining refers to the ‘mining’ or discovery of new information from large amounts of data. It is undertaken in terms of patterns or rules (Elmasri and Navathe, 2000). Du (2010:3) affirms that “data mining normally refers to the integral step of the ‘knowledge discovery in databases’ process that discovers and outputs hidden information patterns from prepared raw data”. According to Romero, Ventura and García (2008), data mining in the context of education is an emerging discipline that develops new methods for exploring the unique forms of data that occur within education. Furthermore, Romero *et al.* (2008) explain that the most useful data mining tasks and methods are statistics, visualisation, clustering, classification and association rule mining. The use of these methods in this study tends to uncover new, interesting and useful knowledge based on learner data that can be used in the prediction of future trends and behaviours.

Four phases of data mining used in the current study

Rob and Coronel (1997) identify four phases in data mining:

- (a) Data preparation – the main data sets to be used by the data mining operation are identified and cleansed from any data impurities;
- (b) Data analysis and classification – the data is studied to identify common data characteristics or patterns;
- (c) Knowledge acquisition – the data mining tool selects the appropriate modeling or knowledge acquisition algorithms;
- (d) Prognosis – the findings of data mining are used to predict future behaviour and forecast outcomes.

These four phases contribute towards an analysis framework implemented during the data analysis processes in this study and will be addressed in Chapters 5 and 6.

What are Self-Organising Maps?

The Self-Organising Map (SOM) was first introduced by Professor Teuvo Kohonen. It employs an unsupervised learning algorithm, which is a neural network technique that has been used successfully to electronically classify and analyse many types of data without supervision (Anniroot and Dean, 2005). Deboeck and Kohonen (1998) define neural networks as collections of mathematical techniques that can be used for signal processing, forecasting and clustering. When producing a network by the Kohonen technique, information is stored in a manner that preserves topological relationships within training sets (Htike and Khalifa, 2010).

The non-linear regression technique can be trained to learn or discover relationships between input and output data, or to organise data in a way that discloses unknown patterns or structures. The SOM serves as a clustering tool, as well as a tool for visualising high-dimensional data. Research conducted by Yu, Li, Xu and Wu (2010) indicates that among the artificial intelligence data mining methods, the technique of self-organising maps based on Kohonen's neural network, has become one of the powerful techniques of data mining through cluster analysis.

Relevance of Viscovery SOMine to the current study

Viscovery SOMine is a high-potential data mining tool used for the visual analysis and exploration of numerical data sets. One of the main reasons for using SOM is that no prior assumptions are required regarding the distribution of the data. Furthermore, SOM can detect unexpected structures or patterns by adopting an unsupervised neural network. SOM also has the capability to generalise, which means that the network has the capability to recognise or characterise inputs that are different from those it has previously encountered (Deboeck and Kohonen, 1998).

Viscovery SOMine can import routines for various tools including Microsoft Excel (Viscovery SOMine, no date). Numeric data sets can therefore be captured into a Microsoft Excel spreadsheet. Based on similarities in the data, Viscovery SOMine automatically structures the original, complex data into logical clusters, and presents them as graphical two-dimensional maps, known as cluster maps. Such maps consist of several clusters, with each cluster depicted in a different colour. Figure 4.8 is an example of a cluster map that contains six clusters. These visualisations support the user in intuitively discovering, analysing, and interpreting relationships within the data. For example, a cluster could represent a group of learners who have achieved a similar average mark for tests and the exam. Further discussion on cluster maps follow in Section 5.6.2.2.

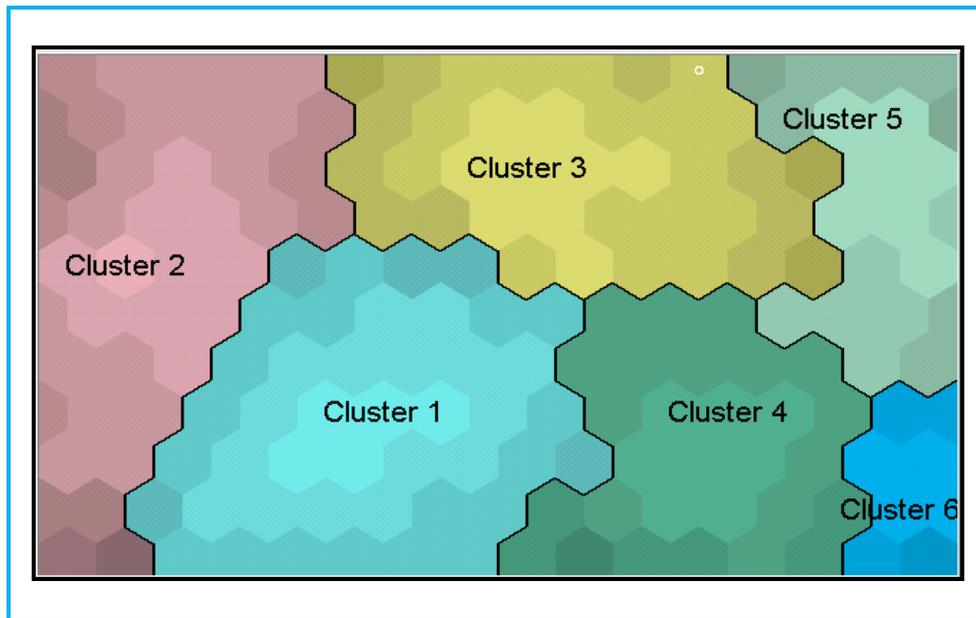


Figure 4.8 A sample cluster map generated from Viscovery SOMine

4.8.2 Qualitative data analysis

Qualitative data analysis was performed on the open-ended responses to the questionnaire and interviews. This was implemented using an *applied thematic analysis* approach, which is defined by Guest *et al.* (2012) as a methodological framework that integrates concepts from grounded theory, positivism, interpretivism, and phenomenology. It bears repeating the description of ATA given in Section 4.5.1, as a “rigorous, yet inductive, set of procedures designed to identify and examine themes from textual data in a way that is transparent and credible” Guest *et al.* (2012:15). The authors suggest that qualitative data analysis is more than just making sense of the data. Rather, it involves locating meaning within it. The goal of applied thematic analysis is therefore “a skillful expedition executed with forethought, appropriate tools, and systematic planning prior to entering unexplored terrain” Guest *et al.* (2012:49).

Furthermore, a quantification of the qualitative data derived from both the open-ended responses to the questionnaires and interviews allowed the researcher to draw comparisons from the statistical sample. Creswell (2009) defines this process as *data transformation*, which entails the qualitative identification of patterns, codes and themes, then tallying the number of times they occur in the textual data. With reference to Figure 4.7, Guest *et al.* (2012) state that most definitions of qualitative research include only quadrant A of the figure and overlook an entire group of analytic strategies available to them, namely the application of quantitative analytic procedures on qualitative data, as seen in quadrant C. This approach will be used in the present study.

The applied thematic analysis approach adopted by Guest *et al.* (2012) was implemented for qualitative data analysis in this research. The discussion is structured under three stages:

- (a) The definition of common terms referred to in qualitative analysis - Section 4.8.2.1.
- (b) The planning and preparation of the analysis, i.e. the first phase of ATA - Section 4.8.2.2.
- (c) An explanation of the codebook formulation, i.e. the second phase of ATA - Section 4.8.2.3.

4.8.2.1 The definition of commonly used qualitative terms

Guest *et al.* (2012:50) provide a clear definition of the terminology used in textual qualitative analysis. This is presented in Table 4.8.

Table 4.8 Definition of basic terms in textual qualitative analysis (adapted from Guest *et al.* (2012:50))

Term	Description
Data	The textual representation of a conversation, observation or interaction.
Theme	A unit of meaning that is observed (noticed) in the data by a reader of the text.
Code	A textual description of the semantic boundaries of a theme or a component of a theme.
Codebook	A structured compendium of codes, including a description of how the codes are related to each other.
Coding	The process by which a qualitative analyst links specific codes to specific data segments.

4.8.2.2 Planning and preparing the analysis

The most critical, but basic, analytical technique that should be employed when working with textual data, is reading the text. For this study, a basic analysis plan, relative to the amount of text to be analysed and availability of resources, was developed before the text was read. The steps that preceded the development of the analysis plan, included establishing analytical objectives, as well as considering the use of a quick and targeted analysis due to the relatively low volume of data in the study, followed by the selection of an analytical approach. Discussion follows of each step:

Establishing analytic objectives

In the context of this study, the development of an analysis plan commenced after the data had been collected, processed and cleaned. The core analysis objective was answering research Sub-questions 1 and 2. This required a good match between the view of what the study should achieve; the quality of the data available; as well as resources and time available.

The view: The crucial first step in applied analysis is defining the view. This should bring structure to subsequent decisions. For instance, the amount of detail that goes into a codebook will depend on the level of thematic identification envisaged, e.g. whether the codebook would be used by multiple researchers in a large-scale study. The current study is on a small-scale, with a single researcher and a small amount of data. Furthermore, “bounding the view” is a question of what data to use to answer specific research questions (Guest *et al.*, 2012:41). The level of investigation employed during the qualitative analysis in this study, related to participants’ responses from open-ended questionnaires and interviews. The analysis was narrowly targeted to inform the specific analytic domains in Table 4.6 and Table 4.7.

The quality of data: The amount of qualitative data in this study was relatively small, thereby warranting a detailed reading and re-reading of the text and simultaneous taking of notes.

The resources and time available: According to Guest *et al.* (2012), applied thematic analysis explicitly takes account of issues of resources and time, as well as considering the quality of the data in specifying an analytical research objective. In the context of this study, the researcher worked on the analysis unassisted. Due to her expertise as an academic in the field of the study for over ten years, she was able to complete it within a few weeks. Furthermore, the technology that was available included SPSS and Viscovery SOMine, which she used to quantify the qualitative frequencies and conduct counts of patterns and themes that emerged. The interview data was rich but, due to the limited resources and time frame available to complete the study, the researcher focused only on the targeted analysis objective of answering Sub-questions 1 and 2. However, she was able to store other useful information for future use in another study. This is the practice in applied thematic analysis, where excess data is systematically cataloged, so the researcher or others can go back to it as opportunities present themselves.

The quick and targeted analysis

Arising from the mixed-methods research design adopted in this study, the qualitative data collection helped to confirm and triangulate findings from the quantitative analysis. Exhaustive analysis would not have been worth the time and effort within the current study, so the quick and targeted strategy was used to inform the specific domains of inquiry listed in Table 4.6 and Table 4.7.

The analysis plan

Guest *et al.* (2012:35) propose a list of items to consider for inclusion in an analysis plan. These items are presented in the first column of Table 4.9, while the second column indicates how the item is addressed within the current study.

Table 4.9 The analysis plan for Case Study 1 and Case Study 2 (adapted from Guest *et al.* (2012:35))

Item in analysis plan	How item is addressed in this study
<i>The research question(s) that will be informed by the item</i>	Sub-questions 1 and 2
<i>The data that will be used</i>	Responses to the open-ended questions in the questionnaires and interviews.
<i>The researcher/s who will be involved in the analysis, and the roles of each</i>	The present researcher, in her capacity as an academic lecturer within the field of the study.
<i>The primary analytical purpose</i>	To evaluate the effectiveness of using the <i>Alice</i> visual environment in improving the teaching and learning of OOP and subsequent learner performance.
<i>The creation and definition of codes</i>	A basic codebook will be developed for the critical themes identified in Table 4.6 and Table 4.7. Coding will be done <i>in vivo</i> , i.e., coding for words or phrases within the text. Code definitions are to be developed in line with coding notes that are associated with specific text.
<i>The rules for applying codes to the data</i>	Only participant responses that relate to the critical domains of this study (see Table 4.6 and Table 4.7) will be coded.
<i>Ensuring that coding reliability is established</i>	(a) The use of a codebook ensures that all codes are clearly defined. It must be consistently adhered to during the coding process (see Appendix E.1). (b) The same codebook should be used to qualitatively analyse the data derived from the open-ended responses to questionnaires across Case Study 1 and Case Study 2, and interviews in Case Study 2. (c) Each interview transcript will be transcribed verbatim without losing the richness and accuracy of the participant responses, and will be double-checked by the researcher.
<i>The expected output</i>	Sub-sections of two chapters in the dissertation, respectively, presenting data collection and analysis for Case Study 1 and data collection and analysis for Case Study 2.

Choosing an analytical approach

This study employs the classic content-driven exploratory analysis approach (Guest *et al.*, 2012). The inductive nature of this approach emphasises what emerges from the interaction between the researcher and the respondent. The content generated, serves as a stimulus towards developing codes and identifying themes. The ATA approach was successfully used for subsets of textual data, thus providing both depth and breadth in addressing the research sub-questions.

4.8.2.3 Themes and codes

This section outlines the analytical strategy employed to segment the text for analysis. ATA adopts the following iterative process:

- (a) Identify features (i.e. themes); and
- (b) Define a boundary around each feature (i.e. text segmentation).

For small data sets, as in this study with only 18 interviewees, and 21 and 55 participants in the open-ended questionnaire surveys in Case Study 1 and Case Study 2 respectively, it was not deemed necessary to develop an explicit segmentation strategy. This was particularly due to the rapid identification and description of a limited number of major themes. Creswell (2009) affirms that this form of analysis is useful in designing detailed descriptions for case studies with a small number of themes or categories, where only five to seven themes emerge.

Themes can be classified into two major categories, namely *structural coding* and *content or emergent* themes. The latter describe what is observed or discussed in the context of the applied research design. In the present study, however, structural coding is used to identify the structure explicitly imposed on a qualitative data set due to the research questions and design of the study. Structural coding is elaborated below, followed by an explanation of the codebook, which is one of the most vital components of ATA.

Structural coding

The structural coding strategy was used for analysis of the participants' responses to questions, because of the underlying structure imposed on the qualitative data by the questions in the research instruments. It was the most appropriate approach, due to the consistent, structured layout of the open-ended section of the questionnaire, as well as the core questions in the semi-structured interviews. The interview text was segmented, based on the core questions formulated for the five domains. The structured guide used by the researcher in the coding process, provided clear starting

and ending points. The text segments within each interview transcript were analysed, with each text segment being a response to a particular question.

A combined *interview* and *questionnaire guide* was developed, as shown in Table 4.10, which includes the core questions from the questionnaires and interview protocols (see Appendices B.2, C.3 and C.4), along with the critical domain areas identified in Table 4.6 and Table 4.7. Using this guide as a starting point, the researcher developed five structural codes in the codebook for these questions. These are shown in Appendix E.1. Four of the codes are common to both interviews and questionnaires, while one, namely teaching and learning at DUT (TL@DUT) emerged only from the interviews.

Table 4.10 Structural coding: The interview and questionnaire guide (adapted from Guest *et al.* (2012:56))

Interview Questions	Codes	Themes
1. How easy is it to use <i>Alice</i> ? Why?	Usability	Usability of the <i>Alice</i> visual programming environment.
2. How would you describe your experience with the teaching and learning of programming at DUT? Tell me more.	TL@DUT	The learners' personal experience with the teaching and learning of programming at DUT.
3. What are the challenges you are faced with in learning object-oriented programming?	ChallengesOOP	Challenges faced by learners in the learning of OOP.
4. How would you like to see the teaching of object-oriented programming improve?	TechniquesOOP	Useful ways to improve the teaching of OOP.
5. How has <i>Alice</i> impacted on your understanding of OOP and addressing the challenges you mentioned?	AliceImpact	The impact of <i>Alice</i> on the learners' understanding of OOP and addressing the above-mentioned challenges.
Open-ended questions from questionnaire	Codes	Themes
1. (a) Do you like <i>Alice</i> ? Explain your experience with using <i>Alice</i> . (b) Name the problems you encountered in using <i>Alice</i> .	Usability	Usability of the <i>Alice</i> visual programming environment.
2. (a) Has working with <i>Alice</i> improved your understanding of object-oriented programming? Give reasons for your answer above. (b) Do you agree that <i>Alice</i> as a VPE can help address some of these challenges, and why?	AliceImpact	The impact of <i>Alice</i> on the learners' understanding of OOP and addressing the above-mentioned challenges.

3. Briefly outline the challenges that you face in learning object-oriented programming.	ChallengesOOP	Challenges faced by learners in the learning of OOP.
4. Explain what changes you would like to see in improving the teaching of object-oriented programming.	TechniquesOOP	Useful ways to improve the teaching of OOP.

Although the interview questions and open-ended questions in the questionnaire survey are similar, the idea with the interviews was that the interactive semi-structured format should probe further and elicit richer data than that obtained from the questionnaires.

The Codebook

“An applied thematic analysis codebook provides an efficient baseline for moving beyond basic description to an explanatory analysis” (Guest *et al.*, 2012:53). The development of the codebook is seen as a discrete analysis step. It is an iterative process in which the text is read, re-read and analysed, whilst being systematically coded into categories, types and relationships of meaning. The codebook is refined and developed during the iterations of re-reading and re-coding.

The structural codebook, provided in Appendix E.1, used the structured guide as the basis for code development. The structured guide comprises questions that form the basis for structural codes. Code definitions typically include both the main question as well as additional probing questions to enrich the response (Guest *et al.*, 2012). The researcher could easily denote the beginning and end of text segments by using each new question in the questionnaire and interview as an indicator. Each text segment included the learner’s response for each core question and any subsequent probes and dialogues about the question.

The next section addresses the ethical practices applied during this study.

4.9 Ethical considerations

Scientific research is a form of human conduct and should conform to norms and values acceptable to the scientific community. “The *epistemic imperative* refers to the moral commitment that scientists are required to make to the search for truth and knowledge” (Mouton, 2001:239). The researcher ensured that this study maintained acceptable, uncompromising ethical principles. She was officially authorised to conduct this research at DUT, where this study was conducted with learners as participants (see Appendix A.1). Furthermore, an application for ethical clearance was

approved by the College of Research and Ethics Committee (CREC) of the College of Science, Engineering and Technology (CSET) at UNISA, where the researcher is registered for MSc studies (see Appendix A.2).

All subjects have the right to anonymity and confidentiality. According to Isreal and Hay (2006), as cited by Creswell (2009:87), “Researchers need to protect their research participants, develop a trust with them, promote the integrity of the research, guard against misconduct and impropriety that might reflect on their organisations or institutions, and cope with new, challenging problems”. In line with ethical practice, all participants were required to sign an informed consent form before engagement in this research (see Appendices B.1 and C.1). It was explained to them that their exposure to *Alice* was for research purposes, and that the completion of a questionnaire was required. Information obtained would be used for academic purposes only and possibly for inclusion in academic publications.

The consent form contains the identification of the researcher; the sponsoring institution; a document describing the purpose of the research; benefits of participation; and the level and type of participant involvement. It indicates how the participants were selected. It guarantees anonymity and confidentiality of participants, together with the assurance that a participant could withdraw at any time. Furthermore, it includes details of a contact person, should queries arise. These elements included in the consent form are in line with those identified by Sarantakos (2005) and cited by Creswell (2009). Furthermore, the sub-sample of interview participants were required to sign an additional informed consent form, which also requested permission to audio-record the sessions (see Appendix C.2).

The researcher complied with the principle of maintaining objectivity and integrity and did not change the data or observations of the participants, as obtained from both the questionnaire and interview instruments.

Regarding the licensing of software, *Alice* is an open source teaching tool and is freely downloadable from www.alice.org (Dann *et al.*, 2012). For the purpose of this study, the software was successfully installed on computers in the laboratories of the Department of IT.

The next section concludes Chapter 4 with an overall summary.

4.10 Summary and conclusion

An overview of the research sub-questions and corresponding locations in the dissertation was presented in Section 4.2. Section 4.3 described the research design and methodology adopted in this study, explaining the philosophical worldviews; the mixed-methods research strategy employed in this study; and the research methods used for data collection. The target population and sample were addressed in Section 4.4. An explanation of the design of the data collection instruments, which included the questionnaire and interview protocols, was provided in Section 4.5. Section 4.6 outlined the purpose and nature of the pilot study. The reliability and validity measures conformed to during this study were explained in Section 4.7, after which the quantitative and qualitative data analysis processes were discussed in Section 4.8. The SPSS and Viscovery SOMine tools used for data analysis were also explained in Section 4.8. Finally, Section 4.9 reported on the ethical issues and practices adhered to during this research.

Chapter 5: Data Collection and Analysis, Case Study 1

5.1 Introduction

Chapter 5 is concerned with the analysis and interpretation of the quantitative and qualitative research data collected for Study 1, conducted during the academic year 2011. This chapter, as well as the next chapter, will attempt to find answers to the sub-questions listed in Sections 1.6.2 and 4.2, and interpret the case findings in relation to what was originally sought. The discussion of the findings in Chapter 5 also refers to literature sources presented in previous chapters. The researcher was thus able to assert the findings of similar studies; or in some instances highlight the differences in the context of the present study. The sub-questions are repeated below:

1. *What is the effectiveness, as perceived by learners, of using the Alice visual programming environment in addressing the challenges facing novice programming learners within the object-oriented domain?*
2. *How do learners experience the usability of Alice?*
3. *To what extent do the test and exam results of participating learners relate to those of similar learners who were not exposed to the Alice intervention?*

It must be borne in mind that Study 1, conducted in 2011, was a small-scale study, hence Chapter 5 provides preliminary answers to all three sub-questions. The next chapter, Chapter 6, presents the findings of the main empirical study, while further expounding the findings of this chapter.

Data collection was performed using questionnaires to elicit feedback from participants shortly after their workshop experience with *Alice*. The questionnaires constituted closed-ended and open-ended questions. The primary data analysis tool used for quantitative analysis was SPSS (Statistical Package for the Social Sciences) version 19.0, with results presented in the form of graphs, cross tabulations and other figures. This included quantitative analysis of the closed-ended questions in the questionnaire and the test and exam learner data. Qualitative analysis was performed on the open-ended questions in the questionnaire, using applied thematic analysis followed by a quantification of the qualitative analysis using frequency counts. Supplementary data mining analysis was conducted using Viscovery SOMine version 4.0, used only for quantitative data analysis of the learner data derived from tests and exams. Results are presented in the form of graphical cluster maps and component pictures.

An analysis framework comprising two phases was implemented to structure the data collection and analysis process, and is addressed in Section 5.2. Section 5.3 overviews the reliability of the questionnaire. The quantitative data analysis of the closed-ended questions in the questionnaire is discussed in Section 5.4, with the qualitative open-ended questions described in Section 5.5. The quantitative data analysis of the test and exam learner data is considered in Section 5.6. Section 5.7 briefly highlights the problems encountered during the course of Study 1. Finally, the chapter concludes with a summary in Section 5.8.

5.2 Analysis framework

An analysis framework was adopted for the data analysis and collection processes used in Chapter 5 and 6. The structure of this framework is similar to the processes of data analysis proposed by Rob and Coronel (1997) and Creswell (2009), which are outlined below.

Rob and Coronel (1997) considered a four-phase process used in data mining, as mentioned earlier in Section 4.8.1.2. To reiterate, their process includes:

- (a) Data preparation – the main data sets to be used by the data mining operation are identified and cleansed from any data impurities;
- (b) Data Analysis and Classification – the data is studied to identify common data characteristics or patterns;
- (c) Knowledge Acquisition – the data mining tool selects the appropriate modeling or knowledge acquisition algorithms, with only minimal end-user intervention during the actual knowledge-discovery phase;
- (d) Prognosis – the findings of data mining are used to predict future behaviour and forecast outcomes.

A similar data analysis process is highlighted by Creswell (2009), and comprises:

- (a) Preparing the data for analysis;
- (b) Conducting different analysis whilst drawing a deeper understanding of the data;
- (c) Representing the data; and
- (d) Making an interpretation of the larger meaning of the data.

The present researcher integrated the data analysis processes of both Rob and Coronel (1997) and Creswell (2009) into an analysis framework. This was used to structure the discussion of both the current and subsequent chapter and comprised the two major phases listed below:

- (a) Data collection and preparation;
- (b) Data analysis and interpretation.

A brief discussion of the salient elements of these two phases follows.

Data collection and preparation

Data preparation is an important part of the first phase. It makes an imperative contribution to the data analysis process in ensuring more accurate analysis and results. Du (2010:14) describes data preparation as “a complex process that may involve data collection and selection, pre-processing and formatting”. Rob and Coronel (1997) emphasise that the main data sets to be used, should be identified and cleansed from any data impurities.

Data analysis and interpretation

The second phase is *data analysis and interpretation*, and involves feeding the input data into analytical tools to establish patterns (Du, 2010). The prognosis component uses findings to predict future behaviour (Rob and Coronel, 1997), and therefore studies the data to identify common data characteristics, groupings, sequences or patterns. Data dependencies that reveal links or relationships can be identified. Results are presented in the form of tables, graphs, maps and others.

A detailed visual representation of the research process for Study 1 has been provided in Figure 4.4, Section 4.3.2.2. In order to facilitate the flow of discussion for the data analysis in this chapter, Figure 4.4 will be re-used in Sections 5.4, 5.5 and 5.6. Each of these sections includes the figure, but highlights the current focal point of discussion by means of yellow fill and a black striped border.

The next section addresses the reliability of the questionnaire.

5.3 Reliability of the questionnaire

The summative results of participation in Case Study 1 are presented in Table 5.1. A total of 21 participants responded to the questionnaire, as indicated by ‘N’. Sixteen of the 21 participants, representing 76.2% of the sample, provided a complete set of responses to the closed-ended questions in the questionnaire. In contrast, by virtue of their incomplete responses to the closed-ended questions, five participants (23.8%) were excluded from the sample.

Table 5.1 Case processing summary

		N	%
Cases	Valid	16	76.2
	Excluded	5	23.8
	Total	21	100.0

As mentioned in Section 4.7, reliability and, in particular, quantitative reliability refers to consistency and replicability over time, over instruments and over groups of respondents, and is concerned with precision and accuracy (Cohen *et al.*, 2011). Reliability is computed by taking several measurements on the same subjects, and can be measured, for example, by using Cronbach’s Alpha. A reliability coefficient of 0.70 or higher is considered as ‘acceptable’ (Cronbachs Alpha, no date). The overall reliability score of 0.838 in Table 5.2 indicates a high degree of acceptability, i.e. there is consistent scoring for the different categories within Study 1. The ‘N of items’ in column 2 of Table 5.2 refers to the 70 closed-ended questions in the questionnaire.

Table 5.2 Reliability statistics

Cronbach's Alpha	N of Items
.838	70

The next section describes the quantitative data analysis for the closed-ended questions in the questionnaire.

5.4 Quantitative data analysis: Closed-ended questions

The questionnaire, which is provided in Appendix B.2, includes a section containing a set of 25 categories comprising 70 closed-ended questions. The responses to the questions were in the form of options on a Likert scale. As mentioned in Section 4.5.1, eight out of 70 questions in the

questionnaire were negatively-phrased questions for Study 1. This was done intentionally, so as to ascertain whether participants had read the questions carefully before answering. Prior to analysis, the ratings to negatively-phrased questions were reversed so as to get a uniform format. This will be further explained in Section 5.4.2. Quantitative data analysis was performed on the numeric responses to Question 1 to 25. The component being addressed in this section is highlighted in Figure 5.1. Section 5.4.1 explains data collection and preparation, followed by data analysis and interpretation in Section 5.4.2.

Much of the material in Sections 5.4.2, 5.5.2 and 5.6.2, is based on a conference paper presented by Anniroot and de Villiers (2012) at the IADIS Information Systems 2012 Conference in Berlin (March) and published in the proceedings (see Appendix G.1). Case Study 1 was conducted specifically for the purpose of this masters study, regardless of the conference presentation. ('Anniroot' is the maiden name of the researcher, now 'Dwarika').

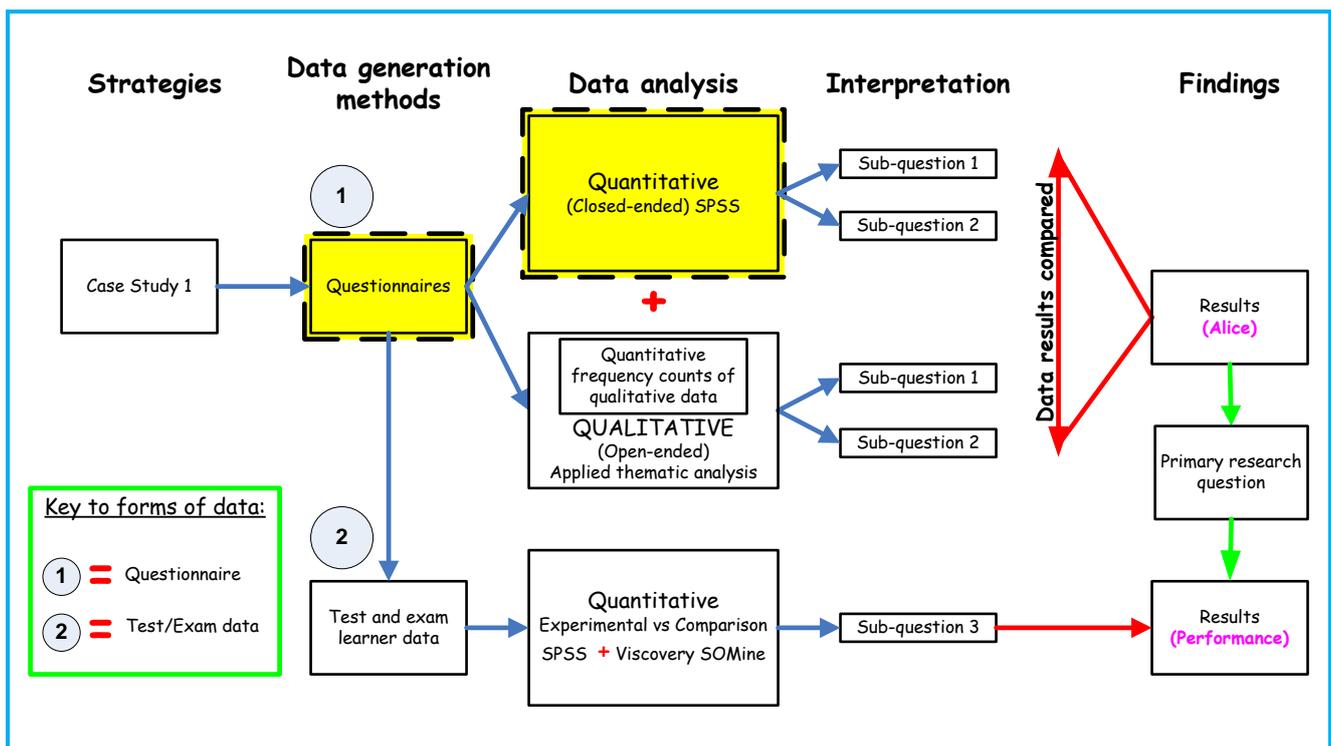


Figure 5.1 Detailed research processes of Case Study 1, highlighting the quantitative data analysis of the closed-ended responses to the questionnaire

5.4.1 Data collection and preparation: Closed-ended questions

The data collected from the questionnaires is presented in Appendix D.1, indicating the overall results and averages for each question. Furthermore, responses acquired from each participant were consolidated in a single data set in a Microsoft Excel spreadsheet, prior to being processed using SPSS. A segment of the participant responses is shown in Table 5.3. This data is given in its form after the reversal of ratings to negatively-phrased questions. This data included the student number (indicated by participant P1, P2,...P21), gender, race and the participant responses to the closed-ended questions. The responses to the closed-ended questions (i.e. Questions 1 to 25) were captured in accordance with the rating, where Strongly Agree = 5, Agree = 4, Neutral = 3, Disagree = 2 and Strongly Disagree = 1.

In this case, obtaining quantitative data from the questionnaire required very little data preparation. A null response was considered as missing data and left as a blank when preparing the data set, for example, see *Participant P10* under Question 4.3 in Table 5.3. However, as explained in Section 5.3, these learners were excluded from the sample during data processing due to null responses for certain closed-ended questions. Furthermore, SPSS requires numeric data sets. Therefore the ‘Participant’ column of such respondents was excluded from the data set before processing the data.

Table 5.3 A segment of the data set of responses to closed-ended questions in the questionnaire

Participant	Demographic Data		Responses											
	Gender	Race	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q3.1	Q3.2	Q3.3	Q4.1	Q4.2	Q4.3	...
P1	1	2	4	4	4	5	4	4	4	4	4	4	3	...
P2	0	3	3	5	5	4	4	3	3	4	5	5	5	...
P3	1	4	3	2	4	4	4	3	3	3	4	4	4	...
P4	0	4	3	4	4	3	3	4	4	4	4	3	3	...
P5	1	4	5	2	5	5	4	3	4	5	4	5	5	...
P6	0	3	4	5	5	4	4	4	4	3	5	5	5	...
P7	1	4	3	2	4	5	5	3	3	4	4	5	3	...
P8	1	4	4	5	5	4	5	3	3	4	5	5	4	...
P9	1	4	4	5	5	4	4		4	4	4	4	5	...
P10	1	4	5	5	5	5	5	5	5	5	4	5		...
P11	1	4	4	4	4	3	4	5	5	4	5	5	5	...
P12	1	2	4	4	4	3	3	4	4	4	3	5	4	...
P13	0	4	5	5	5	4	5	4	4	3	4	4	4	...
P14	1	3	4	4	5	5	5	4	4	3	5	5	5	...
P15	1	4	5	5	5	5	5	4	4	4	5	5	5	...
P16	1	3	4	4	4	4	2	3	4	4	4	4	4	...
P17	1	3	5	4	4	5	4	5	4	4	4	4	4	...
P18	1	4	5	3	5	5	5	4	5	5	4	5	4	...
P19	1	3	4	5	5	5	5	5	5	5	4	5	4	...
P20	1	1	4	4	4	4	4	4	4	4	4	4	4	...
P21	1	4	5	5	5	5	4	4	4		5	4	5	...

Key:

Gender: M = 1, F = 0
Race: Black = 4, Asian = 3, Coloured = 2, White = 1
Q1.1, Q1.2,...Qn: Strongly Agree = 5, Agree = 4, Neutral = 3, Disagree = 2, Strongly Disagree = 1

Furthermore, the ‘Gender’ and ‘Race’ columns were allocated numeric codes, according to the key shown in Table 5.4, which is known as a two-way frequency table or contingency table. In this case, the word contingency is used to determine whether there is an association between race and gender. According to Willemse (2009), data resulting from observations made on two different related categorical variables (bivariate) can be summarised in a table. The descriptive statistics based on the demographic information for Case Study 1, are shown in Table 5.4, and reveal that a total of 21 participants completed the questionnaire, comprising four females and seventeen males. Furthermore, the racial constitution comprised 4.8% White, 9.5% Coloured, 28.6% Asian and 57.1% Black participants.

Table 5.4 Participant profile table, including bivariate race * gender cross-tabulation

Key			Gender			Total
				0	1	
				Female	Male	
Race	1	White	Count	0	1	1
			% of Total	.0%	4.8%	4.8%
	2	Coloured	Count	0	2	2
			% of Total	.0%	9.5%	9.5%
	3	Asian	Count	2	4	6
			% of Total	9.5%	19.0%	28.6%
	4	Black	Count	2	10	12
			% of Total	9.5%	47.6%	57.1%
Total			Count	4	17	21
			% of Total	19.0%	81.0%	100.0%

5.4.2 Data analysis and interpretation: Closed-ended questions

This section aims to interpret the data collected from responses to the closed-ended questions and to present quantitative findings. The questionnaire was completed by the experimental group only, as it would have been irrelevant to the comparison group. A discussion of each of the 25 categories in the questionnaire follows in Sub-sections 5.4.2.1 to 5.4.2.5. As explained in Chapter 4, the questions, which are in the form of evaluation statements, are based on the theoretical evaluation criteria in Section 4.5.2, but suitably re-phrased. The findings of the closed-ended questionnaire analysis are discussed in relation to existing literature sources.

The first column of Table 5.5, Table 5.7, Table 5.8, Table 5.9 and Table 5.10 refers to the question numbers. The second column indicates the criterion on which that question is based. The distribution of percentages for participant responses is indicated under the sub-headings ‘D + SD’, ‘N’ and ‘A + SA’. The categories were collapsed so that the third column (D + SD) contains the sum of percentage responses for Disagree and Strongly Disagree, the fourth column (N) represents

a neutral response to the question, while the fifth column (A + SA) contains the sum of percentage responses for Agree and Strongly Agree. The average rating in response to each question is given in the sixth column, and matches the final column of Appendix D.1. The average rating indicates the degree of agreement, on a scale of one to five, where one indicates strong disagreement and five indicates strong agreement. In order to reflect an accurate average rate, the eight negatively-phrased questions, which span across Table 5.5 to Table 5.10, were re-phrased in a positive direction and the response percentages were reversed. For example, Question 9.1 in the questionnaire was negatively phrased as ‘*Alice* crashes while I’m using it’ (see Appendix B.2) but is re-phrased positively in Appendix D.1 as ‘*Alice* does not crash while I’m using it’.

The last column in these tables refers to the criterion numbers, which appear as the first column of Table 4.3, Table 4.4 and Table 4.5, the purpose of which is to make explicit the connection between the theoretical evaluation criteria in Chapter 4 and the evaluation statements in the questionnaire, as given in the current section. Finally, the category ratings at the bottom of each category are the average ratings of disagreement, neutral and agreement for that category. These rows are highlighted by pink fill beneath each of the 25 categories.

5.4.2.1 Usability of *Alice* in relation to Nielsen’s general interface design heuristics

Nielsen’s ten heuristics (Nielsen, 1994), as mentioned in Section 4.5.2.1, form the foundation for the first ten items in the questionnaire, i.e. they serve as the criteria for categories one to ten, thus allowing the usability of the *Alice* VPE to be evaluated with respect to a small set of classic usability principles. Nielsen (2003) posits that many aspects of usability can best be studied by simply asking the users. In this case, the participants who engaged with *Alice* were well suited to perform such an evaluation, after having attended the *Alice* workshop and having used the environment during lunch hours over a period of three-weeks. The percentage distribution of participant responses to the first ten questions is provided in Table 5.5.

Table 5.5 Percentage distribution of participant responses for usability of *Alice*

General interface design heuristics, based on Nielsen (1994) and (Dix <i>et al.</i> , 2004), used to assess the usability of the <i>Alice</i> visual programming environment						
1	Visibility of the system status	D + SD (%)	N (%)	A + SA (%)	Avg Rating	Crit#
1.1	I am always aware of what is going on in the system.	0.0	19.0	81.0	4.14	1.1
1.2	Whenever <i>Alice</i> is opened, the system indicates that it is in the process of loading the software.	14.3	4.8	81.0	4.10	1.1
1.3	When I save a world in <i>Alice</i> , the system indicates that files are being saved.	0.0	0.0	100.0	4.57	1.1
	Category Rating	4.8	7.9	87.3	4.27	
2	Match between the system and the real world	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
2.1	The system uses words, terms and phrases that I can easily understand.	0.0	14.3	85.7	4.33	2.1
2.2	The templates used for new worlds and the objects in the system gallery, relate to real-world objects that I encounter in my day-to-day experiences.	4.8	9.5	85.7	4.19	2.2
	Category Rating	2.4	11.9	85.7	4.26	
3	User control and freedom	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
3.1	I am comfortable with the level of control that I have over the system.	0.0	30.0	70.0	3.90	3.1
3.2	<i>Alice</i> allows me the flexibility to use the environment to perform a task.	0.0	19.0	81.0	4.00	3.1
3.3	I can recover from mistakes quickly and easily.	0.0	20.0	80.0	4.00	3.2
	Category Rating	0.0	23.0	77.0	3.97	
4	Consistency and standards	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
4.1	The <i>Alice</i> interface maintains a consistent look and feel.	0.0	4.8	95.2	4.29	4.3
4.2	The startup dialog box, play button, main menu and tab controls are clearly and consistently displayed.	0.0	4.8	95.2	4.52	4.2
4.3	The use of the mouse controls buttons and camera navigation controls are always available to move and arrange objects.	0.0	15.0	85.0	4.25	4.1
	Category Rating	0.0	8.1	91.9	4.35	

5	Error prevention	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
5.1	The <i>Alice</i> software allows me to focus on the objects and their associated properties and methods, rather than dealing with complex syntax.	0.0	9.5	90.5	4.24	5.1
5.2	The <i>Alice</i> interface design does not cause me to make errors.	5.0	20.0	75.0	3.90	5.1
	Category Rating	2.4	14.6	82.9	4.07	
6	Recognition rather than recall	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
6.1	The actions to be taken and options available for selection are clear and visible at all times.	4.8	14.3	81.0	3.90	6.1 6.2
6.2	I do not have to remember the information from a previous screen in order to proceed with the next one.	28.6	38.1	33.3	3.05	6.1
	Category Rating	16.7	26.2	57.1	3.48	
7	Flexibility and efficiency of use	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
7.1	<i>Alice</i> caters for novice to expert users.	0.0	23.8	76.2	3.95	7.1
7.2	I often use shortcut keys and/or combination of keyboard and mouse use to control the movement of objects.	19.0	52.4	28.6	3.14	7.2
7.3	I often use both the single view and the quad view to move, align and reposition objects on the screen.	28.6	28.6	42.9	3.33	7.1
	Category Rating	15.9	34.9	49.2	3.48	
8	Aesthetic and minimalist design	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
8.1	There is no irrelevant information in the <i>Alice</i> interface design that distracts me and slows me down.	9.5	23.8	66.7	3.67	8.1
	Category Rating	9.5	23.8	66.7	3.67	
9	Recognition, diagnosis and recovery from errors	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
9.1	<i>Alice</i> does not crash while I'm using it.	14.3	14.3	71.4	3.86	9.1
9.2	In cases where I encounter system errors, the system provides an appropriate error message in simple language.	5.0	25.0	70.0	3.80	9.2 9.3
	Category Rating	9.8	19.5	70.7	3.83	

10	Help and documentation	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
10.1	The four tutorials in the startup dialog box are useful in helping me to learn how to use <i>Alice</i> .	0.0	9.5	90.5	4.48	10.1
10.2	The example worlds in the startup dialog box are useful.	0.0	4.8	95.2	4.52	10.1
	Category Rating	0.0	7.1	92.9	4.50	

Due to the classic role of Nielsen’s (1994) heuristics in evaluating usability, special attention is paid to them by ranking the conformance of *Alice* to those heuristics. For seven of the ten interface design heuristics shown in Table 5.6, there is a general agreement ranging between 70.7 and 92.9 percent of responses and an average rating of 3.83 – 4.50. The first column refers to the heuristic number, followed by the category in column 2, in descending order of best-rated heuristics. Column 3 provides the average rate for the heuristic (i.e. the average of column 6, per category in Table 5.5). Column 4 in Table 5.6 provides the category rating as shown in the fifth column (A+SA) in Table 5.5. These findings indicate that, according to the participants, *Alice* provides good usability in seven of the ten respects.

Table 5.6 Ranking of heuristics indicating usability of *Alice*

#	Heuristic (Highest → Lowest Rating)	Average Rate for heuristic	Average% of A+SA (Category rating)
10	Help and documentation	4.50	92.9
4	Consistency and standards	4.35	91.9
1	Visibility of the system status	4.27	87.3
2	Match between the system and the real world	4.26	85.7
5	Error prevention	4.07	82.9
3	User control and freedom	3.97	77.0
9	Recognition, diagnosis and recovery from errors	3.83	70.7

A brief discussion on each of the five highest-rated heuristics, i.e. those that had more than 80% agreement, follows below in descending order of category ratings:

Help and documentation – Most of the experimental group (more than 90%) agreed that the four tutorials and example worlds available in the startup dialog box helped them when learning how to use *Alice*. During Case Study 1, the researcher spent the first *Alice* lesson demonstrating the help features of the VPE, which may have contributed to these findings. The method of teaching differed

for Case Study 2, in that handouts were given to the learners to aid the learning process during the *Alice* sessions, and when practicing in their own time.

Consistency and standards – A high percentage of participants (above 85%) found that the clear and consistent display of dialog boxes, buttons and controls on the *Alice* interface, made it easy to use.

Visibility of the system status – Learners unanimously agreed that the system informed them when files were being saved. Most learners (81%) were continually aware of the system status, even when *Alice* was in the process of loading.

Match between the system and the real world – Eighty-six percent (85.7%) adapted easily to words, terms and phrases used in the system. Moreover, the *Alice* templates and objects related to real-world objects that learners encounter on a daily basis.

Error prevention – Whilst 90.5% agreed that *Alice* allowed the learners to focus on coding, rather than dealing with complex syntax, only 75% found that the interface design prevented them from making errors.

In the other three cases, participants were somewhat less positive. These aspects are discussed below in descending order of category ratings:

In addressing *aesthetic and minimalist design* (Heuristic 8), 66.7% of the experimental participants disagreed that there is irrelevant information in the *Alice* interface design, i.e. they did not feel there was information that distracts users and slows them down (Question 8.1). Only 9.5% agreed with the statement and 23.8% of participants were unsure. In general, this indicates a positive impression and shows that users were not distracted, although the responses were less positive than those summarised in Table 5.6.

With regards to *recognition rather than recall* (Heuristic 6), only a third of the participants agreed that they did not have to remember the information from a previous screen in order to proceed with the next one (Question 6.2). However, 38.1% were unsure about this matter and 28.6% disagreed with it, implying that they did need to explicitly remember facts. In contrast, however, a high percentage (81%) of participants agreed that the actions to be taken and options available for selection were clear and visible at all times (Question 6.1).

The lowest-rated heuristic, *flexibility and efficiency of use* (Heuristic 7), resulted in 76.2% of participants agreeing that *Alice* caters for novice to expert users (Question 7.1). However, 52.4% of participants were unsure about how often they used shortcut keys and/or combination of keyboard and mouse use to control the movement of objects (Question 7.2). With regard to Question 7.3, a low percentage of 28.6 participants disagreed, and a further 28.6% were unsure about their frequency of use in both single and quad views to move, align and reposition objects on the screen. The low-to-average rate of neutral responses for this heuristic creates doubt about whether participants were comfortable with using these functionalities.

5.4.2.2 The teaching and learning of programming

This part of the questionnaire aimed to investigate the problem of *learner attrition*, generally faced by Computer Science departments and institutions (Salim *et al.*, 2010). This is also prevalent at DUT. A study of successive cohorts of first-time ND: IT entrants for the period 2007–2010 was conducted by the DUT Management Information Systems Department and the findings were made available to the researcher. The statistical results revealed dropout rates of 50% (2007), 51% (2008), 52% (2009) and 40% (2010), indicating the severity and relevance of the issue. The percentage distribution of learner responses to issues regarding the teaching and learning of programming at DUT, is provided in Table 5.7.

Table 5.7 Percentage distribution of participant responses for teaching and learning of programming

The teaching and learning of programming						
11	Learner attrition	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
11.1	I had a high level of comfort with computer usage prior to studying programming.	14.3	0.0	85.7	4.05	11.1
11.2	I am able to cope with the breadth and depth of the course work covered in the programming subjects of this course.	0.0	23.8	76.2	3.95	11.1
11.3	The current teaching methods and techniques used in programming subjects helped me to program successfully.	0.0	28.6	71.4	3.81	11.2 11.3
11.4	I had a high level of comfort with object-oriented programming prior to the <i>Alice</i> workshop.	14.3	33.3	52.4	3.38	11.4
11.5	I like to learn object-oriented programming in a more fun and interactive way.	0.0	4.8	95.2	4.62	11.4
	Category Rating	5.7	18.1	76.2	3.96	

Nearly eighty-six percent (85.7%) of participants expressed a *high level of comfort with computer usage* prior to studying programming (Question 11.1), which may have contributed to the consistently high percent (76.2%) being able to cope with the breadth and depth of course work in programming subjects (Question 11.2). It must be borne in mind that the sample of learners selected for Case Study 1 had passed all four first year subjects at first attempt. Their high level of comfort with computer usage prior to studying programming may have contributed to good performance during the first year of study.

It is notable that over three quarters of the experimental participants were *able to cope with the large volumes of course work* in programming subjects (Question 11.2) and that 71.4% agreed that the *current teaching methods and techniques used at DUT were helpful* (Question 11.3). None of the participants disputed either issue. According to Moskal *et al.* (2004), learners without prior programming experience are likely to be overwhelmed by the breadth and depth of material, thus contributing to learner attrition. Stiller (2009) further states that attrition in programming courses suggests that a large number of learners express interest in computing initially, but the curriculum or pedagogical techniques tend to drive many of them away. The high percentage of participant satisfaction pertaining to both the volume of course work and the teaching styles, indicates the unlikelihood that these are reasons for learner attrition. Further investigation of this problem is therefore warranted.

While 52.4% of the experimental participants indicated a *high level of comfort with OOP* prior to the *Alice* workshop, a third were unsure (Question 11.4). A very high percentage (95.2%) of participants enthused about *learning OOP in a more fun and interactive way* (Question 11.5). The participants were clearly motivated to learn OOP via *Alice*. Jenkins (2001) posits that motivation of learners is a key issue in learning. If *Alice* can contribute to raising motivation, it is an important factor in encouraging its use in tertiary institutions.

5.4.2.3 Challenges faced by learners in learning OOP

The information in Sections 5.4.2.3 and 5.4.2.4 is closely related to a paper by Annirroot and de Villiers (2012), presented at the IADIS conference in Berlin in March 2012 by the primary author. The paper was based entirely on data obtained as part of this MSc research.

Novice programmers face various challenges and difficulties in learning OOP, in particular:

- (a) Lack of motivation for programming;
- (b) Complex syntax, logic and semantics;

- (c) The need for immediate feedback and identifying results of computation as a created program runs; and
- (d) Difficulties in understanding compound logic and the application of algorithmic problem-solving skills.

Table 5.8 provides participant responses in relation to these core challenges. Some of the questions relate to general experiences of learning OOP, while others refer implicitly to features of *Alice*, without explicitly mentioning the name ‘*Alice*’.

Table 5.8 Percentage distribution of participant responses for challenges facing learners of OOP

Challenges faced by learners in learning object-oriented programming						
12	Lack of motivation for programming	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
12.1	I am motivated to learn programming.	4.8	4.8	90.5	4.38	12.1
12.2	I spend a lot of time intensively practicing programming exercises.	14.3	19.0	66.7	3.57	12.2
	Category Rating	9.5	11.9	78.6	3.98	
13	Fragile mechanics of program creation, particularly syntax	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
13.1	Learning the syntax and semantics of a programming language is challenging.	9.5	28.6	61.9	3.67	13.1
13.2	It would be easier for me to learn how to solve a problem and learn basic concepts of object-orientation without having to deal with brackets, commas and semicolons.	9.5	38.1	52.4	3.62	13.1
13.3	I have felt intimidated by direct exposure to programming syntax, such as do-while, if-then-else, switch-case, and for-next.	55.0	35.0	10.0	2.45	13.2
13.4	The textual nature of the conventional programming environments I use, makes it difficult for me to learn how to program.	47.6	47.6	4.8	2.52	13.3
	Category Rating	30.1	37.3	32.5	3.07	
14	Identifying results of computation as the program runs	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
14.1	I am able to identify errors and correct them using the feedback given by the program.	9.5	23.8	66.7	3.76	14.1 14.3
14.2	I am able to work independently on a program, from coding to testing.	4.8	23.8	71.4	3.81	14.2
14.3	My experience from running and debugging programs allows me to solve similar problems.	0.0	23.8	76.2	3.95	14.2
	Category Rating	4.8	23.8	71.4	3.84	

15	Difficulty of understanding compound logic	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
15.1	I am able to apply basic problem-solving techniques to create algorithms that solve problems.	0.0	23.8	76.2	3.95	15.1
15.2	I have a good understanding of pseudocode.	0.0	19.0	81.0	4.00	15.1
15.3	I use pseudocode to outline and understand the logic of a program before I start coding.	33.3	38.1	28.6	2.95	15.1
15.4	I am able to break down a large and complex programming task into smaller subtasks.	14.3	9.5	76.2	3.71	15.1
Category Rating		11.9	22.6	65.5	3.65	

Lack of motivation for programming: While 90.5% of the experimental participants agreed that they were motivated to learn programming (Question 12.1), only 66.7% admitted to spending a lot of time intensively practicing programming exercises, and a further 19% were unsure (Question 12.2). Law *et al.* (2010) state that in order to develop good programming skills, learners should do a great deal of intensive practice on programming exercises to gain experience in debugging. This cannot be sustained unless they are adequately motivated. Further investigation is needed to establish the reasons behind the lack of practice on the part of the learners.

Complex syntax, logic and semantics: Winslow (1996), Kelleher and Pausch (2005) and Carlisle (2009) present a strong debate, affirming that learners frequently spend more time dealing with syntactic complexity and detailed issues of coding than on learning the underlying principles of object-orientation or solving the problem. A good percentage (61.9%) of the experimental participants agreed that learning the syntax and semantics of a programming language was challenging, while 28.6% were unsure (Question 13.1). In order to release the learner from having to deal with complex syntax, *Alice* adopted the drag-and-drop feature. However, only 52.4% agreed that it would be easier to learn how to solve a problem and learn basic concepts of object-orientation without having to deal with brackets, commas and semicolons, while 38.1% were unsure (Question 13.2) (When using *Alice*, learners need not enter syntactic symbols such as brackets, commas and semicolons). Further research conducted by Salim *et al.* (2010) indicates that rapid exposure to the terminology of programming syntax, such as do-while, if-then-else, switch-case, and for-next, may intimidate learners and result in poor performance, as well as learner attrition. Nevertheless, 55% indicated that they were not intimidated by direct exposure to programming syntax, such as do-while, if-then-else, switch-case, and for-next. Thirty-five percent (35%) were unsure and only 10% agreed with the statement (Question 13.3). Forty-eight percent (47.6%) of participants in the experiment disagreed that the textual nature of conventional programming

environments, makes it difficult for them to learn how to program, while another 47.6% were unsure (Question 13.4). Carlisle (2009) suggests that the textual nature of most programming environments works against typical learning styles, but the results of this study do not appear to support that. Although the participants enjoyed *Alice*, it is important to note that many of the experimental group were also comfortable learning OOP by conventional means.

The need for immediate feedback and identifying results of computation as a created program runs: Psychological evidence offered by Gálvez *et al.* (2009) suggests that feedback provided immediately after an error is the most effective pedagogical action. Sixty-seven percent (66.7%) agreed that they were able to identify errors and correct them using the feedback given by the program (Question 14.1). Seventy-one percent (71.4%) felt that they were able to work independently on a program, from coding to testing (Question 14.2), reaffirming the findings of García-Mateos and Fernández-Alemán (2009), which posit that current educational tendencies centre on the learners' viewpoint rather than on the educators', with the intention of creating independent, reflective and life-long learners. Due to experience from running and debugging programs, 76.2% of the participants felt that they were equipped to solve similar problems (Question 14.3). A gulf of visualisation arises when programmers have difficulty mapping the observed behaviour of the running program against their mental model (Wright and Cockburn, 2000).

Difficulties in understanding compound logic and the application of algorithmic problem-solving skills: Research conducted by Gomes and Mendes (2007), Esteves *et al.* (2008) and Yu and Yang (2010) affirms that core problems experienced by many novice learners are their lack of problem-solving abilities and the difficulties experienced in using basic concepts, such as control structures, to create algorithms that solve concrete problems. The confidence levels of the participants in the experiment were quite high, in that 76.2% claimed they were able to apply basic problem-solving techniques to create algorithms that solve problems (Question 15.1); that they have a good understanding of pseudocode (81% for Question 15.2); and that they were able to decompose a large and complex programming task into smaller subtasks (76.2% for Question 15.4). Conversely, only 28.6% acknowledged using pseudocode to outline and understand the logic of a program before they started coding and a third (33%) disagreed with using pseudocode to help them understand the logic (Question 15.3). There is a general inconsistency between these responses, and this contradicts the participants' claims that they do not experience difficulty in understanding compound logic.

5.4.2.4 How to improve the teaching of OOP

To address the challenges of learning OOP, techniques to help teach OOP were proposed in the questionnaire, namely:

- (a) Algorithmic thinking and expression;
- (b) Abstraction;
- (c) Objects-first strategy; and
- (d) Three-dimensional animation authoring tools and visualisation.

The participants' responses to the final section on 3D tools and visualisation were strongly influenced by their experiences with *Alice*. These techniques are listed in Table 5.9 with the percentage distributions of participant responses to each.

Table 5.9 Percentage distribution of participant responses for techniques to improve teaching of OOP

How to improve the teaching of object-oriented programming						
16	Algorithmic thinking and expression	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
16.1	I am able to read and write in a formal language.	0.0	4.8	95.2	4.29	16.1
	Category Rating	0.0	4.8	95.2	4.29	
17	Abstraction	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
17.1	I am able to use creative thinking and programming concepts to write programs.	0.0	23.8	76.2	4.05	17.1 17.2 17.3 17.4
17.2	When I write a program, I usually understand every line of code.	33.3	19.0	47.6	3.19	17.1
	Category Rating	16.7	21.4	61.9	3.62	
18	Objects-first strategy	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
18.1	I have a good level of understanding of objects, gained from my first year of study.	19.0	14.3	66.7	3.57	18.1
18.2	It would be easier to learn object-oriented programming during the first year of study, and later learn other control structures such as loops, If statements etc.	33.3	19.0	47.6	3.38	18.2
18.3	<i>Alice</i> helps me to see everything as an object.	0.0	14.3	85.7	4.24	18.3 18.4
	Category Rating	17.5	15.9	66.7	3.73	

19	3D animation authoring tools and visualisation	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
19.1	A visual representation improves my understanding of programming concepts.	0.0	9.5	90.5	4.38	19.1 19.2
19.2	The visual effects in <i>Alice</i> provide a meaningful context for understanding classes, objects, methods, and events.	0.0	19.0	81.0	4.05	19.3 19.6
19.3	Three-dimensionality makes objects seem real.	0.0	4.8	95.2	4.38	19.4
19.4	I am able to use <i>Alice</i> to write a new method to make objects perform animated tasks, such as hop, fly, swim etc.	0.0	4.8	95.2	4.24	19.5
Category Rating		0.0	9.5	90.5	4.26	

Algorithmic thinking and expression: Algorithmic thinking and expression involves the ability to read and write in a formal language (Dann *et al.*, 2009). In the present study, 95.2% of the group agreed that they were able to read and write in a formal language (Question 16.1).

Abstraction: According to Salim *et al.* (2010), many novice programming learners lack the ability to express their creative thinking in terms of programming abstractions, because they do not grasp programming concepts on an abstract level. This may lead to learners writing code without fully understanding every line. Although 76.2% agreed that they were able to use creative thinking and programming concepts to write programs (Question 17.1), a third of the participants in this experiment, acknowledged that on occasions they do write programs without understanding each piece of program code (Question 17.2). With a further 19% of experimental participants being unsure about their ability to understand every line of code when writing a program, it can be safe to assume that at some time or the other, more than half of the participants were unable to grasp programming concepts on an abstract level.

Objects-first strategy: While 66.7% of the experimental participants agreed that they have a good level of understanding of objects, gained from their first year of study, 19% had disagreed (Question 18.1). According to the Joint Task Force on Computing Curricula, IEEE (2001), the objects-first strategy commences by immediately introducing the notions of objects and inheritance and then goes on to introduce more traditional control structures. However, only 47.6% of the participants felt confident that it would be easier to learn OOP during the first year of study, and later learn the conventional control structures such as loops, if statements etc., while a third disagreed (Question 18.2). Phelps *et al.* (2005) state that a way to minimise negative concerns and maximise the positive observations associated with object-oriented and objects-first programming,

is to provide learners with a rich, interactive, collaborative virtual environment that supports the programming experience. A high percentage (85.7%) of the experimental participants stated that *Alice* helps them to view everything as an object (Question 18.3).

3D animation authoring tools and visualisation: As stated previously, participants’ responses to this criterion were greatly influenced by their exposure to *Alice*. Several authors have stated that learners understand programming concepts better when given a visual representation. Three-dimensional animation assists in providing stronger object visualisation and a flexible, meaningful context for helping learners to perceive object-oriented concepts. Three-dimensionality provides a sense of reality for objects (Cooper *et al.*, 2003; Gomes and Mendes, 2007; Sajaniemi *et al.*, 2008; and Carlisle, 2009). A very high percentage (90.5%) of the experimental participants agreed that a visual representation improves their understanding of programming concepts (Question 19.1). Furthermore, 81% agreed that the visual effects in *Alice* provide a meaningful context for understanding classes, objects, methods, and events (Question 19.2). Finally, 95.2% of the experimental participants agree that three-dimensionality makes objects seem real (Question 19.3) and that they were able to use *Alice* to write a new method to make objects perform animated tasks, such as hopping, flying, swimming etc. (Question 19.4).

5.4.2.5 Criteria based on the researcher’s experience

The final part of the questionnaire is based on criteria identified by the researcher. These criteria emerged from her personal 10-year involvement in teaching OOP to IT learners. The percentage distributions of the responses to these criteria are provided in Table 5.10.

Table 5.10 Percentage distribution of participant responses for criteria based on researcher’s experience

The researcher’s experience						
20	Appreciation of trial-and-error	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
20.1	When using <i>Alice</i> , I use trial-and-error to ‘try out’ individual animation instructions as I create new methods.	4.8	38.1	57.1	3.71	20.1
20.2	I can visibly see the effect that each new animation instruction has on the animation.	19.0	19.0	61.9	3.57	20.2
	Category Rating	11.9	28.6	59.5	3.64	

21	Incremental construction approach	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
21.1	<i>Alice</i> has taught me how to program incrementally i.e. I write one method at a time, testing and running each piece.	0.0	20.0	80.0	4.05	21.1
	Category Rating	0.0	20.0	80.0	4.05	
22	Impact of <i>Alice</i> on understanding OOP concepts	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
22.1	Inheritance	0	25	75	3.95	22.1
22.2	Methods	0	25	75	4.00	22.2
22.3	Properties	0	15	85	4.10	22.3
22.4	Functions	5	35	60	3.80	22.4
	Category Rating	1.3	25.0	73.8	3.96	
23	Impact of <i>Alice</i> on understanding basic programming concepts	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
23.1	Loops	4.8	28.6	66.7	4.05	23.1
23.2	If..statements	9.5	33.3	57.1	3.67	23.2
23.3	Data types	4.8	42.9	52.4	3.71	23.3
23.4	Event-driven programming	9.5	42.9	47.6	3.57	23.4
	Category Rating	7.1	36.9	56.0	3.75	
24	Ability to collaborate	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
24.1	The <i>Alice</i> workshop has provided the opportunity to work in pairs with other learners and I have chosen to do so.	9.5	19.0	71.4	3.90	24.1
24.2	I have been able to interact and communicate well with other learners.	0.0	23.8	76.2	4.00	24.1
24.3	The experience of working with other learners has helped me to learn the <i>Alice</i> programming environment.	0.0	28.6	71.4	3.90	24.1
	Category Rating	3.2	23.8	73.0	3.94	

25	Impact of <i>Alice</i> on DS2 learners	D + SD (%)	N (%)	A + SA (%)	AvgRating	Crit#
25.1	The <i>Alice</i> workshop relates directly to the sections on object-oriented programming covered in the Development Software 2 syllabus.	4.8	28.6	66.7	3.90	25.1
25.2	I am interested in learning more about computer graphics and animation.	0.0	0.0	100.0	4.75	25.2
25.3	I am interested in learning and working more with the <i>Alice</i> visual programming environment.	0.0	9.5	90.5	4.38	25.3
25.4	I used <i>Alice</i> during my personal time after attending the first lesson of the <i>Alice</i> workshop.	23.8	14.3	61.9	3.67	25.4
25.5	I am interested in learning more about object-oriented programming.	0.0	4.8	95.2	4.57	25.5
Category Rating		5.8	11.5	82.7	4.25	

During conventional teaching, the researcher observed a prevalence of bricolage (*trial-and-error*) problem-solvers (Stiller, 2009) amongst the DS2 learners. A fairly high number (57.1%) of the experimental participants agreed that they had used trial-and-error to ‘try out’ individual animation instructions, whilst creating new methods, when using *Alice* (Question 20.1), while 61.9% of participants could visibly see the effect that each new animation instruction has on the animation (Question 20.2). A further observation made amongst DS2 learners was their inherent ability to *incrementally construct a program*. A high percent (80%) of participants in the experiment agreed that *Alice* had taught them to program incrementally i.e. writing one method at a time, testing and running each piece (Question 21.1). It is envisaged that learners will be able to visualise and transpose the knowledge of the construction of methods and functions in conventional programs.

Regarding the four core concepts that form a foundation for learning OOP, 75-85% of participants agreed that *Alice* had improved their *understanding of inheritance, methods and properties* (Questions 22.1 to 22.3), while 60% agreed that *Alice* helped with learning functions (Question 22.4). Moreover, an average percentage of 56% agreed that *Alice* helped to improve their *understanding of iteration, selection, data types and event-driven programming* (Questions 23.1 to 23.4). It must be borne in mind that although the experimental participants were being exposed to OOP for the first time during their second year, they had already been exposed to the basic concepts during their first year.

A good percentage (71.4%) agreed that the *Alice* workshop had given them the opportunity to work in pairs with other learners (Question 24.1), and that this *collaboration* had helped them to learn how to use *Alice* (Question 24.3). With regard to teamwork, 76.2% expressed their ability to

interact and communicate well with other learners (Question 24.2). It is re-assuring that *Alice* was able to foster learner collaboration, which forms an integral aspect of the third-year course work.

A good percent (66.7%) of the experimental participants agreed that the *Alice* workshop *relates directly to the sections on OOP* covered in the DS2 syllabus, whilst 28.6% were unsure (Question 25.1). All the participants expressed an interest in learning more about computer *graphics and animation* (Question 25.2), with 90.5% *eager to work more with Alice* (Question 25.3), and 95.2% *wanting to learn more* about OOP (Question 25.5). It was noted that 61.9% had used *Alice* during their *personal time* after attending the first lesson of the *Alice* workshop (Question 25.4).

These findings substantiate the need for further use of *Alice* in teaching and learning OOP.

The next section describes the qualitative data analysis of responses to the open-ended questions in the questionnaire.

5.5 Qualitative data analysis: Open-ended questions

Qualitative data analysis was performed on the open-ended responses to Category 26 of the questionnaire, which contained six open-ended questions (see Appendix B.2). The participants in the experimental group were encouraged to express their perceptions spontaneously, unprompted and unsolicited. They did so in a free-text format, describing their experiences – both positive and negative – with using *Alice*; the challenges they faced in learning OOP; techniques that could improve the teaching of OOP; and the impact of *Alice* on learners' understanding of OOP. They also made suggestions for ways of addressing the challenges they had identified. The component being considered in this section is highlighted in Figure 5.2. Section 5.5.1 explains data collection and preparation, followed by data analysis and interpretation in Section 5.5.2.

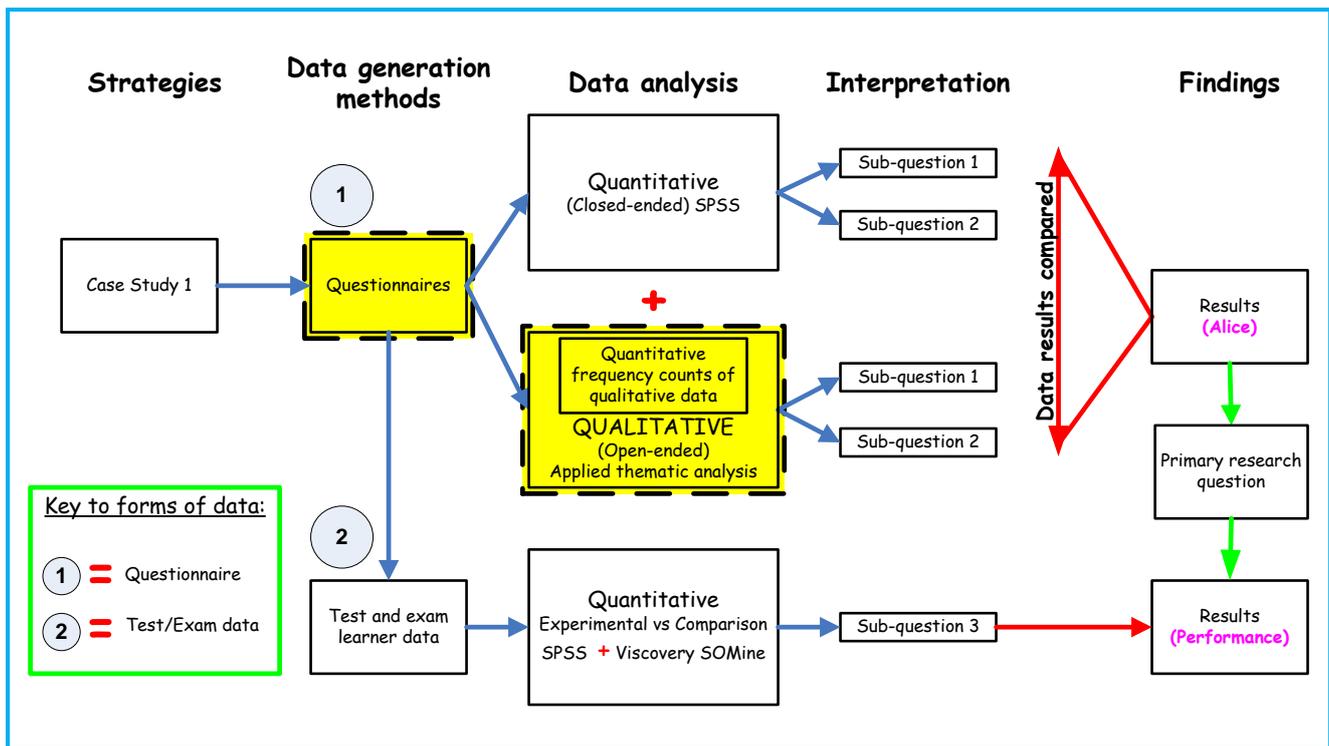


Figure 5.2 Detailed research processes of Case Study 1, highlighting a quantification of qualitative data derived from the open-ended responses to the questionnaire

5.5.1 Data collection and preparation: Open-ended questions

As mentioned in Section 4.8.2, this was implemented using an *applied thematic analysis* (ATA) approach to extract themes from the free text in the open-ended responses. ATA is defined by Guest *et al.* (2012:15) as a “rigorous, yet inductive, set of procedures designed to identify and examine themes from textual data in a way that is transparent and credible”.

Furthermore, a quantification of the qualitative data derived from the open-ended responses provided statistical findings relating to the textual data. Creswell (2009:218) defines this process as *data transformation*, which “involves creating codes and themes qualitatively, then counting the number of times they occur in the text data”. Appendix E.1 depicts the frequency counts of participant responses to the various domains.

No data preparation was required. The researcher studied the participants’ text verbatim.

5.5.2 Data analysis and interpretation: Open-ended questions

Analysis was conducted by manual thematic analysis. On completion of the questionnaires by the statistical sampling, valid and relevant data was transcribed into the codebook, as described in Section 4.8.2.3. The codebook is provided as Appendix E – Qualitative Analysis Tables. Once the

codebook had been updated, substantive corroborative interpretations were extrapolated from it to reach pertinent conclusions.

5.5.2.1 General overview comments

Participants’ personal opinions and general attitudes emerged from their qualitative responses to the open-ended questions, as shown in the first columns of Table 5.11 and Table 5.12. The second columns indicate the number of participants who mentioned that theme. The frequency percentages in the last column of each table convert the number to the percentage of participants who gave responses in line with that theme.

Table 5.11 Positive overview comments on the usability of *Alice*

Positive overview comments	Frequency counts (n=21)	%
I like <i>Alice</i>	21	100
<i>Alice</i> is a fun/exciting/enjoyable/interesting way to learn programming	8	38
Suitable for novice programmers, e.g. high school learners or first-year learners starting OOP	1	5
Promotes learner collaboration	1	5
Promotes self-learning; anticipated future personal use	2	10
Total	33	-

All participants in the experiment expressed a liking for *Alice*. A pattern observed from 38% revealed that *Alice* is a fun, engaging tool that assists in learning programming. One participant (5%) suggested that *Alice* can be optimally used by novice programmers, whilst another participant stated that it can be used for collaborative learning. Dickey (2003) believes that 3D virtual worlds provide means of creating rich and compelling contexts to support learning and collaboration (see Section 2.4.4). A theme that emerged from two participants (10%) indicated that using *Alice* had improved their motivation to self-learn and had developed their interest in further use of the VPE in their personal capacity. Hadjerrouit (2008) affirms that learners are able to learn better when they explore and discover things themselves, which is known as *constructivist* learning (see Section 2.2.4).

Table 5.12 Negative overview comments on the usability of *Alice*

Negative overview comments	Frequency counts (n=21)	%
Time consuming – takes practice getting accustomed to <i>Alice</i> at first exposure	2	10
The challenges of <i>Alice</i> are that it requires active thinking, creativity and prior understanding of OOP concepts	1	5
Total	3	-

Two (10%) participants indicated that it was time-consuming to develop the initial skills necessary to use *Alice*. Another (5%) stated that the challenge of using *Alice* was the capacity to think critically and apply one’s creative abilities. This participant further believed that it required a prior understanding of OOP concepts. This relates to the *cognitive* perspective of learning described by Hadjerrouit (2008), as a psychology that focuses on mental activities, such as analytical reasoning and critical thinking. The cognitive paradigm emphasises understanding of concepts and their relationships (see Section 2.2.3). However, it is notable that these negative remarks came from only three of the 21 participants in Case Study 1.

5.5.2.2 Spontaneous positive responses on usability of *Alice*

Nielsen’s (1994) ten heuristics, as mentioned in Section 4.5.2.1, formed the foundation for the first ten items in the questionnaire, thus allowing the usability of the *Alice* VPE to be evaluated with respect to this set of classic usability principles. Furthermore, these heuristics formed a basis for the researcher to use in categorising the qualitative responses to questions on the usability of *Alice*. The criterion number in the first column of Table 5.13 is cross-referenced to Table 4.3 for the purposes of validity and completeness.

Table 5.13 relates to positive learning experiences pertaining to the usability of *Alice* and categorised into sub-themes. Eight of the ten Nielsen’s heuristic characteristics were evident in the open-ended responses.

Table 5.13 Spontaneous positive responses on usability of *Alice*

#	Nielsen's Heuristics	The heuristic as it applies to <i>Alice</i>	Frequency counts (n=21)	%
1	Visibility of the system status	Immediate feedback after program execution	1	5
		<i>Alice</i> is interactive	3	14
2	Match between the system and the real world	Graphics and animation	5	24
		Dealing with real objects brings coding into reality	1	5
		Visualisation, e.g. see creation of methods, see objects move on command	4	19
		3-Dimensionality: visual effects reflect real-world scenarios	1	5
3	User control and freedom	Learning how to create movies/storytelling	2	10
		Learning how to develop video games	2	10
		More control over my programs	1	5
6	Recognition rather than recall	Drag-and-drop feature limits typing	4	19
		Releases learner to focus on problem-solving	2	10
		No complex syntax, only English-like statements	1	5
7	Flexibility and efficiency of use	User-defined methods are created to manipulate objects and can be tested individually	1	5
		Objects are available from the local gallery or can be downloaded from the Internet	1	5
8	Aesthetic and minimalist design	<i>Alice</i> interface is simple and easy to use	2	10
9	Recognition, diagnosis and recovery from errors	Can recover from errors quickly and easily	2	10
10	Help and documentation	Built-in tutorials assist with learning the basic techniques of programming; help teach how to use <i>Alice</i> ; and provide an overview of <i>Alice</i>	1	5
Total			34	-

In relation to *visibility of the system status*, one experimental participant indicated that the visual feedback improved his understanding of programming concepts, corresponding with the findings of Gomes and Mendes (2007), Sajaniemi *et al.* (2008), Carlisle (2009), and Montero *et al.* (2010), all of whom consider visualisation to be a mechanism that promotes the understanding of programming concepts. Regarding the *match between the system and the real world*, a pattern emerged in that 24% of the participants mentioned the enjoyment of working with graphics and animation. Moreover, 19% were satisfied with the ability to see results as objects moved on command from the methods they created. One participant indicated a preference of having control over the program. Others (10%) enjoyed the freedom to develop their own video games and movies, pertaining to the heuristic *user control and freedom*. In relation to *recognition rather than recall*, 19% appreciated not having to concern themselves with tedious typing, whilst one participant (5%) expressed satisfaction with not having to be concerned about syntax and semantics. One participant appreciated the ability to create user-defined methods to manipulate objects, thus contributing to the *flexibility and efficiency of using Alice*. Regarding *aesthetic and minimalist design*, 10% mentioned that the *Alice* interface is simple and easy to use. Two participants (10%) indicated that they were able to *recover from errors* quickly and easily and one mentioned that the tutorials in *Alice* were helpful for learning the basic techniques of programming, thus contributing to the heuristic *help and documentation*.

5.5.2.3 Spontaneous negative responses on usability of *Alice*

The criterion number in the first column of Table 5.14 is cross-referenced to Table 4.3 for the purposes of validity and completeness. The responses in Table 5.14 are the negative perceptions regarding the usability of *Alice*. Once again, they are categorised against related heuristics of Nielsen (1994) and further categorised into sub-themes.

Table 5.14 Spontaneous negative responses on usability of *Alice*

#	Nielsen's Heuristics	The heuristic as it applies to <i>Alice</i>	Frequency counts (n=21)	%
2	Match between the system and the real world	Poor quality of graphics	1	5
4	Consistency and standards	<i>Alice</i> takes too long to load	1	5

6	Recognition rather than recall	Difficulty with the use of Math functions, e.g. avoiding the collision of objects	1	5
		Insufficient built-in methods to manipulate objects	2	10
		Difficulty with the use of methods/functions	1	5
7	Flexibility and efficiency of use	Time consuming – use of the camera to turn the point of view/zoom to a particular object, e.g. set to a new scene; following an object such as a flying bird	1	5
		Have to consider every object	1	5
		Time consuming – movement of objects, e.g. moving single limbs of bodies, so as to propel the object across the screen	2	10
		Time consuming – advanced functions are difficult to find and use	1	5
8	Aesthetic and minimalist design	User interface cannot be customised, e.g. restrictive world templates and objects; non-adjustable windows	2	10
		I like coding and <i>Alice</i> moves away from this	1	5
		Structured interface doesn't provide alternate ways to access a method	1	5
		Ambiguous method names	1	5
9	Recognition, diagnosis and recovery from errors	<i>Alice</i> stalls/crashes intermittently	2	10
		Time consuming - no recovery from errors, e.g. 'Console error' upon deletion of code; re-adding objects that 'disappeared'	1	5
10	Help and documentation	Insufficient literature and tutorials available on <i>Alice</i>	1	5
Total			20	-

Seven of the ten basic interface design heuristics were evident in these negative responses. When analysing negative experiences pertaining to the usability of *Alice*, listed in Table 5.14, a pattern was observed in that only five to ten percent of the experimental participant responses were negative for each sub-theme that emerged.

5.5.2.4 Challenges faced by learners in learning OOP

The unprompted, unsolicited feedback of the experimental participants regarding the challenges that they are faced with when learning OOP, is shown in Table 5.15. This question related to learning OOP in general, not to learning OOP via *Alice*.

Table 5.15 Challenges faced by learners in learning OOP

Challenges of learning OOP	Frequency counts (n=21)	%
Inheritance	1	5
Methods (calling/overriding/abstract/virtual/no default/event-driven)	4	19
Parameter passing	1	5
Creating classes and sub-classes (abstract)	2	10
Creating objects and instantiating objects	1	5
Difficulty with problem-solving and applying concepts, e.g. when to write code for objects and methods	6	29
Logically connecting theoretical concepts of OOP with practical examples	1	5
Lack of visual representation to aid with understanding logic	1	5
Lack of motivation, for reasons such as boredom, intimidation or frustration	1	5
Lack of resources to facilitate learning, such as time, money, computers, open labs	2	10
Having to remember syntax	5	24
Total	25	-

Similar findings emanated from the quantitative and qualitative studies, where in response to the closed-ended Question 15.1 in Table 5.8, almost 24% of experimental participants were unsure about their ability to apply basic problem-solving techniques to create algorithms that solve problems. Correspondingly, in the qualitative unprompted responses to Question 26.4 in Table 5.15, 29% of the experimental group expressed difficulty with problem-solving and the application of programming concepts.

Further qualitative findings emerged, where 24% stated that having to remember syntax was a challenge. Nineteen percent of the experimental participants had experienced difficulty in understanding the logic of methods. Apart from these more common challenges, others indicated challenges that were general in nature or that related directly to features of OOP, such as inheritance, objects, classes and so on.

5.5.2.5 Techniques to improve the teaching of OOP

In order to address the challenges identified in Section 5.5.2.4, the participants were required to suggest techniques that would help alleviate these issues.

Table 5.16 Techniques to improve teaching of OOP

Techniques to improve teaching of OOP	Frequency counts (n=21)	%
Detailed, interactive explanations of programs with practical examples that enforce the understanding of theoretical concepts, instead of just giving solutions and/or notes	6	29
Pace of lecturing could be less rapid, i.e. avoid moving ahead with new concepts before learners have had enough time to grasp other concepts	2	10
Using visual, graphical, interactive environments as part of standard teaching, to improve learner interest and motivation to learn OOP	3	14
Introduce <i>Alice</i> as a supplementary teaching tool	2	10
Providing solutions to examples	2	10
More examples during lecture time	2	10
Student support systems and resources for IT learners, such as more open lab time, additional tutorials with examples, references and tutors to assist learners one-on-one	3	14
Introducing OOP earlier in the ND: IT programme	2	10
Total	22	-

Twenty-nine percent of the experimental participants suggested that lecturers should provide explanations of programs, supplemented with practical examples that enforce the understanding of theoretical concepts. Participants were not favourable to merely being given solutions to problems without discussions thereof. Three participants (14% of the participants) suggested that standard use of a visual, graphical environment, such as *Alice*, would improve learners' interest and motivation to learn OOP. A further 14% requested a wider student support system and resources to facilitate the learning process.

5.5.2.6 Impact of *Alice* on improving understanding of OOP

Table 5.17 presents the spontaneous responses of the experimental participants regarding the impact of *Alice* on improving their personal understanding of OOP. As indicated by the three pink shaded rows in Table 5.17, there were defined differences in responses to this question. For instance, most participants (76%) firmly agreed that working with *Alice* had improved their understanding of OOP. Conversely, some participants (10%) did not agree that *Alice* had a positive impact on improving

their understanding of OOP, whilst another 10% felt that *Alice* had helped them only to a certain extent. Discussion follows according to these three definitive types of responses, giving the sub-themes in each category.

Table 5.17 Impact of *Alice* on improving understanding of OOP

Impact of <i>Alice</i> on improving understanding of OOP	Frequency counts (n=21)	%
Yes, working with <i>Alice</i> has improved my understanding of OOP	16	76
I have an improved understanding of OOP concepts such as methods, functions, events, inheritance, properties, parameters, classes, objects, instantiation and polymorphism	11	52
<i>Alice</i> helped me understand OOP visually	6	29
I can create methods to manipulate and animate objects to perform actions	4	19
Everything in <i>Alice</i> is viewed as an object	1	5
I have a clearer understanding that objects have different parts. It is easier to work with parts of an object	1	5
Reality of objects in <i>Alice</i> helps to understand objects in conventional languages	1	5
This environment improved my understanding of basic concepts such as loops and selection statements	2	10
Working with <i>Alice</i> has improved my understanding of OOP only to a certain extent	2	10
<i>Alice</i> provided extra means of revision to reinforce concepts on OOP, in a fun way	1	5
No, working with <i>Alice</i> has not improved my understanding of OOP	2	10
I already have a sound understanding of OOP	1	5
The theory of OOP still eludes me	1	5
Total	49	-

In response to the effects of learning with *Alice*, a theme that emerged consistently (76% of participants) was acknowledgement that their understanding of OOP had improved. This qualitative finding helps to confirm the quantitative finding of the closed-ended Question 19.2 in Table 5.9, where 81% of the experimental participants agreed that the visual effects in *Alice* provide meaningful contexts for understanding OOP concepts such as classes, objects, methods, and events. Further qualitative findings emerged, where 52% indicated that *Alice* enhanced their grasp of specific concepts within the OOP domain, such as methods, functions, events, inheritance, properties, parameters, classes, objects and polymorphism. Twenty-nine percent suggested that such improvement can be attributed to the visual nature of *Alice*.

Two participants (10%) believed that *Alice* had partially improved their understanding of OOP. Their interaction with the VPE had proven to be a suitable means of revision on OOP concepts. On the other hand, one expressed having a sound grasp of OOP concepts prior to using *Alice*, thereby indicating that *Alice* had not served to improve his understanding. Conversely, another felt intimidated by the theoretical concepts of OOP and indicated that the *Alice* intervention did not dispel these uncertainties.

5.5.2.7 Impact of *Alice* in addressing challenges of OOP

Feedback from the experimental participants concerning the impact of *Alice* in addressing the challenges of OOP is presented in Table 5.18. Many of the participants (71%) spontaneously indicated that *Alice* as a VPE can help address some of these challenges, whilst only one (5%) explicitly stated that *Alice* did not help. Discussion follows according to these two definitive groups of responses, giving the sub-themes in each category.

Table 5.18 Impact of *Alice* in addressing challenges of OOP

Impact of <i>Alice</i> in addressing challenges of OOP	Frequency counts (n=21)	%
Yes, I agree that <i>Alice</i> as a VPE can help address some of these challenges	15	71
It is easier to learn programming through visualisation/graphics, than having to remember the syntax for coding	3	14
Ability to see the visual effects of every statement of code	4	19
Use of trial-and-error to alter the desired output	1	5
Drag-and-drop feature releases the learner from having to write code and complex syntax	3	14
<i>Alice</i> is fun, engaging and cultivates an interest in programming	4	19
A visual representation of how objects interact with each other is valuable	2	10
It makes programming concepts easy to learn and understand	6	29
No, I do not agree that <i>Alice</i> helps address these challenges	1	5
Drag-and-drop feature does not assist with the hard-coding required in programming languages	1	5
Total	40	-

A clear pattern emerged in that 71% felt that *Alice's* VPE helped them address some of the challenges they had faced in learning OOP. Conversely, only one participant (5%), doubted that *Alice* helped to address these challenges. Six participants (29%) indicated that *Alice* cultivates an easy approach to learning and understanding programming concepts. Nineteen percent appreciated

the ability to see the effects of every statement of code, i.e. immediate feedback, whilst 14% felt that programming through visualisation alleviates the learner from having to remember coding syntax. Similarly, three participants (14%) stated that the drag-and-drop features of *Alice* release the learner from complex syntax and semantics. Conversely, one participant felt that the drag-and-drop feature does not equip the learner with the necessary hard-coding skills that are required in conventional programming.

5.6 Quantitative data analysis: Test and exam learner data

This final section on analysing the data of Case Study 1, presents the quantitative analysis of test marks and exam results, comparing performances of the experimental group and the comparison group. The component being addressed in this section is highlighted in Figure 5.3.

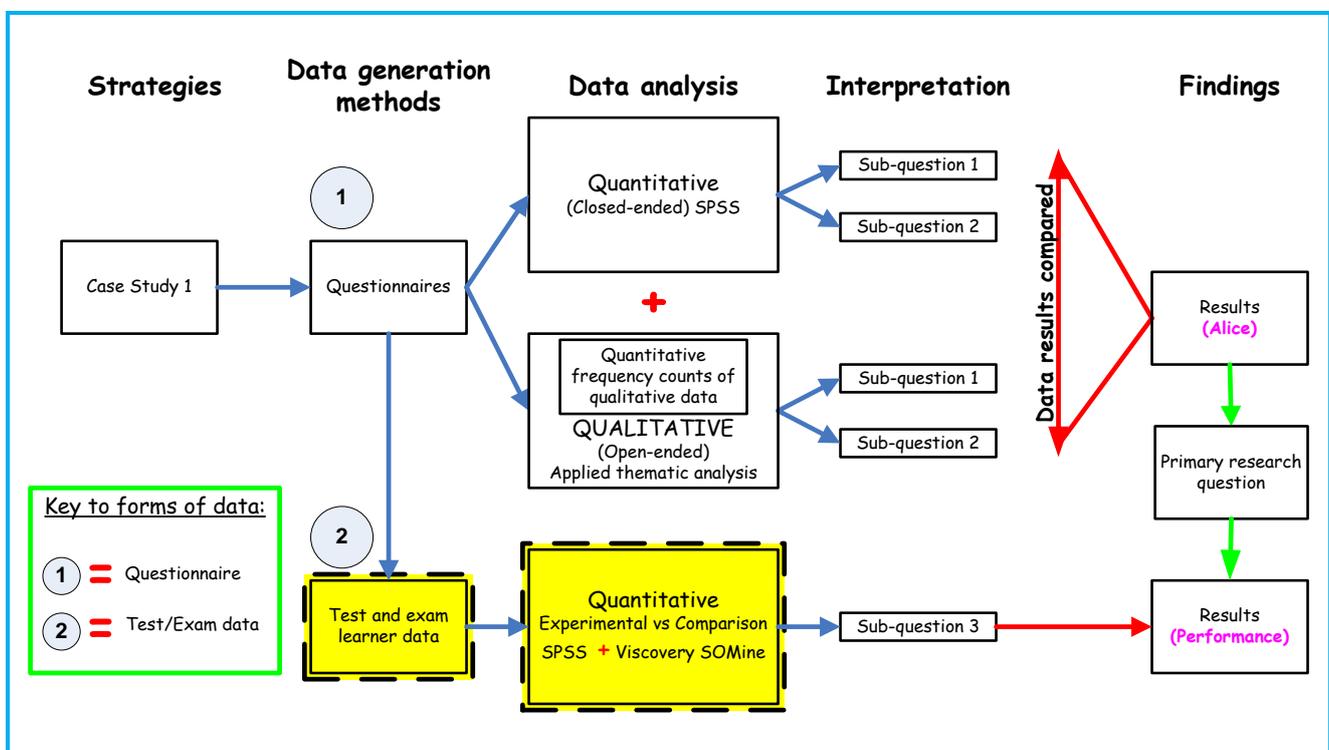


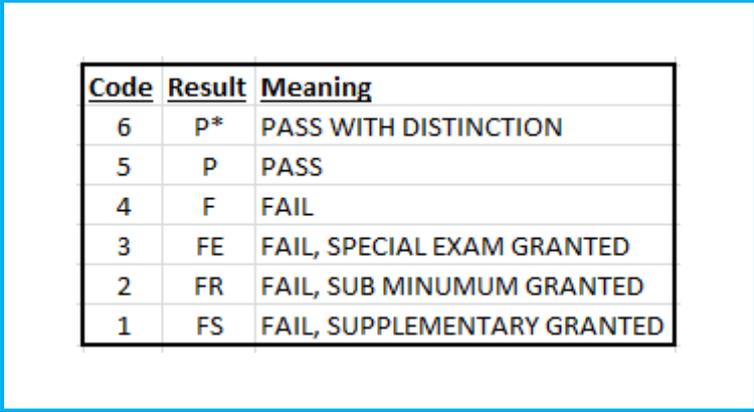
Figure 5.3 Detailed research processes of Case Study 1, highlighting quantitative data analysis of learner data from the tests and exam

5.6.1 Data collection and preparation: Test and exam learner data

The second quantitative data set formulated during Case Study 1 was based on academic performance data from the experimental and comparison groups, after the former group's exposure to learning OOP via *Alice*. The 2011 test and exam learner data for both groups was extracted from the Integrated Tertiary Software (ITS), which is an internal database at DUT and was used with permission from DUT. This database holds all records of learners registered at DUT and contains

their personal details and subject results. Authorisation from DUT to conduct this research is provided in Appendix A.1.

The raw data extracted from the ITS database was originally in text format (.txt files). In this part of the case study, the original data had to undergo data preparation in order to ensure that the resulting formatted data sets were cleansed from any data impurities. The data was then imported into a Microsoft Excel spreadsheet that is compatible with SPSS as well as Viscovery SOMine. Considering that both these statistical packages require the data to be in a numerical format, gender and race were converted to numeric formats using the coding pattern shown in the key for Table 5.19 and Table 5.20. Similarly, the subject result symbols (P*, P, F, FE, FR, FS) were converted to numeric equivalents according to the coding pattern in Figure 5.4, and which appear as the final column in Table 5.19 and Table 5.20.



Code	Result	Meaning
6	P*	PASS WITH DISTINCTION
5	P	PASS
4	F	FAIL
3	FE	FAIL, SPECIAL EXAM GRANTED
2	FR	FAIL, SUB MINIMUM GRANTED
1	FS	FAIL, SUPPLEMENTARY GRANTED

Figure 5.4 Codes for results, listed against the associated description

The complete data sets for the experimental group and the comparison group are shown in Table 5.19 and Table 5.20 respectively. It must be noted that these final data sets are entirely numeric and exclude the student number from the 'Participant' column, instead giving only the participant number as used in the present research. The 'DS101' and 'DS102' columns give the participants' first-year results for Development Software 1, Module 1 and Module 2 respectively, which served as prerequisites for DS2. The 'Test 1', 'Test 2', 'Exam Mark' and 'Final Mark' columns are performance assessments for Development Software 2. The 'Result' column provides each participant's final result, of which the meaning is given in Figure 5.4. The average marks appear at the bottom of the data sets.

Table 5.19 Data set 1 - Experimental group

Participant	DEMOGRAPHIC DATA		FIRST-YEAR MARKS		SECOND-YEAR MARKS				
	Gender	Race	DS101	DS102	Test1	Test2	Exam Mark	Final Mark	Result
P1	1	2	66	75	44	74	22	38	3
P2	0	3	68	57	48	59	46	50	5
P3	1	4	52	62	60	83	54	59	5
P4	0	4	52	72	69	71	60	62	5
P5	1	4	68	73	90	32	28	42	3
P6	0	3	75	77	80	89	79	80	6
P7	1	4	64	70	54	38	30	40	4
P8	1	4	55	69	50	82	88	78	6
P9	1	4	59	72	75	72	82	76	6
P10	1	4	61	71	42	75	51	57	5
P11	1	4	52	50	77	79	65	67	5
P12	1	2	75	89	89	100	63	70	5
P13	0	4	66	75	71	69	57	62	5
P14	1	3	56	62	59	86	59	64	5
P15	1	4	75	82	51	80	66	67	5
P16	1	3	67	50	43	17	39	40	4
P17	1	3	59	57	65	78	28	46	2
P18	1	4	75	79	71	76	50	59	5
P19	1	3	71	85	84	72	86	80	6
P20	1	1	63	58	33	63	21	35	4
P21	1	4	53	69	67	74	70	71	5
AVERAGES			63.43	69.24	62.95	69.95	54.48	59.19	4.71

Key:

Gender: M = 1, F = 0

Race: Black = 4, Asian = 3, Coloured = 2, White = 1

Result: Pass with distinction = 6, Pass = 5, Fail = 4, Fail, special exam granted = 3, Fail, sub minimum granted = 2, Fail, supplementary granted = 1

As mentioned in Section 4.4, in Case Study 1 the experimental learners chosen to participate (P1...P21 in Table 5.19) were selected from those who had passed all four first-year subjects at first attempt. The selection of learners for the comparison group (Q1...Q21 in Table 5.20), had a similar success rate at first-year level, and were hand-picked from the remaining learners. The selection criteria differed slightly for Case Study 2, as will be explained in Section 6.7.1.

Table 5.20 Data set 2 - Comparison group

Participant	DEMOGRAPHIC DATA		FIRST-YEAR MARKS		SECOND-YEAR MARKS				
	Gender	Race	DS101	DS102	Test1	Test2	Exam Mark	Final Mark	Result
Q1	1	4	65	60	72	58	86	76	6
Q2	1	3	72	56	57	69	33	45	2
Q3	1	3	72	76	67	84	48	58	5
Q4	1	4	50	56	71	43	23	36	4
Q5	0	4	71	63	61	43	53	51	5
Q6	0	4	61	55	54	24	29	33	4
Q7	0	4	50	52	56	55	42	50	5
Q8	1	3	69	84	67	80	45	55	5
Q9	0	4	60	60	58	86	63	66	5
Q10	1	4	56	62	71	73	69	71	5
Q11	1	4	65	75	59	50	44	51	5
Q12	1	4	72	78	74	71	55	63	5
Q13	1	3	77	69	65	69	40	51	5
Q14	1	4	58	59	51	31	47	46	1
Q15	1	3	73	72	72	18	41	46	1
Q16	1	4	53	71	78	69	57	64	5
Q17	1	3	63	69	80	86	56	63	5
Q18	1	4	58	56	56	87	64	67	5
Q19	1	3	59	67	73	59	57	62	5
Q20	1	4	55	60	58	52	26	38	4
Q21	1	4	65	69	60	52	57	59	5
AVERAGES			63.05	65.19	64.76	59.95	49.29	54.81	4.38

Key:

Gender: M = 1, F = 0

Race: Black = 4, Asian = 3, Coloured = 2, White = 1

Result: Pass with distinction = 6, Pass = 5, Fail = 4, Fail, special exam granted = 3, Fail, sub minimum granted = 2, Fail, supplementary granted = 1

The test and exam data in Table 5.19 and Table 5.20 will be used to compare the performance of the learners from the experimental group with those from the comparison group. This is addressed in the next section.

5.6.2 Data analysis and interpretation: Test and exam learner data

5.6.2.1 SPSS data analysis of test and exam learner data

As mentioned in Section 4.8.1.1, SPSS is a comprehensive and flexible statistical analysis tool used to generate tabulated reports, charts, plots of distributions and trends in this study. One of the forms of analysis available in SPSS is inferential statistical analysis, which is concerned with the testing of hypotheses and allows the researcher to draw conclusions about populations from sample data (Lind *et al.*, 2002). Inferential statistical analysis was applied to the test and exam marks to compare the performance of the learners from the experimental group with those from the comparison group. The independent t-test was used to compare the mean values between the two groups. The results are presented in Table 5.21, listed according to sequence in time of assessments, with the intervention of *Alice* indicated by the red border, prior to Test 1.

Table 5.21 t-test for equality of means, mean scores and standard deviation for the experimental group and the comparison group

Assessment	Sig. (2-tailed)	Group	N	Mean	Std. Deviation	Std. Error Mean
DS101	.879	Comparison	21	63.0476	8.00298	1.74639
		Experimental	21	63.4286	8.16438	1.78161
DS102	.189	Comparison	21	65.1905	8.72708	1.90440
		Experimental	21	69.2381	10.78844	2.35423
Test1	.656	Comparison	21	64.7619	8.46702	1.84766
		Experimental	21	62.9524	16.43921	3.58733
Test2	.113	Comparison	21	59.9524	20.40460	4.45265
		Experimental	21	69.9524	19.52556	4.26083
Exam Mark	.360	Comparison	21	49.2857	15.13652	3.30306
		Experimental	21	54.4762	20.74999	4.52802
Final Mark	.288	Comparison	21	54.8095	11.64740	2.54167
		Experimental	21	59.1905	14.57607	3.18076
Result	.380	Comparison	21	4.3810	1.35927	.29662
		Experimental	21	4.7143	1.05560	.23035

The traditional approach to reporting a result requires a statement of statistical significance. A p-value is generated from a test statistic. A significant result is indicated with " $p < 0.05$ " (Lind *et al.*, 2002:347). It can be noted from the 'Sig. (2-tailed)' column in Table 5.21, that all of the 2-tail significance values are greater than 0.05, which thus implies that there is no significant difference between the mean scores per variable. For example, in the case of DS102, the p-value for DS102 is

greater than 0.05 ($p=.189$), implying that there is no significant difference between the mean values of the two groups.

However, the mean scores for three of the four assessments conducted after the *Alice* intervention were higher for the experimental group, indicating a positive difference, even if not a significant difference. This is indicated in the ‘Mean’ column for the assessments conducted after the learner’s exposure to *Alice*. For example, Table 5.21 indicates that both the examination performance and the final mark were approximately 5% higher in the experimental group, with a 10% higher difference for Test 2. This is indicated by bold red font in Table 5.21, and graphically depicted in Figure 5.5.

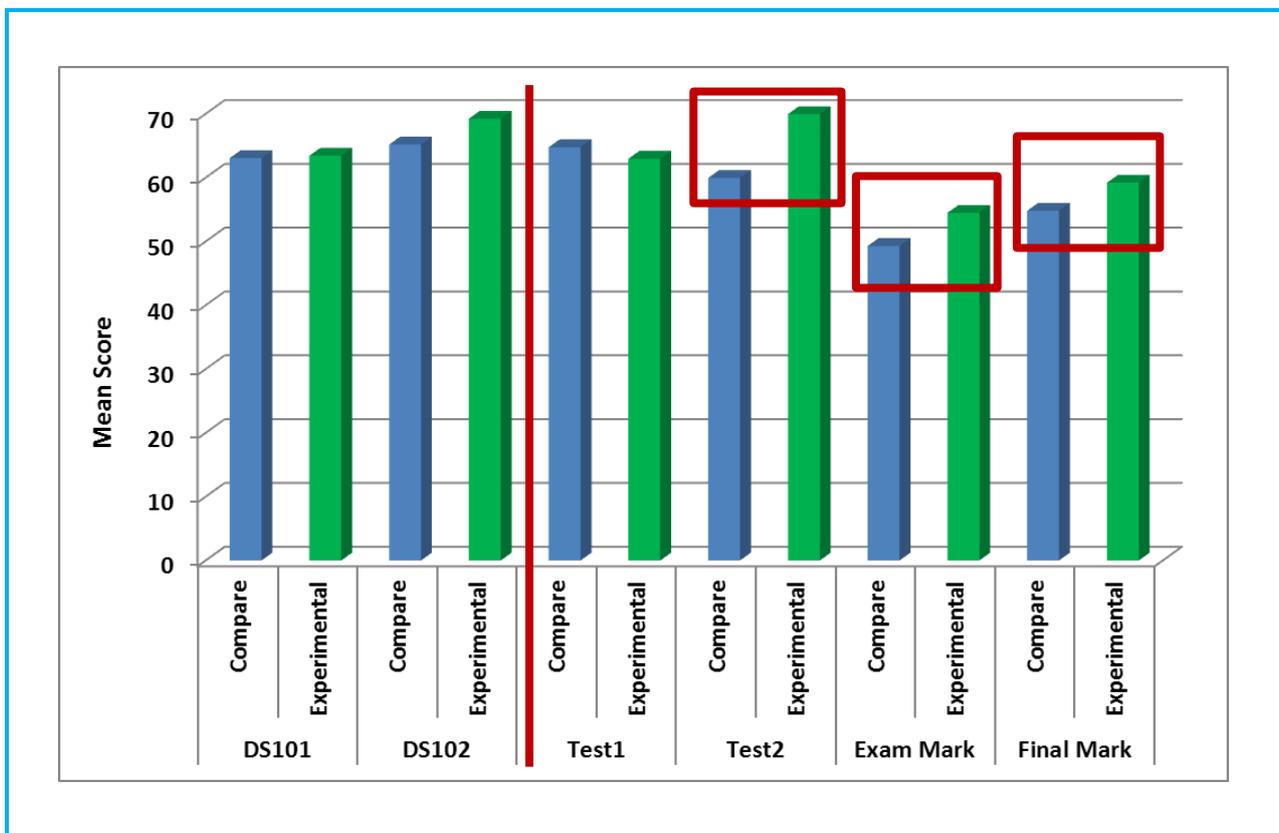


Figure 5.5 Bar chart representing the mean score comparison between the experimental and comparison groups

According to Clark (1994), teaching methods delivered by different media or by various combinations of media, all produce similar learning results. Clark’s claim appears to hold relevance in the present study. Similarly, Owusu, Monney, Appiah and Wilmot (2010) conducted a study with cohorts of learners from two randomly selected schools, exposing one group to Computer-Assisted Instruction (CAI) and the other to conventional teaching methods. Results revealed that the CAI learners did not perform better than the conventional group, hence the study concluded that the use of CAI is not superior to the conventional approach. However, the learners in the CAI group found

their e-learning exposure to be interesting and engaging. Owusu *et al.* established that media in the form of CAI did not improve performance over performances attained by conventional teaching methods, but nevertheless served as a catalyst to promote learner motivation amongst low achievers. In the present study, responses to Questions 25.2, 25.3 and 25.5 (Table 5.10) indicated that *Alice* had notably increased motivation amongst DS2 learners.

5.6.2.2 Viscovery SOMine data mining analysis of test and exam learner data

As mentioned in Section 4.8.1.2, Viscovery SOMine is a high-potential data mining tool used in this study for the visual analysis and exploration of numerical data sets. *Knowledge acquisition* is the third phase of data mining. “During this phase, the data mining tool (with possible intervention by the end user) selects the appropriate modeling or knowledge acquisition algorithms” (Rob and Coronel, 1997:731). The result generated is usually presented in the form of a computer model that reflects the behaviour of the target data set. This process of acquiring knowledge will now be addressed, by observing cluster formations and by analysing cluster maps and component pictures, which were introduced in Section 4.8.1.2. The cluster maps can be used optimally when viewed on a computer screen and also by overlaying of component pictures.

Note: Refer to the DVD included in Appendix F.1 for screen shots of the cluster maps and component pictures for Case Study 1 and Case Study 2.

Cluster Maps

When the data sets shown in Table 5.19 and Table 5.20 were processed by Viscovery SOMine, a ‘display’, also known as a *cluster map* was generated. The cluster map displays coloured map regions, known as *clusters*, as shown in Figure 5.6 and Figure 5.7. In this case, Viscovery SOMine classified the data into 20 clusters, each indicated visually in a different colour.

A cluster may be seen as a map area containing similar vectors or entities. The clusters divide the input data into disjoint classes; this partitioning is called clustering. Each cluster is made up of hexagonal units, called *nodes*. Each node represents a part of the source data set, which in this case represents learner data. Nodes are distinguished on the edges of the cluster and cannot be visibly noticed within the central part of the cluster. The relationship between a cluster map, a cluster and a node is depicted in Figure 5.6. A *separator* is the distinctive black line that is drawn between two nodes if they are in different clusters, as shown in Figure 5.7 (Eudaptics Software, 2001).

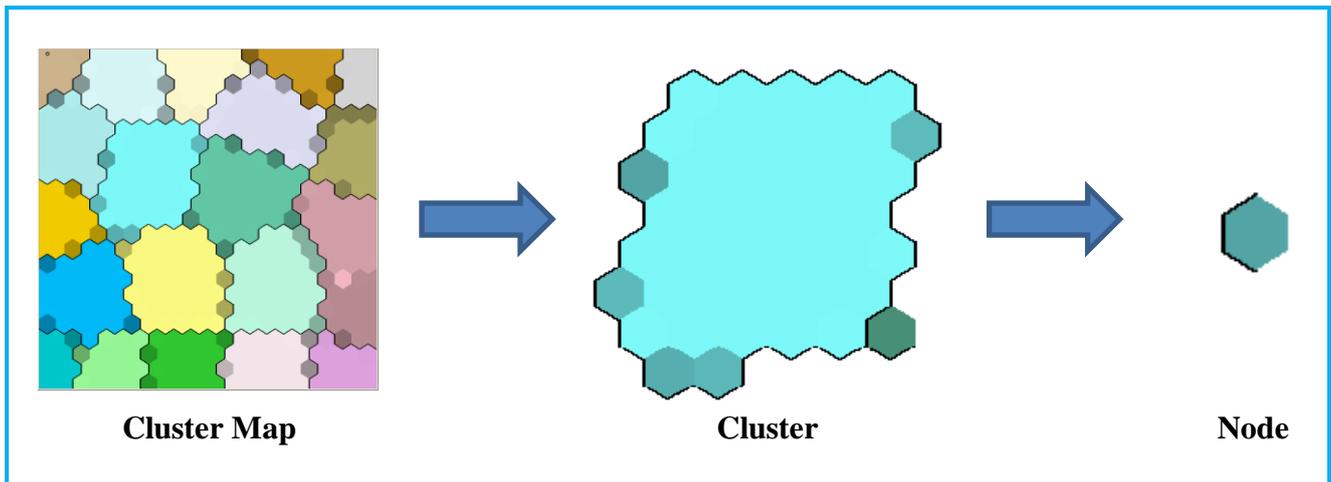


Figure 5.6 The relationship between a cluster map, a cluster and a node

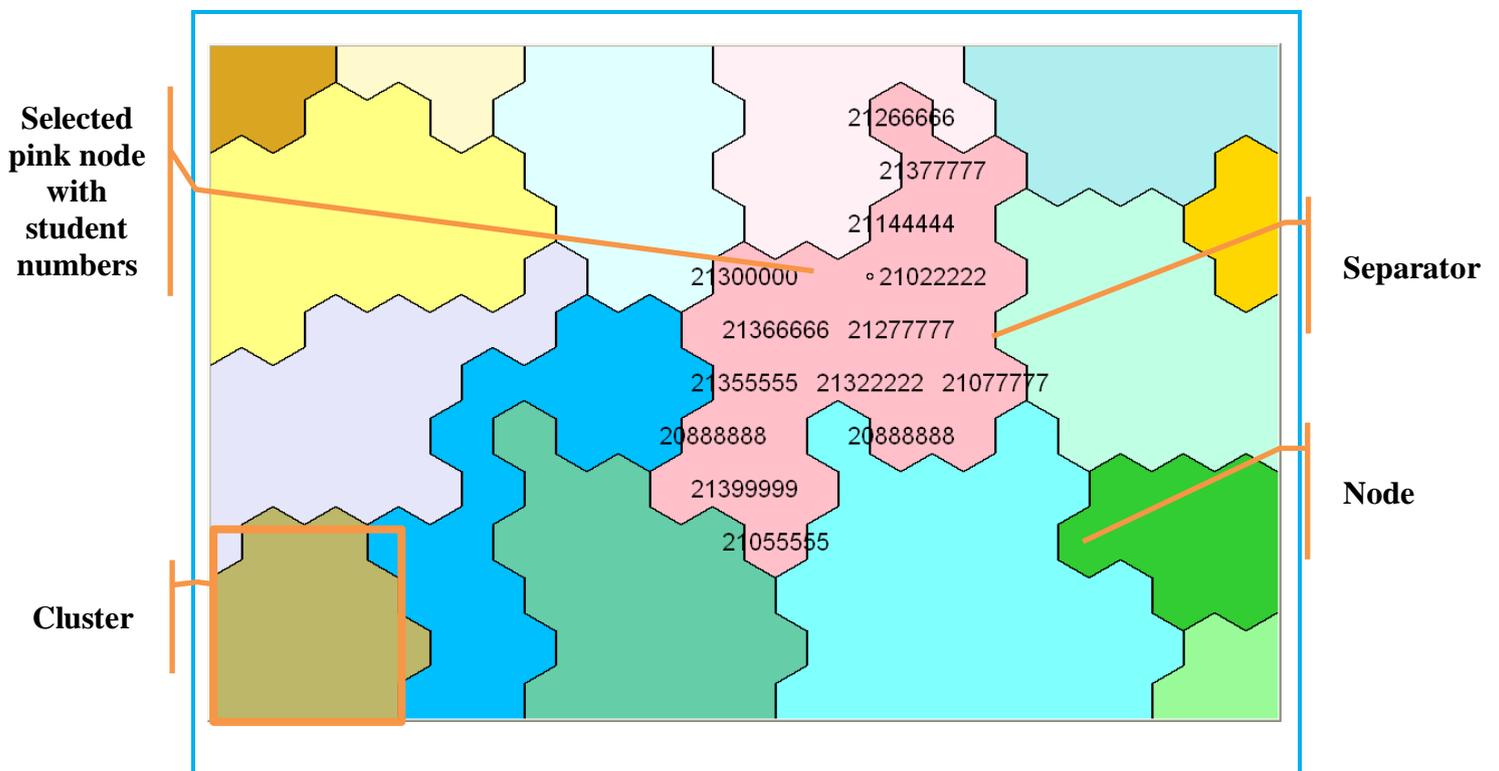


Figure 5.7 Cluster map indicating student numbers within a selected node

(The student numbers in the pink cluster are fictitious and bear no relation to any learners)

Each node in this study represents a subgroup of similar learners who obtained marks within a certain range. This can be observed in the case of the pink cluster of Figure 5.7. When a particular node within the cluster is selected on screen, the node's values can be viewed. The nodes can be labeled according to any attribute of the data set and, in the case of Figure 5.7, the attribute is the set of student numbers of the learners belonging to the selected node. This information proves useful in identifying high performers as well as at-risk learners based on their test results. For example,

learners whose results fall within the range of 40-49% in Test 1 are potential at-risk learners, who would benefit from further tutorial classes to improve their chances of passing the subject.

Component Pictures

In addition to the cluster map, Viscovery SOMine generates component pictures, as shown in Figure 5.8. A component picture is generated for each component from the data set, where a component refers to a column in the data file. This is done so that the non-linear dependencies and relationships between components can be extracted. Component pictures represent the local average component value at each node in a certain colour. The colour scale below the picture shows the relationship between colours and component values. By default, the scale at the bottom of each component picture ranges from the minimum node value to the maximum node value of the respective component (Eudaptics Software, 2001).

One can see how the learning process used by Viscovery SOMine has grouped the nodes in the map with respect to the different components. For example, with respect to the 'Gender' component picture situated centrally in Figure 5.8, one can see nodes that represent the male learners in red and those representing the female learners coloured blue. This corresponds to the male:female ratio, for which the numerical equivalent is 17:4, as shown in Table 5.4. Similarly, the red nodes in the 'Race' component picture on the right represent black learners and those representing the white learners are dark blue, and so on.

Furthermore, the component pictures are directly related to each other. Therefore, overlaying the component pictures will allow one to visually judge, for example, the correspondence between the 'Student Number' component picture and all other components. There is a strong correlation between the red nodes in the 'Race' component picture and the red nodes in the 'Gender' component picture, indicated by their intersection on the overlays on which these nodes are distributed. This observation affirms the statistics presented in Table 5.4, which indicated that the highest number of learners in a gender-race category was the set of black males.

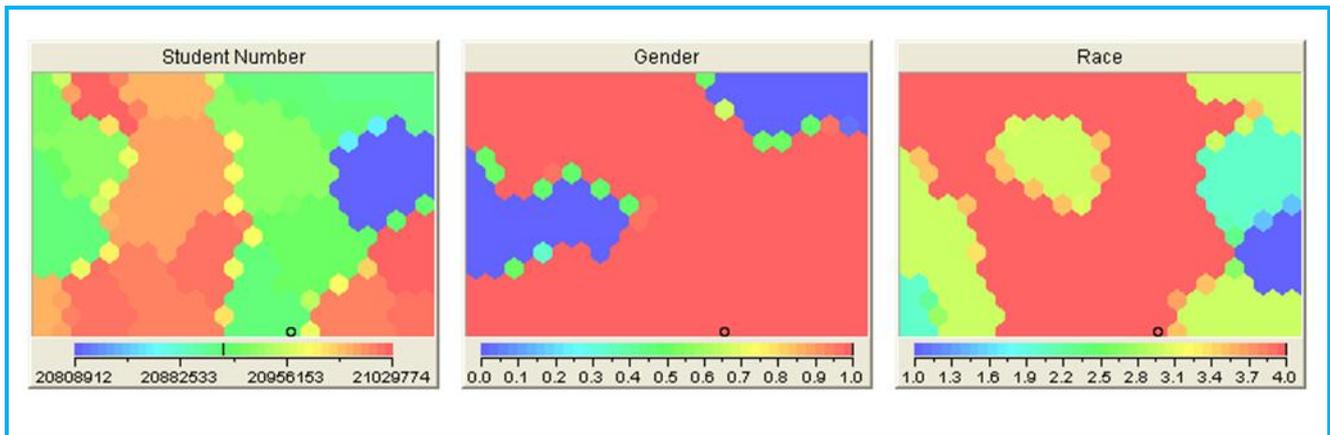


Figure 5.8 Component pictures for student number, gender and race in the experimental group

Key:

Student Number: Ascending order of learner identification (blue to red nodes)

Gender: Male = red nodes (1 on scale), Female = blue nodes (0 on scale)

Race: Black = red nodes (4 on scale), Asian = yellow nodes (3 on scale),
Coloured = light blue nodes (2 on scale), White = dark blue nodes (1 on scale)

According to Du (2010:15), appropriate visualisation of patterns can assist interpretation. “This is because human eyes are a powerful tool to identify visual patterns and trends”. The use of Viscovery SOMine component pictures helps to achieve this, as can be seen in Figure 5.9 and Figure 5.10. It must be borne in mind that the scale at the bottom of each component picture ranges from the minimum to the maximum node value of the respective component. Therefore, the ‘Exam Mark’ and the ‘Final Mark’ range from blue to red coloured nodes in ascending order of marks from 0 to 100%.

Similarly, the ‘Result’ component picture ranges from blue and green (1, 2 and 3 on scale) to yellow, orange and red (4, 5 and 6 on scale). This presents the learner results from failed to passed, as indicated in Figure 5.4. A notable observation from the data sets for the experimental and comparison groups shown in Table 5.19 and Table 5.20, is that the results for the experimental group did not contain the result ‘Fail, supplementary granted’. Similarly, the results for the comparison group did not contain the result ‘Fail, special exam granted’. Hence, the keys given below Figure 5.9 and Figure 5.10 are different from each other.

There is a visual higher frequency of red, orange and yellow nodes in the component pictures pertaining to the experimental group, when compared to the comparison group. This indicates superior performance on the part of the experimental group. This can be seen across all three maps.

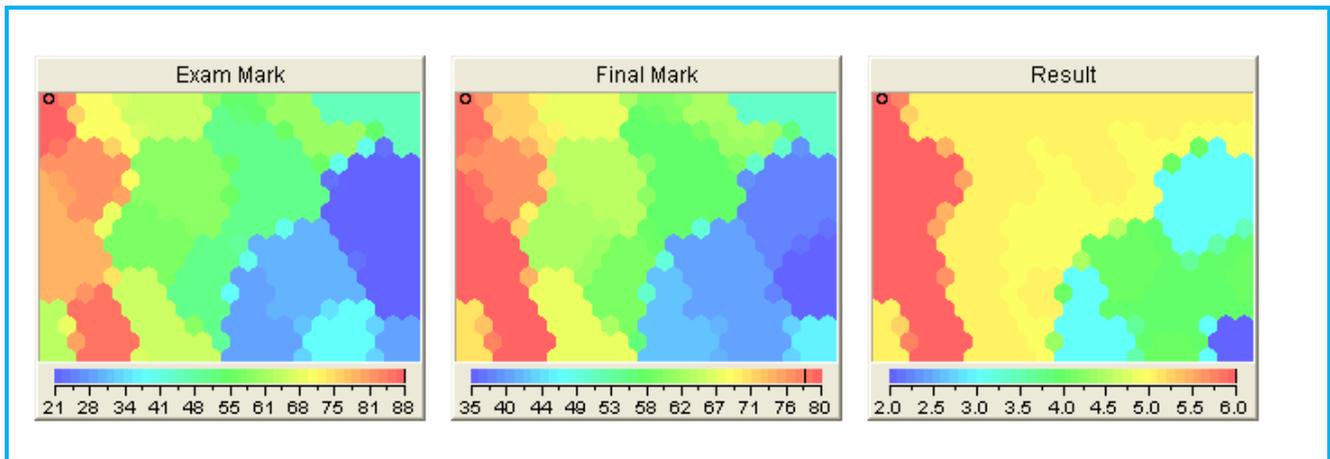


Figure 5.9 Experimental group – 2011

Key:

- Exam mark:** Ascending order of exam marks from 0 to 100% (blue to red nodes)
- Final mark:** Ascending order of final marks from 0 to 100% (blue to red nodes)
- Result:**
 - Pass with distinction = red and orange nodes (6 on scale),
 - Pass = dark/light yellow nodes (5 on scale),
 - Fail = green nodes (4 on scale),
 - Fail, special exam granted = light blue nodes (3 on scale),
 - Fail, sub minimum granted = dark blue nodes (2 on scale)

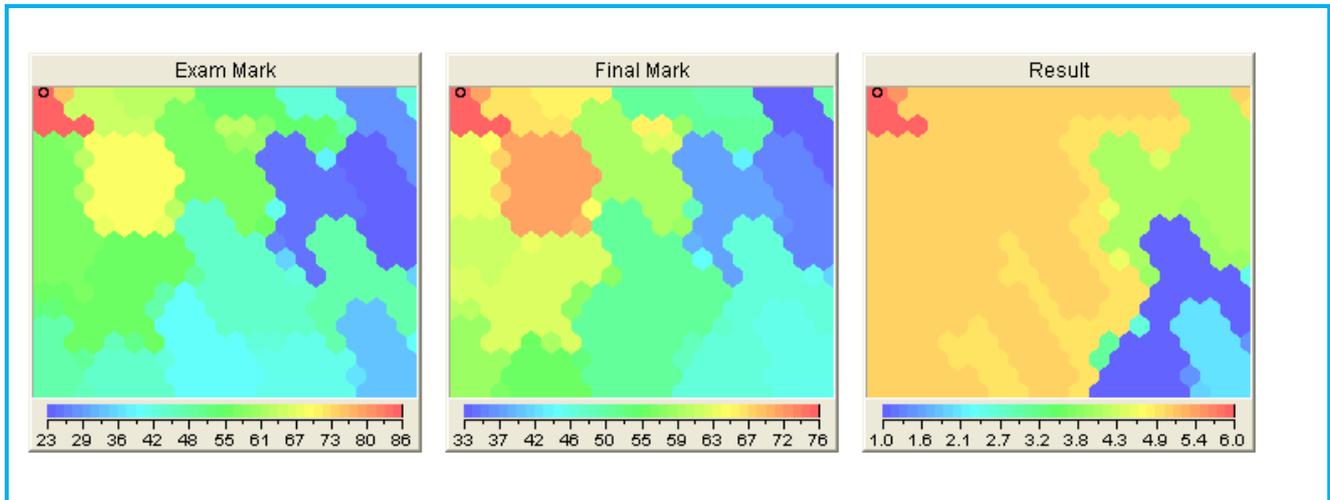


Figure 5.10 Comparison group – 2011

Key:

- Exam mark:** Ascending order of exam marks from 0 to 100% (blue to red nodes)
- Final mark:** Ascending order of final marks from 0 to 100% (blue to red nodes)
- Result:**
 - Pass with distinction = red nodes (6 on scale),
 - Pass = dark/light orange and yellow nodes (5 on scale),
 - Fail = green nodes (4 on scale),
 - Fail, sub minimum granted = light blue nodes (2 on scale),
 - Fail, supplementary granted = dark blue nodes (1 on scale)

The next section addresses certain problems encountered during Case Study 1.

5.7 Problem situations in Case Study 1, to be improved in Case Study 2

Certain issues that occurred during the course of Study 1, called for improvements to be made in the second study. One of the problems encountered was the attrition rate during the 2011 *Alice* workshop. Fifty participants were present at the beginning, but only twenty-one remained until the end and completed the questionnaire evaluation. This posed a limitation on Study 1, as it reduced the number of participants to less than the usual number desirable for statistical analysis. The statistician, however, was satisfied that this number was sufficient for a small-scale study. The researcher had informal discussions with the participants, and ascertained some of their reasons for the high drop-out rate:

- (a) Participants needed time during the lunch-break to relax and have meals.
- (b) Other lecturers from the Department of IT were utilising the lunch breaks to conduct additional lectures to complete their syllabi, making up for time lost due to the mass-action student strike that had occurred during February 2011.
- (c) Participants were insufficiently motivated to attend the workshop for the full duration.
- (d) The three-week duration proved to be too long to maintain participants' attention.

Pro-active measures were therefore implemented to sustain participation during the second study in 2012, so as to counteract the problems identified above and to motivate the participants to complete the 2012 *Alice* workshop. These measures are discussed in Section 6.1, Chapter 6.

The next section concludes and summaries the key findings of Case Study 1.

5.8 Summary and conclusion

The mixed-methods approach employed in this research involved quantitative and qualitative studies. This triangulated approach to data collection and analysis, contributed to confirming the findings. This occurred, for example, when the qualitative findings revealed an enhanced understanding of OOP concepts after use of *Alice*. This helped to confirm and support corresponding quantitative results (see Section 5.5.2.6). Similar findings also emerged from the quantitative and qualitative studies, where learners expressed difficulty with problem-solving and the application of programming concepts (see Section 5.5.2.4).

Findings from Case Study 1 in its entirety, indicate that learners:

- (a) lack problem-solving abilities;
- (b) are unable to grasp object-oriented programming concepts on an abstract level; and
- (c) spend insufficient time practicing programming exercises.

With the context depicted above and, in response to the research questions in Section 5.1, *Alice* has, firstly, shown itself to be an effective tool that addresses challenges faced by novice programming learners within the object-oriented domain. Furthermore, the learners' subsequent programming processes and performance in course assessment activities were investigated. The findings do not demonstrate a statistically significant difference in learner performance when using *Alice*, although participants in the experimental group were highly motivated by using *Alice* and achieved higher mean marks in the final assessment (Anniroot and de Villiers, 2012).

An analysis framework comprising two phases was implemented to structure the data collection and analysis process. This was presented in Section 5.2. Section 5.3 addressed the reliability of the questionnaire. The quantitative data analysis of the closed-ended questions in the questionnaire was discussed in Section 5.4, with the findings from qualitative open-ended responses described in Section 5.5. Quantitative data analysis of the test and exam learner data was considered in Section 5.6. Finally, Section 5.7 listed problems encountered during the course of Case Study 1.

This work sets the scene for the subsequent similar, though more extensive, study undertaken in 2012. Chapter 6 discusses the data collection and analysis in Case Study 2 of 2012.

Chapter 6: Data Collection and Analysis, Case Study 2

6.1 Introduction

This chapter is concerned with the analysis and interpretation of the qualitative and quantitative research data collected for Study 2, conducted during the academic year 2012. Chapter 6, addressing the second case study, follows a pattern that is analogous to the structure and style of Chapter 5, on the first case study. This involved an investigation of qualitative and quantitative data from the questionnaire survey and interviews, as well as analysis of the results of summative assessments. The questionnaire used in Case Study 1 was slightly extended by a few new questions and some of the original questions were amended. The qualitative sections in Chapter 6 are augmented with more extensive analysis than Chapter 5, in that they incorporate analysis and interpretation of post-questionnaire interviews of participants. The qualitative analysis of interview data was conducted using applied thematic analysis, followed by a quantification of the qualitative analysis using frequency counts.

This chapter differs from Chapter 5 in the following respects:

- (a) It should be noted by the reader that the discussions in Chapter 5 are rich and detailed in relating the findings to certain literature sources mentioned in previous chapters. Hence, these references and comparisons will not be repeated in Chapter 6, except where findings are different or are considered to be an interesting confirmation.
- (b) In each sub-section of Section 6.6, i.e. Sub-sections 6.6.2.1 to 6.6.2.7, the themes that emanated from the interviews are discussed in combination with the findings of the open-ended responses to the questionnaire.
- (c) The statistical analysis of the 2012 test and exam assessments in Section 6.7 of Chapter 6 is more extensive than the study of the 2011 test and exam assessments in Chapter 5.
- (d) This chapter presents a comparison between the quantitative results of Case Study 2 (closed-ended questions in the questionnaire) and two sets of qualitative findings of this study (open-ended questions in the questionnaire and interviews), as seen in Section 6.8.

Finally, a comparison between the core findings of Study 1 and Study 2 will be addressed in Chapter 7, the concluding chapter.

Further, the analysis framework implemented in Chapter 5, comprised the two phases:

- (a) data collection and preparation; and
- (b) data analysis and interpretation.

These two phases are mirrored in this chapter.

The recommendations implemented to address problems highlighted in Case Study 1 are reported in Section 6.2. The reliability of the questionnaire is reviewed in Section 6.3. Section 6.4 addresses the quantitative data analysis of the closed-ended questions in the questionnaire, with responses to the qualitative open-ended questions described in Section 6.5. The qualitative data analysis of the interviews is considered in Section 6.6. Section 6.7 presents the quantitative data analysis of the learners' test and exam data. Section 6.8 discusses the triangulation between the quantitative and qualitative findings of Case Study 2. Finally, the chapter concludes with an overall summary in Section 6.9.

The next section describes interventions used to mitigate the attrition encountered in the first study.

6.2 Recommendations implemented to address problems highlighted in Case Study 1

Following the problems regarding learner attrition in Study 1 (Section 5.7), a considerable improvement was observed by the researcher in the Study 2 situation. This was a result of proactive measures implemented to address the issues. Some of these measures are the following:

- (a) Refreshments were served to the participants at the end of each lesson;
- (b) The academic staff from the Department of IT were adequately informed of the *Alice* workshop programme;
- (c) Lecturers were requested to inform the researcher in the event of additional lectures being scheduled, so that alternative arrangements could be made for the *Alice* programme;
- (d) Incentives were provided in the form of book prizes and attendance certificates upon full attendance of the workshop;
- (e) The duration of the workshop for Study 2 was limited to a two-week period, but more days were utilised within each week.

These measures proved to be highly successful in terms of sustaining the entire complement of participants for the full duration of the second study. Consequently, participants displayed enthusiasm and a keen interest in the *Alice* visual environment.

The outputs produced by individual participants and those who preferred working in groups, towards submitting a non-compulsory *Alice* project were considered to be encouraging. The

researcher found some of these projects to be of a high standard, with well-thought out ideas for the storytelling scenarios, as well as well-structured, methodical code statements.

The reliability of the questionnaire is considered below.

6.3 Reliability of the questionnaire

A total of 55 participants responded to the questionnaire, as indicated by ‘N’ in Table 6.1. Fifty-two of the 55 participants, representing 95% of the sample, provided a complete set of responses to the closed-ended questions in the questionnaire. In contrast, due to their incomplete responses to the closed-ended questions, three participants (6%) were excluded from the sample.

Table 6.1 Case processing summary

		N	%
Cases	Valid	52	94.5
	Excluded	3	5.5
	Total	55	100.0

Reliability as previously defined (Section 5.3), is characterised by consistency, replicability, precision, measurability and accuracy. Ordinarily, this is demonstrated through a measurement model, for example, Cronbach’s Alpha, which dictates that reliability coefficients of 0.70 or greater, are acceptable correlation factors. It is noted in column 1 of Table 6.2 that the reliability value for Case Study 2 is 0.919, which is even higher than that of Case Study 1 (0.838) and indicates a high degree of acceptability. There is consistent scoring for the different categories listed in Study 2. The ‘N of items’ in column 2 of Table 6.2 refers to the 72 closed-ended questions in the questionnaire.

Table 6.2 Reliability statistics

Cronbach's Alpha	N of Items
0.919	72

The next section describes the quantitative data analysis of the closed-ended questions in the questionnaire.

6.4 Quantitative data analysis: Closed-ended questions in the questionnaire

Some refinements were made to improve the clarity and depth of questions in the closed-ended section of the questionnaire, where responses were in the form of options on a Likert scale (see Appendix C.3). This section contained a set of 25 categories comprising 72 questions. Eight out of 72 questions in the questionnaire were negatively-phrased questions for Study 2. This was done intentionally, so as to ascertain whether participants had read the questions carefully before answering. Prior to analysis, the ratings to negatively-phrased questions were reversed so as to get a uniform format. Quantitative data analysis was performed on the numeric responses to the 72 questions that made up the 25 categories. The component being addressed in this section is highlighted in Figure 6.1. Section 6.4.1 explains data collection and preparation, followed by data analysis and interpretation in Section 6.4.2.

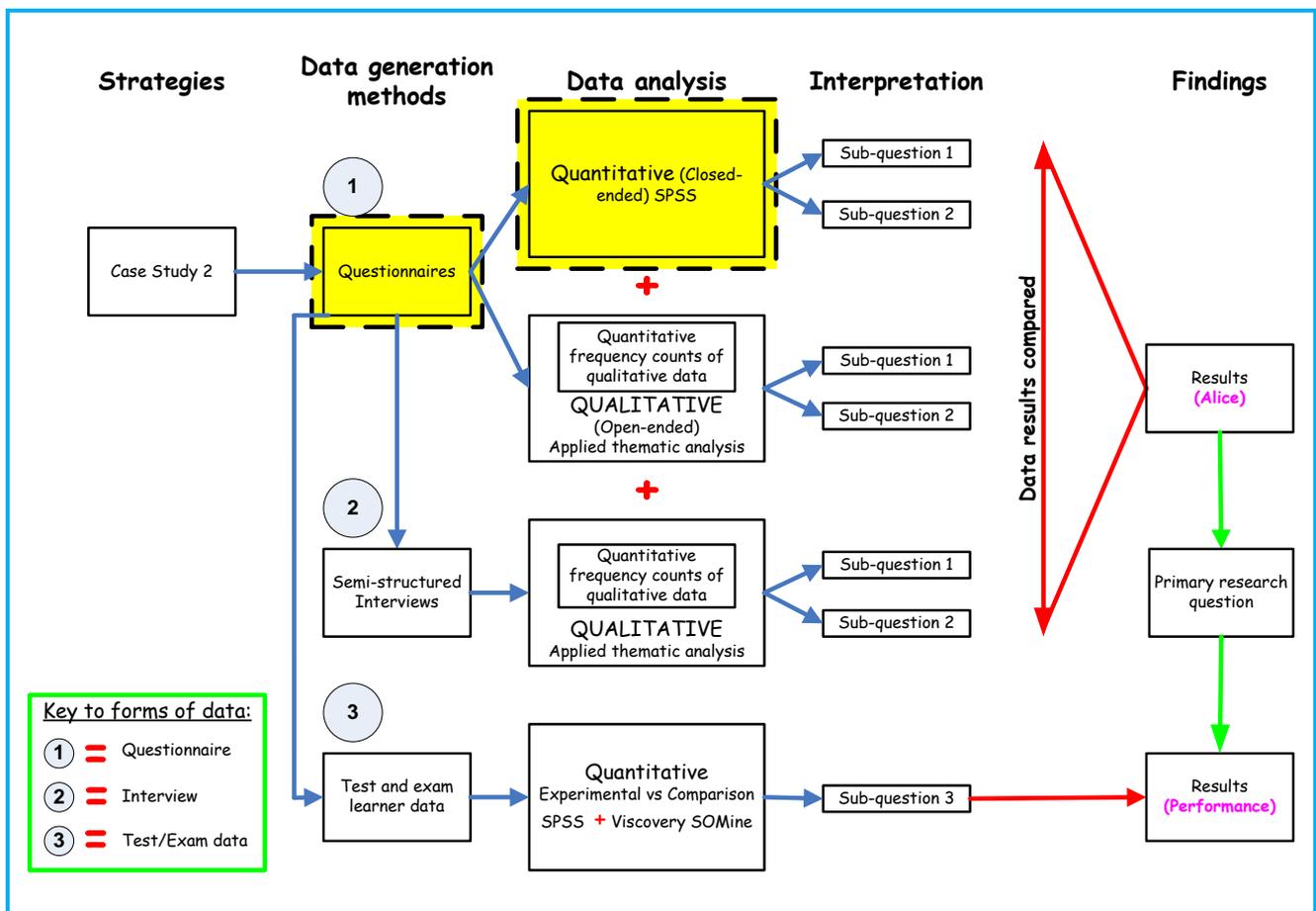


Figure 6.1 Detailed research processes of Case Study 2, highlighting the quantitative data analysis of the closed-ended responses to the questionnaire

6.4.1 Data collection and preparation: Closed-ended questions in the questionnaire

The questionnaire was completed by the experimental group only, since it would have been irrelevant to the comparison group. The data collected from the questionnaires is presented in Appendix D.2, indicating the overall results and averages for each question. The data collection and preparation process was described extensively in Section 5.4.1, and has been implemented similarly for the second study. Moreover, an additional stratification factor was included in the demographic data. Apart from gender and race, as shown in Table 5.3, an additional variable was included to indicate the participants' ages. This is demonstrated in Table 6.3.

Table 6.3 A segment of the data set of responses to closed-ended questions in the questionnaire

Participant	DEMOGRAPHIC DATA			RESPONSES										
	Gender	Race	Age	Q1.1	Q1.2	Q1.3	Q2.1	Q2.2	Q3.1	Q3.2	Q4.1	Q4.2	Q4.3	...
P1	1	4	20	4	5	5	4	5	4	4	4	5	5	...
P2	1	3	20	4	4	5	4	3	5	4	4	2	4	...
P3	1	3	19	5	5	5	5	5	5	4	5	4	5	...
P4	1	4	22	4	5	5	5	5	5	5	4	5	5	...
P5	0	3	19	4	3	4	5	5	5	5	5	5	5	...
P6	1	4	36	4	4	5	5	5	4	5	5	5	4	...
P7	0	3	20	4	5	3	5	4	4	5	5	4	5	...
P8	1	3	20	5	5	5	5	4	5	5	5	5	5	...
P9	0	4	23	4	5	5	5	5	5	4	5	5	5	...
P10	0	4	18	2	4	5	4	4	4	4	4	4	4	...
P11	1	4	19	5	3	5	5	5	5	5	5	5	5	...
P12	0	4	20	4	5	5	5	4	3	3	4	5	3	...
P13	1	4	20	4	5	4	4	4	3	3	3	4	4	...
P14	1	4	21	4	5	5	5	4	4	4	5	5	5	...
P15	1	3	19	5	4	4	5	4	4	3	4	5	4	...
P16	1	1	21	5	5	5	4	5	4	3	4	5	4	...
P17	0	4	20	3	5	5	5	4	4	4	4	4	3	...
P18	1	3	21	5	5	5	5	4	5	5	4	5	5	...
P19	0	3	19	5	5	5	5	5	5	5	4	5	4	...
P20	0	4	21	5	5	5	5	5	5	5	5	5	5	...
P21	1	4	20	4	5	5	5	5	5	4	4	4	5	...
P22	0	4	20	4	4	3	5	3	5	5	4	5	3	...
P23	0	4	25	4	4	5	5	5	4	4	4	5	4	...
P24	1	3	20	5	5	5	5	5	5	5	5	5	4	...
P25	1	4	21	5	5	5	5	5	5	5	5	5	4	...
P26	1	4	21	5	5	5	5	5	5	5	4	4	5	...
P27	1	4	20	5	4	5	4	2	3	4	3	4	4	...
P28	1	4	22	5	5	5	5	4	5	4	5	5	5	...
P29	1	4	22	4	3	4	5	4	5	4	4	4	4	...
P30	1	4	18	4	5	5	5	5	4	5	4	4	4	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Key:

Gender: M = 1, F = 0

Race: Black = 4, Asian = 3, Coloured = 2, White = 1

Q1.1, Q1.2,...Qn: Strongly Agree = 5, Agree = 4, Neutral = 3, Disagree = 2, Strongly Disagree = 1

The bivariate cross-tabulations mentioned by Willemse (2009), indicating an association between categorical variables, are summarised in Table 6.4, Table 6.5 and Table 6.6 below. The descriptive statistics based on the race and gender demographic information for Case Study 2, are shown in Table 6.4. The results reveal that a total of fifty-five learners completed the questionnaire, comprising forty-three males and twelve females. Furthermore, the racial constitution comprised 1.8% White, 32.7% Asian and 65.5% Black participants.

Table 6.4 Participant profile table, including bivariate race * gender cross-tabulation

			Gender		Total
			Male	Female	
Race	Black	Count	28	8	36
		% of Total	50.9%	14.5%	65.5%
	Asian	Count	14	4	18
		% of Total	25.5%	7.3%	32.7%
	White	Count	1	0	1
		% of Total	1.8%	0.0%	1.8%
Total	Count	43	12	55	
	% of Total	78.2%	21.8%	100.0%	

The descriptive statistics based on the age and gender demographic information for Case Study 2, are shown in Table 6.5. Ages of the participants ranged from eighteen to thirty-six. However, results reveal that the strata comprising learners aged nineteen to twenty-one, equated to a total of 72.7% of the participants.

Table 6.5 Participant profile table, including bivariate age * gender cross-tabulation

			Gender		Total
			Male	Female	
Age	18	Count	1	1	2
		% of Total	1.8%	1.8%	3.6%
	19	Count	9	3	12
		% of Total	16.4%	5.5%	21.8%
	20	Count	12	4	16
		% of Total	21.8%	7.3%	29.1%
	21	Count	11	1	12
		% of Total	20.0%	1.8%	21.8%
	22	Count	3	0	3
		% of Total	5.5%	0.0%	5.5%
	23	Count	4	1	5
		% of Total	7.3%	1.8%	9.1%
	24	Count	1	1	2
		% of Total	1.8%	1.8%	3.6%
	25	Count	0	1	1
		% of Total	0.0%	1.8%	1.8%
	26	Count	1	0	1
		% of Total	1.8%	0.0%	1.8%
	36	Count	1	0	1
		% of Total	1.8%	0.0%	1.8%
Total	Count	43	12	55	
	% of Total	78.2%	21.8%	100.0%	

The descriptive statistics based on the age and race demographic information for Case Study 2, are shown in Table 6.6. The results reveal that 21.8% of participants were in age group nineteen, of whom 7.3% were Black and 14.5% were Asian. Furthermore, in age group twenty, 18.2% were Black participants and 10.9% were Asian participants thus comprising a total of 29.1%. In the material stratum of 21 year olds, 12.7% were Black participants, 7.3% were Asian participants and 1.8% (only one) comprised White participants, i.e. 21 year olds constituted 21.8% overall of the participants.

Table 6.6 Participant profile table, including bivariate age * race cross-tabulation

		Race			Total	
		Black	Asian	White		
Age	18	Count	2	0	0	2
		% of Total	3.6%	0.0%	0.0%	3.6%
	19	Count	4	8	0	12
		% of Total	7.3%	14.5%	0.0%	21.8%
	20	Count	10	6	0	16
		% of Total	18.2%	10.9%	0.0%	29.1%
	21	Count	7	4	1	12
		% of Total	12.7%	7.3%	1.8%	21.8%
	22	Count	3	0	0	3
		% of Total	5.5%	0.0%	0.0%	5.5%
	23	Count	5	0	0	5
		% of Total	9.1%	0.0%	0.0%	9.1%
	24	Count	2	0	0	2
		% of Total	3.6%	0.0%	0.0%	3.6%
	25	Count	1	0	0	1
		% of Total	1.8%	0.0%	0.0%	1.8%
	26	Count	1	0	0	1
		% of Total	1.8%	0.0%	0.0%	1.8%
36	Count	1	0	0	1	
	% of Total	1.8%	0.0%	0.0%	1.8%	
Total	Count	36	18	1	55	
	% of Total	65.5%	32.7%	1.8%	100.0%	

6.4.2 Data analysis and interpretation: Closed-ended questions in the questionnaire

This section aims to interpret the data collected from responses to the closed-ended questions and present quantitative findings. As mentioned in Section 6.4.1, the questionnaire was administered to the experimental group only, since it would have been irrelevant to the comparison group. A discussion of each of the 25 categories in the questionnaire follows in Sub-sections 6.4.2.1 to 6.4.2.5. As explained in Chapter 4, the questions, which are in the form of evaluation statements,

are based on the theoretical evaluation criteria in Section 4.5.2, but suitably re-phrased. Certain findings of the closed-ended questionnaire analysis are discussed in relation to existing literature sources.

The first column of Table 6.7, Table 6.9, Table 6.10, Table 6.11 and Table 6.12 refers to the question numbers. The second column indicates the criterion on which that question was based. The distribution of percentages for participant responses in Case Study 1, as discussed in Section 5.4.2, was shown in three categories, namely 'D + SD', 'N' and 'A + SA'. However, Case Study 2 is a more intensive and in-depth study, hence five categories are considered in the discussions that follow. These responses are indicated in columns three to seven with respective sub-headings 'SD' [Strongly Disagree], 'D' [Disagree], 'N' [Neutral], 'A' [Agree] and 'SA' [Strongly Agree]. The average rating in response to each question is given in the eighth column, and matches the final column of Appendix D.2. In order to reflect an accurate average rate, the eight negatively-phrased questions, which span across Table 6.7, Table 6.9, Table 6.10, Table 6.11 and Table 6.12, were re-phrased in a positive direction and the response percentages were reversed.

The last column refers to the criterion numbers, which appear as the first column of Table 4.3, Table 4.4 and Table 4.5, the purpose of which is to make explicit the connection between the theoretical evaluation criteria in Chapter 4 and the evaluation statements in the questionnaire, as given in the current section. Finally, the category ratings at the bottom of each category are the average ratings for that category. These rows are highlighted by pink fill beneath each of the 25 categories.

6.4.2.1 Usability of *Alice* in relation to Nielsen's general interface design heuristics

The first ten items in the questionnaire that relate to usability, are founded on Nielsen's ten heuristics (Nielsen, 1994), as mentioned in Section 4.5.2.1 and Section 5.4.2.1. Participants exposed to the *Alice* intervention during lunch hours over a period of two-weeks, were suitably competent to evaluate the usability of the VPE in relation to these classic usability principles. The percentage distribution of participant responses to the questions in the first ten categories is provided in Table 6.7.

Table 6.7 Percentage distribution of participant responses for usability of Alice

General interface design heuristics, based on Nielsen (1994) and (Dix <i>et al.</i> , 2004), used to assess the usability of the <i>Alice</i> visual programming environment								
Criterion								
1	Visibility of the system status	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
1.1	I am always aware of what is going on in the system.	0.0	1.8	1.8	43.6	52.7	4.47	1.1
1.2	Whenever <i>Alice</i> is opened, the system indicates that it is in the process of loading the software.	0.0	3.6	12.7	20.0	63.6	4.44	1.1
1.3	When I save a world in <i>Alice</i> , the system indicates that files are being saved.	0.0	1.8	3.6	20.0	74.5	4.67	1.1
Category Rating		0.0	2.4	6.1	27.9	63.6	4.53	
2	Match between the system and the real world	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
2.1	The system uses words, terms and phrases that I can easily understand.	0.0	0.0	1.8	21.8	76.4	4.75	2.1
2.2	The templates used for new worlds and the objects in the system gallery, relate to real-world objects that I encounter in my day-to-day experiences.	0.0	3.6	9.1	34.5	52.7	4.36	2.2
Category Rating		0.0	1.8	5.5	28.2	64.5	4.55	
3	User control and freedom	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
3.1	I am comfortable with the level of control that I have over the system.	0.0	0.0	7.3	50.9	41.8	4.35	3.1
3.2	<i>Alice</i> allows me the flexibility to use the environment to perform a task.	0.0	1.8	10.9	34.5	52.7	4.38	3.1
Category Rating		0.0	0.9	9.1	42.7	47.3	4.36	
4	Consistency and standards	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
4.1	The <i>Alice</i> interface maintains a consistent look and feel.	0.0	0.0	9.1	58.2	32.7	4.24	4.3
4.2	The startup dialog box, play button, main menu and tab controls are clearly and consistently displayed.	0.0	1.8	3.6	32.7	61.8	4.55	4.2
4.3	The use of the mouse controls buttons and camera navigation controls are always available to move and arrange objects.	0.0	0.0	5.5	47.3	47.3	4.42	4.1
Category Rating		0.0	0.6	6.1	46.1	47.3	4.40	

5	Error prevention	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
5.1	The <i>Alice</i> software always gives error messages to prevent errors from occurring.	0.0	5.5	27.3	34.5	32.7	3.95	5.1
5.2	The <i>Alice</i> interface design does not cause me to make errors.	0.0	3.6	23.6	52.7	20.0	3.89	5.1
	Category Rating	0.0	4.5	25.5	43.6	26.4	3.92	
6	Recognition rather than recall	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
6.1	The actions to be taken and options available for selection are clear and visible at all times.	0.0	3.6	14.5	54.5	27.3	4.05	6.1 6.2
6.2	I do not have to remember the information from a previous screen in order to proceed with the next one.	1.8	32.7	30.9	25.5	9.1	3.07	6.1
	Category Rating	0.9	18.2	22.7	40.0	18.2	3.56	
7	Flexibility and efficiency of use	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
7.1	<i>Alice</i> caters for beginner to expert users.	1.8	0.0	14.5	40.0	43.6	4.24	7.1
7.2	I often use both the single view and the quad view to move, align and reposition objects on the screen.	0.0	9.1	21.8	36.4	32.7	3.93	7.1
	Category Rating	0.9	4.5	18.2	38.2	38.2	4.08	
8	Aesthetic and minimalist design	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
8.1	There is no irrelevant information in the <i>Alice</i> interface design that distracts me and slows me down.	1.8	16.4	16.4	49.1	16.4	3.62	8.1
	Category Rating	1.8	16.4	16.4	49.1	16.4	3.62	
9	Recognition, diagnosis and recovery from errors	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
9.1	<i>Alice</i> does not crash while I'm using it.	1.8	5.5	20.0	32.7	40.0	4.04	9.1
9.2	In cases where I encounter system errors, the system provides an appropriate error message in simple language.	1.8	7.3	21.8	43.6	25.5	3.84	9.2 9.3
9.3	I can recover from mistakes quickly and easily.	1.9	3.7	13.0	42.6	38.9	4.13	9.3
	Category Rating	1.8	5.5	18.3	39.6	34.8	4.00	

10	Help and documentation	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
10.1	The four tutorials in the startup dialog box are useful in helping me to learn how to use <i>Alice</i> .	0.0	1.8	10.9	49.1	38.2	4.24	10.1
10.2	The example worlds in the startup dialog box are useful.	0.0	0.0	12.7	52.7	34.5	4.22	10.1
	Category Rating	0.0	0.9	11.8	50.9	36.4	4.23	

Due to the classic role of Nielsen’s (1994) heuristics in evaluating usability, special attention is paid to them to this aspect by ranking the conformance of *Alice* to these heuristics. For seven of the ten interface design heuristics shown in Table 6.8, there is a general agreement ranging between 74.4 and 92.7 percent of responses and an average rating of 4.00 – 4.55.

The first column in Table 6.8 refers to the heuristic number, followed by the category name in Column 2, in descending order of highest-rated heuristics. Column 3 provides the average rate for the heuristic (i.e. the average of column 8 per category, extracted from the shaded rows in Table 6.7). Column 4 in Table 6.8 provides the sum of category ratings as shown in the sixth and seventh columns (A+SA) in Table 6.7. These findings indicate that, according to the participants, *Alice* provides good usability in seven of the ten aspects. Moreover, there is undeniable similarity in the data of Case Study 1 and Case Study 2. Six of the seven highlighted interface design heuristics in Case Study 1, as shown in Table 5.6, also occur among the highly rated heuristics in the second case study, indicating a notable degree of consistency over time and participants.

Table 6.8 Ranking of heuristics indicating usability of *Alice*

#	Heuristic (Highest → Lowest Rating)	Average Rate for heuristic	Average% of A+SA (Category rating)
2	Match between the system and the real world	4.55	92.7
1	Visibility of the system status	4.53	91.5
4	Consistency and standards	4.40	93.4
3	User control and freedom	4.36	90
10	Help and documentation	4.23	87.3
7	Flexibility and efficiency of use	4.08	76.4
9	Recognition, diagnosis and recovery from errors	4.00	74.4

The seven heuristics ranked highest in Case Study 2 (shown in Table 6.8) are compared with the seven highest-ranked heuristics in Case Study 1 (shown in Table 5.6):

- (a) The six heuristics that were common to the top seven in both studies are heuristic numbers 1, 2, 3, 4, 9 and 10.
- (b) An interesting observation was that *Visibility of the system status* and *Consistency and standards* featured within the top three heuristics of both studies.
- (c) *Flexibility and efficiency of use* appeared within the top seven heuristics in Case Study 2, but was not evident in Case Study 1 (see Table 5.6). Conversely, *Error prevention* was highly ranked in Case Study 1 but did not elicit the same learner responses in Case Study 2.
- (d) In the large-group study, Case Study 2, with 55 participants, the participants rated *Match between the system and the real world* as the best heuristic, whereas it had been ranked fourth in Case Study 1.
- (e) It was mentioned in Section 5.4.2.1, that handouts were given to participants in the 2012 workshop to facilitate the teaching and learning of OOP using *Alice*. The researcher therefore spent less time demonstrating the tutorials and example worlds available in the startup dialog box, than she had in Case Study 1. It appears that this change in teaching style impacted on the ranking of *Help and documentation*, bringing it down from first place in Case Study 1 to fifth place in Case Study 2.

Category ratings greater than 4.00 were regarded as positive. In Case Study 2, three cases demonstrated that the participants' perceptions regarding the usability of *Alice* were somewhat less positive, although still relatively high. These aspects are discussed in descending order of category ratings.

In addressing *error prevention* (Heuristic 5), 67.2% (combined agreement) of the experimental participants agreed that the *Alice* software always gives error messages to prevent errors from occurring (Question 5.1). Furthermore, 72.7% (combined agreement) felt that the *Alice* interface design does not cause the learner to make errors (Question 5.2). In general, this indicates a positive impression and shows that users were fairly satisfied with the level of error prevention that *Alice* has to offer, although the responses were less positive than those summarised in Table 6.8.

With regards to *aesthetic and minimalist design* (Heuristic 8), 65.5% (combined agreement) of the participants felt there was no irrelevant information in the *Alice* interface design that distracted learners and slowed them down (Question 8.1). With an average rating of 3.62, the heuristic is indicative that *Alice* was designed with a goal of appealing aesthetics.

Recognition rather than recall (Heuristic 6) is the lowest-rated heuristic with a category rating of 3.56. A high percentage (81.8% combined agreement) of participants agreed that the actions to be taken and options available for selection were clear and visible at all times (Question 6.1). In contrast, only 34.6% (combined agreement) of the participants found that they did not have to remember the information from a previous screen in order to proceed with the next one (Question 6.2). However, 30.9% were unsure about this statement and 34.5% (combined disagreement) disagreed with it, implying that they explicitly needed to remember facts.

6.4.2.2 The teaching and learning of programming

It has been highlighted in the problem statement (Section 1.3) and in subsequent chapters, that learner attrition is a major concern amongst academic staff at the Department of Information Technology at DUT. This is supported with statistical data given in Section 5.4.2.2.

This section of the questionnaire is devoted to:

- (a) extracting learner feedback pertinent to their levels of experience in computer usage prior to studying at DUT;
- (b) assessing learner ability to cope with the volume of course work in the ND: IT qualification;
- (c) evaluating the effectiveness of teaching styles at DUT; and
- (d) gauging the level of confidence of the learners in understanding OOP.

The percentage distribution of participant responses pertaining to each of these factors, with particular reference to DUT, is provided in Table 6.9.

Table 6.9 Percentage distribution of participant responses for teaching and learning of programming

The teaching and learning of programming								
11	Learner attrition	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
11.1	I had a high level of comfort with computer usage prior to studying programming.	0.0	0.0	5.5	38.2	56.4	4.51	11.1
11.2	I am able to cope with the breadth and depth of the course work covered in the programming subjects of this course.	0.0	1.8	14.5	34.5	49.1	4.31	11.1
11.3	The current teaching methods and techniques used in programming subjects helped me to program successfully.	0.0	1.8	12.7	45.5	40.0	4.24	11.2 11.3
11.4	I had a high level of comfort with object-oriented programming prior to the <i>Alice</i> workshop.	0.0	3.6	23.6	45.5	27.3	3.96	11.4

11.5	I like to learn object-oriented programming in a more fun and interactive way.	1.8	0.0	0.0	36.4	61.8	4.56	11.4
Category Rating		0.4	1.5	11.3	40.0	46.9	4.32	

Table 6.9 indicates congruency with previous findings regarding the sample of learners participating in the study. It is noted that the pre-requisites used to extract a representative sample, again contributed to a *high comfort level of participants in respect of prior computer usage* (94.6% combined agreement for Question 11.1) and consequently the ability to cope with the breadth and depth of programming concepts (83.6% combined agreement for Question 11.2).

On the two issues of *coping with the large volumes of course work* to be covered within programming subjects (Question 11.2) and whether *teaching methods and techniques employed to conduct courses promoted learning* (Question 11.3), 83.6% and 85.5% of participants respectively indicated an agreeable or strongly agreeable response. As mentioned in Section 5.4.2.2, assertions by Moskal *et al.* (2004) and Stiller (2009) suggest that learners without prior programming experience are likely to be overwhelmed by the breadth and depth of material, and that curricular or pedagogic techniques may reduce the initial enthusiasm of learners. However, these are again considered as unlikely reasons for attrition in the present case. The high percentage of participant satisfaction pertaining to both the extent of course work and the teaching styles, again emphasises the need for further investigation of the cause of this problem.

While 45.5% of the experimental participants indicated an *above-average level of comfort with OOP* prior to the *Alice* workshop, a little under a quarter (23.6%) were unsure and a little over a quarter (27.3%) expressed strongly agreeable opinions regarding their comfort levels (Question 11.4). In Study 2, a very high percentage (98.2%) of participants affirmed that *learning OOP in a more fun and interactive way* was appealing (Question 11.5). The participants were clearly motivated to learn OOP via the *Alice* intervention. As mentioned in Section 5.4.2.2, Jenkins' (2001) proposition that motivation of learners is a key issue in learning, was affirmed in this case as well. The use of *Alice* in supporting OOP instruction appears to increase learner motivation, thereby supporting its use in tertiary institutions.

6.4.2.3 Challenges faced by learners in learning OOP

It has been previously highlighted that novice programmers face various challenges and difficulties in learning OOP. These core issues include, but are not limited to:

- (a) demotivation of learners;

- (b) the complex syntax and semantics of OOP languages;
- (c) the learners' need for feedback; and
- (d) the difficulties experienced in understanding compound logic and algorithmic problem-solving.

Table 6.10 provides participant responses in relation to these core challenges. Some questions relate to general experiences of learning OOP. Others refer implicitly to the features of *Alice*, while not using the name '*Alice*' and yet others are explicit in referring to *Alice* as a positive intervention.

Table 6.10 Percentage distribution of participant responses for challenges facing learners of OOP

Challenges faced by learners in learning object-oriented programming								
12	Lack of motivation for programming	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
12.1	I am motivated to learn programming.	1.8	0.0	1.8	36.4	60.0	4.53	12.1
12.2	I spend a lot of time intensively practicing programming exercises.	0.0	5.5	34.5	38.2	21.8	3.76	12.2
12.3	<i>Alice</i> has improved my motivation for programming.	1.8	1.8	9.1	54.5	32.7	4.15	
	Category Rating	1.2	2.4	15.2	43.0	38.2	4.15	
13	Fragile mechanics of program creation, particularly syntax	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
13.1	Learning the syntax and semantics of a programming language is challenging.	3.6	14.5	34.5	29.1	18.2	3.44	13.1
13.2	It would be easier for me to learn how to solve a problem and learn basic concepts of object-orientation without having to deal with brackets, commas and semicolons, as is the case with <i>Alice</i> .	3.6	12.7	14.5	29.1	40.0	3.89	13.1
13.3	I have felt intimidated by direct exposure to programming syntax, such as do-while, if-then-else, switch-case, and for-next.	14.5	30.9	29.1	18.2	7.3	2.73	13.2
13.4	The textual nature of the conventional programming environments I use, makes it difficult for me to learn how to program.	16.7	61.1	18.5	3.7	0.0	2.09	13.3
	Category Rating	9.6	29.7	24.2	20.1	16.4	3.04	

14	Identifying results of computation as the program runs	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
14.1	I am able to identify errors and correct them using the feedback given by the program.	0.0	3.6	20.0	54.5	21.8	3.95	14.1 14.2
14.2	I am able to work independently on a program, from coding to testing.	1.8	0.0	5.5	60.0	32.7	4.22	14.2
14.3	My experience from running and debugging programs allows me to solve similar problems.	0.0	1.8	9.1	61.8	27.3	4.15	14.2
14.4	Alice provides immediate feedback as the program runs.	0.0	0.0	20.0	50.9	29.1	4.09	
Category Rating		0.5	1.4	13.6	56.8	27.7	4.10	
15	Difficulty of understanding compound logic	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
15.1	I am able to apply basic problem-solving techniques to create algorithms that solve problems.	0.0	1.8	20.0	60.0	18.2	3.95	15.1
15.2	I have a good understanding of pseudocode.	5.5	5.5	20.0	49.1	20.0	3.73	15.1
15.3	I use pseudocode to outline and understand the logic of a program before I start coding.	9.1	23.6	32.7	27.3	7.3	3.00	15.1
15.4	I am able to break down a large and complex programming task into smaller subtasks.	1.8	9.1	23.6	47.3	18.2	3.71	15.1
15.5	Alice allows me to focus on problem-solving.	0.0	1.8	12.7	58.2	27.3	4.11	
Category Rating		3.3	8.4	21.8	48.4	18.2	3.70	

Lack of motivation for programming: On aggregate for agreeable responses, 96.4% of the experimental participants indicated that they were motivated to learn programming (Question 12.1). Sixty percent admitted to spending a lot of time intensively practicing programming exercises, further indicating a notable degree of correlation between motivation and practice (Question 12.2). Approximately a third (34.5%) were unsure whether they spend a lot of time intensively practising. As mentioned in Section 5.4.2.3, the assertion by Law *et al.* (2010) has relevance. They state that in order to develop good programming skills, learners should undertake a great deal of intensive practice on programming exercises to gain experience in debugging. However, the learner must be adequately motivated to sustain this level of competence. Further investigation may be required to establish the reasons behind the indecisiveness of this strata of the learners. However, it is feasible that the responses regarding Question 12.2 were based on prudence on the participants' part, due to the qualifications '...a lot of time...', and '...intensively practising...'. In addressing this important issue, it was considered pertinent to augment the questionnaire with one explicit additional

statement regarding *Alice* as an effective motivational intervention (Question 12.3). Correspondingly, 87.2% (combined agreement) of participants agreed with this statement.

Complex syntax, logic and semantics: A fair percentage (47.3%) of the experimental participants agreed that learning the syntax and semantics of a programming language was challenging, while 34.5% were unsure (Question 13.1). A good percentage (69.1%) agreed that *Alice* promotes learning how to solve problems and learning basic concepts of object-orientation without having to deal with brackets, commas and semicolons, thus indicating appreciation for *Alice*'s drag-and-drop feature, which releases learners from having to deal with complex syntax. Fifteen percent (14.5%), however, remained unsure (Question 13.2). Forty-five percent (45.4%) indicated that they were not intimidated by direct exposure to programming syntax. Twenty-nine percent (29.1%) were unsure and only 25.5% agreed with the statement (Question 13.3). A high percentage (77.8%) of participants in the experiment disagreed that the textual nature of conventional programming environments makes it difficult for them to learn how to program, while another 18.5% were unsure (Question 13.4). To refer again to the suggestion made by Carlisle (2009), mentioned in Section 5.4.2.3, that the textual nature of most programming environments works against typical learning styles, the results of this study do not appear to support Carlisle's assertions. Many participants indicated a high level of comfort with learning OOP by conventional means, in spite of enjoying a visual experience with *Alice*. This is congruent to the findings of Study 1.

The need for immediate feedback and identifying results of computation as a created program runs: Seventy-six percent (76.3%) found that they were able to identify errors and correct them using the feedback given by the program (Question 14.1). Study 2 incorporated an additional question that addressed this assertion directly (Question 14.4). The response to this question indicated an overwhelming feeling by the participants (80%) that *Alice* provided immediate feedback as the program executes. Ninety-three percent (92.7%) were able to work independently on a program, from coding to testing (Question 14.2). Due to experience from running and debugging programs, 89.1% of the participants felt that they were equipped to solve similar problems (Question 14.3).

Difficulties in understanding compound logic and the application of algorithmic problem-solving skills: Eighty-six percent (85.5%) of participants explicitly indicated that *Alice* allowed them to focus on problem-solving (Question 15.5). The confidence levels of the participants in the experiment were quite high, in that 78.2% claimed they were able to apply basic problem-solving techniques to create algorithms that solve problems (Question 15.1); 69.1% indicated that they have a good understanding of pseudocode (Question 15.2); and 65.5% claimed that they were able to

decompose a large and complex programming task into smaller subtasks (Question 15.4). Conversely, only 34.6% acknowledged using pseudocode to outline and understand the logic of a program before they started coding (Question 15.3). Approximately one third (32.7%) of the participants disagreed with using pseudocode to help them understand the logic. There is a general inconsistency between the responses in Study 2, as was the case with Study 1. This contradicts the participants' claims that they do not experience difficulty in understanding compound logic.

6.4.2.4 How to improve the teaching of OOP

As previously mentioned in Section 5.4.2.4, four important techniques that help teach OOP have been identified to address the challenges of learning OOP. These techniques were proposed in the questionnaire, and include:

- (a) algorithmic thinking and expression;
- (b) abstraction;
- (c) objects-first strategy; and
- (d) 3D animation authoring tools and visualisation.

The participants' responses to the final section on 3D animation authoring tools and visualisation were strongly influenced by their experiences with *Alice*. These techniques are listed in Table 6.11 with the percentage distributions of participant responses to each.

Table 6.11 Percentage distribution of participant responses for techniques to improve teaching of OOP

How to improve the teaching of object-oriented programming								
16	Algorithmic thinking and expression	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
16.1	I am able to read and write in a formal language.	0.0	0.0	3.6	54.5	41.8	4.38	16.1
	Category Rating	0.0	0.0	3.6	54.5	41.8	4.38	
17	Abstraction	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
17.1	I am able to use creative thinking and programming concepts to write programs.	0.0	0.0	9.1	61.8	29.1	4.20	17.1 17.2 17.3 17.4
17.2	When I write a program, I usually understand every line of code.	5.5	32.7	21.8	27.3	12.7	3.09	17.1
	Category Rating	2.7	16.4	15.5	44.5	20.9	3.65	

18	Objects-first strategy	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
18.1	I have a good level of understanding of objects, gained from my first year of study.	0.0	7.3	23.6	41.8	27.3	3.89	18.1
18.2	It would be easier to learn object-oriented programming during the first year of study, and later learn other control structures such as loops, If statements etc.	9.1	18.2	14.5	20.0	38.2	3.60	18.2
18.3	<i>Alice</i> helps me to see everything as an object.	0.0	1.9	7.5	50.9	39.6	4.28	18.3 18.4
Category Rating		3.1	9.2	15.3	37.4	35.0	3.92	
19	3D animation authoring tools and visualisation	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
19.1	A visual representation improves my understanding of programming concepts.	0.0	0.0	14.5	43.6	41.8	4.27	19.1 19.2
19.2	Three-dimensionality makes objects seem real.	0.0	1.8	7.3	45.5	45.5	4.35	19.4
19.3	The visual effects in <i>Alice</i> provide a meaningful context for understanding classes, objects, methods, and events.	0.0	1.8	3.6	49.1	45.5	4.38	19.3 19.6
19.4	I am able to use <i>Alice</i> to write a new method to make objects perform animated tasks, such as hop, fly, swim etc.	0.0	0.0	5.5	27.3	67.3	4.62	19.5
Category Rating		0.0	0.9	7.7	41.4	50.0	4.40	

Algorithmic thinking and expression: In the present study, 96.3% of the group agreed that they were able to read and write in a formal language (Question 16.1).

Abstraction: Although 90.9% agreed that they were able to use creative thinking and programming concepts to write programs (Question 17.1), 38.2% of the participants in this experiment, acknowledged that on occasions they do write programs without understanding each piece of program code (Question 17.2). With a further 21.8% of experimental participants being unsure about their ability to understand every line of code when writing a program, it becomes clear that at some time or the other, 60% of the participants were unable to grasp programming concepts on an abstract level. These findings are similar to those of Study 1.

Objects-first strategy: While 69.1% of the experimental participants agreed that they have a good level of understanding of objects, gained from their first year of study, 7.3% had disagreed (Question 18.1). A fair percentage (58.2%) of the participants felt confident that it would be easier to learn OOP during the first year of study, and later learn the conventional control structures such as loops, if statements etc., while 27.3% disagreed (Question 18.2). A high percentage (90.5%) of

the experimental participants stated that *Alice* helps them to see everything as an object (Question 18.3).

3D animation authoring tools and visualisation: As stated previously, participants' responses to this criterion were positively influenced by their exposure to *Alice*. For instance, a very high percentage (85.4%) of the experimental participants agreed that a visual representation improves their understanding of programming concepts (Question 19.1). Furthermore, 94.6% agreed that the visual effects in *Alice* provide a meaningful context for understanding classes, objects, methods, and events (Question 19.3) and that they were able to use *Alice* to write a new method to make objects perform animated tasks, such as hopping, flying, swimming etc. (Question 19.4). Finally, 91% of the experimental participants agree that three-dimensionality makes objects seem real (Question 19.2).

6.4.2.5 Criteria based on the researcher's experience

The final part of the open-ended section in the questionnaire is based on criteria identified by the researcher as a result of her personal 10-year experience in teaching OOP to IT learners. The percentage distributions of the responses to these criteria are provided in Table 6.12.

Table 6.12 Percentage distribution of participant responses for criteria based on researcher's experience

The researcher's experience								
20	Appreciation of trial and error	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
20.1	When using <i>Alice</i> , I use trial and error to 'try out' individual animation instructions as I create new methods.	1.8	7.3	25.5	49.1	16.4	3.71	20.1
20.2	I can visibly see the effect that each new animation instruction has on the animation.	1.8	12.7	30.9	50.9	3.6	3.42	20.2
	Category Rating	1.8	10.0	28.2	50.0	10.0	3.56	
21	Incremental construction approach	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
21.1	<i>Alice</i> has taught me how to program incrementally i.e. I write one method at a time, testing and running each piece.	0.0	0.0	14.5	49.1	36.4	4.22	21.1
	Category Rating	0.0	0.0	14.5	49.1	36.4	4.22	

22	Impact of <i>Alice</i> on understanding OOP concepts	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
22.1	Inheritance	0.0	3.6	23.6	41.8	30.9	4.00	22.1
22.2	Methods	0.0	0.0	14.5	43.6	41.8	4.27	22.2
22.3	Properties	0.0	0.0	18.2	47.3	34.5	4.16	22.3
22.4	Functions	0.0	1.8	10.9	50.9	36.4	4.22	22.4
	Category Rating	0.0	1.4	16.8	45.9	35.9	4.16	
23	Impact of <i>Alice</i> on understanding basic programming concepts	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
23.1	Loops	0.0	0.0	20.0	38.2	41.8	4.22	23.1
23.2	If..statements	0.0	3.6	18.2	40.0	38.2	4.13	23.2
23.3	Data types	1.8	5.5	27.3	32.7	32.7	3.89	23.3
23.4	Event-driven programming	1.8	1.8	25.5	41.8	29.1	3.95	23.4
	Category Rating	0.9	2.7	22.7	38.2	35.5	4.05	
24	Ability to collaborate	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
24.1	The <i>Alice</i> workshop has provided the opportunity to work in pairs with other learners and I have chosen to do so.	0.0	5.5	20.0	47.3	27.3	3.96	24.1
24.2	I have been able to interact and communicate well with other learners.	0.0	5.5	10.9	60.0	23.6	4.02	24.1
24.3	The experience of working with other learners has helped me to learn the <i>Alice</i> programming environment.	0.0	7.3	20.0	52.7	20.0	3.85	24.1
	Category Rating	0.0	6.1	17.0	53.3	23.6	3.95	
25	Impact of <i>Alice</i> on DS2 learners	SD (%)	D (%)	N (%)	A (%)	SA (%)	Avg Rating	Crit#
25.1	The <i>Alice</i> workshop relates directly to the sections on object-oriented programming covered in the Development Software 2 syllabus.	0.0	9.1	7.3	49.1	34.5	4.09	25.1
25.2	I am interested in learning more about computer graphics and animation.	0.0	1.8	5.5	25.5	67.3	4.58	25.2
25.3	I am interested in learning and working more with the <i>Alice</i> visual programming environment.	0.0	0.0	7.3	36.4	56.4	4.49	25.3
25.4	I used <i>Alice</i> during my personal time after attending the first lesson of the <i>Alice</i> workshop.	1.8	10.9	16.4	41.8	29.1	3.85	25.4
25.5	I am interested in learning more about object-oriented programming.	0.0	0.0	0.0	34.5	65.5	4.65	25.5
	Category Rating	0.4	4.4	7.3	37.5	50.5	4.33	

As was the case with Study 1, a fairly high number (65.5%) of the experimental participants agreed they had used *trial-and-error* to ‘try out’ individual animation instructions, whilst creating new methods, when using *Alice* (Question 20.1). Moreover, 54.5% of participants could visibly see the effect that each new animation instruction has on the animation (Question 20.2). A high percent (85.5%) of participants in the experimental group agreed that *Alice* had taught them to *program incrementally* (Question 21.1). This response is indicative of yet another similarity to Case Study 1. As mentioned in Section 5.4.2.5, it would be encouraging to note an improvement in the learners’ ability to construct methods and functions in conventional programs, resulting from their exposure to the *Alice* VPE.

Regarding the four core concepts that form a foundation for learning OOP, 72-88% of participants agreed that *Alice* had helped them to *understand inheritance, methods, properties and functions* (Questions 22.1 to 22.4). Furthermore, an above-average percentage of 65-80% agreed that *Alice* helped to improve their *understanding of iteration, selection, data types and event-driven programming* (Questions 23.1 to 23.4). However, one must take cognisance of the fact that, while the learners were being exposed to OOP for the first time during their second year, they had already been exposed to the basic programming concepts during their first year.

A good percentage (74.6%) of participants agreed that the *Alice* workshop had given them the opportunity to work in pairs with other learners (Question 24.1). Moreover, 72.7% found that this *collaboration* had helped them to learn how to use *Alice* (Question 24.3). With regard to team work, a high 83.6% had been able to interact and communicate well with other learners (Question 24.2). The knowledge acquired by the participants on learner collaboration during their interaction with *Alice*, will equip them with the necessary skills to succeed with their third-year course work.

A high percent (83.6%) of the experimental participants found that the *Alice* workshop *related directly to OOP concepts* covered in the DS2 syllabus, whilst 7.3% were unsure and 9.1% disagreed (Question 25.1). The majority of participants (92.8%) expressed interest in learning more about computer *graphics and animation* (Question 25.2), with 92.8% *eager to work more with Alice* (Question 25.3), and 100% of participants *wanting to learn more about OOP* (Question 25.5). Seventy-one percent (70.9%) had used *Alice* during their *personal time* after the workshop had commenced (Question 25.4). The positive impact of using *Alice* in teaching and learning OOP thus encourages its future use.

The next section describes the qualitative data analysis of responses to the open-ended questions in the questionnaire.

6.5 Qualitative data analysis: Open-ended questions in the questionnaire

As in Case Study 1, qualitative data analysis was performed on responses to Category 26 of the questionnaire, with its six open-ended questions (see Appendix C.3). The participants' positive and negative experiences with using *Alice* were expressed in written-form. They were also required to highlight the challenges they are faced with in learning OOP and to suggest techniques that could improve the teaching of OOP. Finally, the impact of *Alice* on the learners' understanding of OOP was addressed, and the effectiveness of the *Alice* VPE in addressing the challenges identified by the learner. The focal point for discussion in this section is highlighted in Figure 6.2. Section 6.5.1 explains data collection and preparation, followed by data analysis and interpretation in Section 6.5.2.

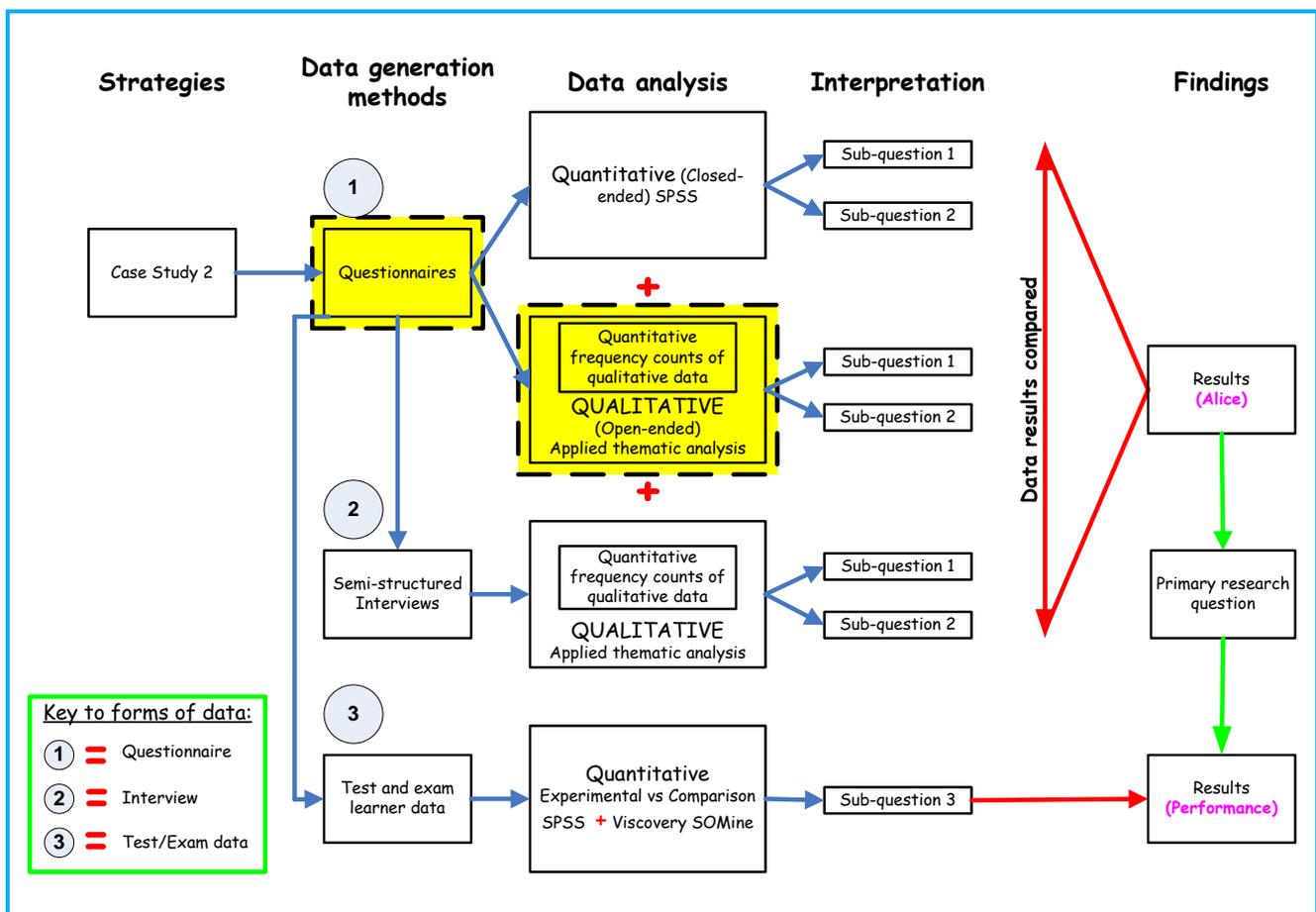


Figure 6.2 Detailed research processes of Case Study 2, highlighting a quantification of qualitative data derived from the open-ended responses to the questionnaire

6.5.1 Data collection and preparation: Open-ended questions in the questionnaire

As mentioned in Section 4.8.2 and Section 5.5.1, this was implemented using an applied thematic analysis approach to extract themes from the open-ended free-text responses. Furthermore, a quantification of the qualitative data derived from the open-ended responses to the questionnaires provided statistical findings relating to the textual data. Appendix E.1 shows the frequency counts of participant responses in the various domains. No data preparation was required, as the researcher studied the participants' text verbatim.

6.5.2 Data analysis and interpretation: Open-ended questions in the questionnaire

Analysis was conducted manually for each open-ended question, in a quest to establish relationships and themes. During this process, valid and relevant data was transcribed into the codebook, as described in Section 4.8.2.3 and Section 5.5.2. The codebook is provided as Appendix E – Qualitative Analysis Tables. Once the codebook for Case Study 2 had been created and expanded in a second round of thematic analysis, substantive corroborative interpretations were extrapolated from it to reach viable and pertinent conclusions. This will be addressed in Sections 6.5.2.1 to 6.5.2.7.

As stated previously, the comparison between these findings and the results from the 2011 cohort, will be discussed in Chapter 7.

6.5.2.1 General overview comments: Open-ended questions in the questionnaire

Participants' personal opinions and general attitudes emerged from their qualitative responses to the open-ended questions, as shown in the first columns of Table 6.13 and Table 6.14. These general overview comments did not fall within any of Nielsen's ten heuristics. The second columns indicate the number of participants who mentioned that theme. The frequency percentages in the last column of each table convert the number to the percentage of participants who gave a response corresponding to that theme.

Table 6.13 Positive overview comments on the usability *Alice* – Open-ended questions in the questionnaire

Positive overview comments	Frequency counts (n=55)	%
I like <i>Alice</i>	46	84
<i>Alice</i> is a fun/exciting/enjoyable/interesting way to learn programming	17	31
Improved understanding of DS2 course work	4	7
Suitable for novice programmers, e.g. high school learners or first-year learners starting OOP	3	5
Promotes learner collaboration	1	2
Promotes self-learning; anticipated future personal use	3	5
Total	74	-

Eighty-four percent of participants in the experimental group expressed a liking for *Alice*. A pattern emanating from 31% showed they found *Alice* to be a fun, engaging tool that helps them to learn OOP. Seven percent of participants indicated that *Alice* had improved their understanding of the DS2 course work. Three participants (5%) suggested that *Alice* can be optimally used by novice programmers, whilst another participant stated that *Alice* lends itself to collaborative learning. Although this was explicitly mentioned by only one participant, team work was observed throughout the workshop and nine groups of participants spontaneously did collaborative projects over and above the required activities.

As was the case in Study 1, this affirms the findings of Dickey (2003) who believes that 3D virtual worlds such as *Alice*, provide means of creating rich and engaging programs to support learning and collaboration (see Section 5.5.2.1). A theme that emerged from three participants (5%) showed that using *Alice* had improved their motivation to self-learn and had developed their interest in further use of the VPE in their personal capacity. As with Case Study 1, this study supports Hadjerrouits' (2008) research findings on *constructivist* learning, mentioned in Section 5.5.2.1. He stated that learners are able to learn better when they explore and discover things themselves.

Table 6.14 Negative overview comments on the usability of *Alice* – Open-ended questions in the questionnaire

Negative overview comments	Frequency counts (n=55)	%
I don't like <i>Alice</i>	1	2
Time consuming – takes practice getting accustomed to <i>Alice</i> at first exposure	5	9
Total	6	-

Only one (2%) participant expressed a dislike for *Alice*. Nine percent of participants felt it was time-consuming to practice and develop the necessary skills to use *Alice*. However, it is notable that these negative remarks came from only six of the 55 participants in Case Study 2.

6.5.2.2 Spontaneous positive responses on usability of *Alice*: Open-ended questions in the questionnaire

Nielsen’s (1994) ten heuristics, as mentioned in Section 4.5.2.1 and Section 5.5.2.2, provided a basis for the formulation of the first ten items in the questionnaire. Consequently, the usability of *Alice* was evaluated according to this classic set of usability principles. Moreover, these heuristics formed a foundation for the researcher to use in categorising the qualitative responses to questions obtaining information on the usability of *Alice*. The criterion number in the first column of Table 6.15 is cross-referenced to Table 4.3 as was done in Table 5.13 and Table 5.14.

Table 6.15 indicates positive learning experiences relating to the usability of *Alice* and categorised into sub-themes. Seven of the ten Nielsen’s heuristic characteristics emerged from the thematic analysis.

Table 6.15 Spontaneous positive responses on usability of *Alice*

#	Nielsen’s Heuristics	The heuristic as it applies to <i>Alice</i>	Frequency counts (n=55)	%
1	Visibility of the system status	Immediate feedback after program execution	2	4
		<i>Alice</i> is interactive	1	2
2	Match between the system and the real world	Graphics and animation	13	24
		Dealing with real objects brings coding into reality	6	11
		Visualisation, e.g. see creation of methods, see objects move on command	9	16
		3-Dimensionality: visual effects reflect real-world scenarios	2	4
3	User control and freedom	Learning how to create movies/storytelling	2	4
		Learning how to develop video games	2	4

6	Recognition rather than recall	Drag-and-drop feature limits typing	9	16
		Releases learner to focus on problem-solving	2	4
		No complex syntax, only English-like statements	2	4
7	Flexibility and efficiency of use	User-defined methods are created to manipulate objects and can be tested individually	4	7
8	Aesthetic and minimalist design	<i>Alice</i> interface is simple and easy to use	16	29
9	Recognition, diagnosis and recovery from errors	Can recover from errors quickly and easily	1	2
Total			71	-

In relation to *visibility of the system status*, two experimental participants indicated that the visual feedback improved their understanding of programming concepts. Regarding the *match between the system and the real world*, a theme emerged in that 24% described their enjoyment of using *Alice*'s graphics and animation. Moreover, 16% were pleased to see objects move on command due to the methods they had created. Four percent enjoyed the functionality to develop their own video games and movies, in line with the heuristic *user control and freedom*. With regard to *recognition rather than recall*, 16% appreciated not having to spend time on tedious typing, whilst two participants (4%) expressed relief about not having to remember details of syntax and semantics. Four participants (7%) appreciated the ability to create user-defined methods to manipulate objects, which contributed to the *flexibility and efficiency of using Alice*. Regarding *aesthetic and minimalist design*, 29% explicitly stated that *Alice*'s interface is easy to use. One participant (2%) indicated that he was able to *recover from errors* quickly and easily. Relative to the greater number of responses pertaining to other heuristics, the fact that this was mentioned by only one participant indicates that support for error recovery is not a major usability feature of *Alice*.

6.5.2.3 Spontaneous negative responses on usability of *Alice*: Open-ended questions in the questionnaire

The criterion number in the first column of Table 6.16 is cross-referenced to Table 4.3 for the purposes of validity and completeness. The responses in Table 6.16 are participants' spontaneous negative perceptions regarding the usability of *Alice*. Once again, they are categorised against related heuristics of Nielsen (1994) and further categorised into sub-themes.

Table 6.16 Spontaneous negative responses on usability of *Alice*

	Nielsen's Heuristics	The heuristic as it applies to <i>Alice</i>	Frequency counts (n=55)	%
2	Match between the system and the real world	Poor quality of graphics	1	2
4	Consistency and standards	Process of saving is complex in relation to other software	1	2
		<i>Alice</i> takes too long to load	2	4
		Installation of <i>Alice</i> software program on laptop – slows processor speed rapidly	1	2
		Transferring a program from a 32-bit OS on one computer to a 64-bit OS on another computer	2	4
6	Recognition rather than recall	Difficulty with the logic of setting statements for step-by-step execution	1	2
		Difficulty with the use of looping statements	1	2
		Difficulty with the use of do in order and do together statements	1	2
		Difficulty with the use of Math functions, e.g. avoiding the collision of objects	4	7
		Insufficient built-in methods to manipulate objects	1	2
		Difficulty with the use of methods/functions	6	11
		Difficulty with the use of variables/parameters	2	4

7	Flexibility and efficiency of use	Time consuming – use of the camera to turn the point of view/zoom to a particular object, e.g. set to a new scene; following an object such as a flying bird	12	22
		Rotating an object or repositioning an object	3	5
		Use of quad-view	3	5
		Copying statements when duplication of code required	1	2
8	Aesthetic and minimalist design	User interface cannot be customised, e.g. restricted world templates and objects; non-adjustable windows	1	2
		Objects allowed to move out of range from execution window	1	2
		User interface cluttered	2	4
9	Recognition, diagnosis and recovery from errors	<i>Alice</i> stalls/crashes intermittently	2	4
		Time consuming – execution of lengthy projects that have errors at the end	1	2
		Sound delay during execution of program	1	2
		Time consuming - no recovery from errors, e.g. ‘Console error’ upon deletion of code; re-adding objects that ‘disappeared’	2	4
10	Help and documentation	Insufficient literature and tutorials available on <i>Alice</i>	1	2
Total			53	-

Characteristics relating to seven of the ten interface design heuristics emerged from the negative responses. When analysing negative experiences pertaining to the usability of *Alice*, listed in Table 6.16, two of the seven heuristics were found to be more challenging than others. Regarding *recognition rather than recall*, for instance, 11% of experimental participants expressed difficulty with using methods and functions; 7% encountered difficulty with the use of Math functions; and 4% mentioned difficulty in using variables and parameters. Participants conveyed negative

sentiments regarding the *flexibility and efficiency of use* of *Alice*. A theme observed from 22% of the participants revealed that using the camera to turn the point of view or zoom to a particular object had been time consuming. A pattern that emerged for the remaining five of the seven heuristics was that there were very few negative responses falling within a range of 2% - 4%. These five heuristics were *match between the system and the real world; consistency and standards; aesthetic and minimalist design; recognition, diagnosis and recovery from errors; and help and documentation*.

6.5.2.4 Challenges faced by learners in learning OOP

The unprompted, unsolicited feedback of the experimental participants regarding the challenges that face them when learning OOP, is shown in Table 6.17. This question referred to learning OOP in general, not via *Alice*.

Table 6.17 Challenges faced by learners in learning OOP

Challenges of learning OOP	Frequency counts (n=55)	%
Inheritance	6	11
Methods (calling/overriding/abstract/virtual/no default/event-driven)	7	13
Functions (calling)	2	4
Properties (get/set)	2	4
Parameter passing	2	4
Creating classes and sub-classes (abstract)	3	5
Creating objects and instantiating objects	11	20
SQL Server and database connectivity	1	2
Poor understanding of every line of code, resulting in regurgitation	3	5
Difficulty with problem-solving and applying concepts, e.g. when to write code for objects and methods	14	25
Logically connecting theoretical concepts of OOP with practical examples	2	4
Lack of visual representation to aid with understanding logic	1	2
Time consuming – practicing concepts	2	4
Lack of motivation, for reasons such as boredom, intimidation or frustration	3	5
Lack of resources to facilitate learning, such as time, money, computers, open labs	1	2
Having to remember syntax	4	7
Understanding compiler errors and debugging	3	5
Language barrier	1	2
Difficulty with self-learning	1	2
Total	69	-

A notable 25% of the sample expressed difficulties in their lack of competence to solve problems and apply programming concepts. Of all the challenges highlighted by the participants of Case Study 1 (29%) and Case Study 2 (25%), the greatest degree of commonality existed within this specific criterion. Furthermore, 20% claimed to have a poor understanding of instantiation (i.e. creating an instance of an object). This is an unsatisfactory situation, particularly in view of the fact that, unwittingly, learners have been practicing instantiation since their first year at DUT. Thirteen percent of the experimental participants had experienced difficulty in understanding the logic of methods. Moreover, a theme emerged from 11% concerning their difficulties with inheritance. Furthermore, 7% stated that having to remember syntax was a challenge. Apart from these more common challenges, a few indicated challenges that were general in nature or that related directly to feature of OOP, such as classes and sub-classes; the inability to understand every line of code; lack of motivation; lack of competence to debug programs; and so on.

6.5.2.5 Techniques to improve the teaching of OOP

In order to address the challenges identified in Section 6.5.2.4, the participants were required to suggest techniques that would help alleviate these issues. Although participants in Case Study 2 reinforced the suggestions made in Case Study 1, they also proposed further techniques, as is noted by comparing Table 6.18 with Table 5.16.

Table 6.18 Techniques to improve teaching of OOP

Techniques to improve teaching of OOP	Frequency counts (n=55)	%
Detailed, interactive explanations of programs with practical examples that enforce the understanding of theoretical concepts, instead of just giving solutions and/or notes	14	25
Pace of lecturing could be less rapid, i.e. avoid moving ahead with new concepts before learners have had enough time to grasp other concepts	10	18
Guiding the learner through the program logic with step-by-step instructions	4	7
Using visual, graphical, interactive environments as part of standard teaching, to improve learner interest and motivation to learn OOP	15	27
Introduce <i>Alice</i> as a supplementary teaching tool	5	9
Providing solutions to examples	2	4
More examples during lecture time	5	9
Preference to work independently on a new problem before getting solutions from lecturer	1	2
Providing partially-coded examples would reduce typing time	1	2

Increasing level of complexity from introductory programs to advanced, in preparation for tests that are more complex	2	4
Student support systems and resources for IT learners, such as more open lab time, additional tutorials with examples, references and tutors to assist learners one-on-one	3	5
Offering 'Logic' as a first-year semester 1 subject/module, before being introduced to a programming language to help learners improve their logical thinking	1	2
Introducing OOP earlier in the ND: IT programme	1	2
Total	64	-

Having nearly doubled the frequency percentage from Case Study 1 (14%), a pattern emerged from 27% of the participants in Case Study 2, who suggested that a visual, graphical environment, such as *Alice*, be used to replace conventional tools or to supplement the conventional tools used in the teaching of OOP. Participants believed that this would improve their interest and motivation to learn OOP. The positive experiences gained from using the *Alice* VPE led them to make such assertions. With a 29% frequency response from Study 1, there was a similar response from 25% of the experimental participants in Study 2, suggesting that lecturers should provide explanations of programs, supplemented with practical examples that enforce the understanding of theoretical concepts. Participants were not keen on merely being given solutions to problems without discussions thereof. Eighteen percent requested that the pace of lecturing be slowed down to afford learners more time to grasp new concepts.

6.5.2.6 Impact of *Alice* on improving understanding of OOP

Table 6.19 presents the spontaneous responses of the experimental participants regarding the impact of *Alice* on improving their personal understanding of OOP. As indicated by the three pink shaded rows in Table 6.19, there were defined differences in responses to this question. This is similar to the situation in Case Study 1, although the factors given on *Alice*'s impact differ somewhat between the two studies. Many participants (64%) agreed that working with *Alice* had improved their understanding of OOP. Conversely, some participants (16%) did not agree that *Alice* had a positive impact on improving their understanding of OOP. Others (13%) were not entirely convinced of the effectiveness of the tool in this regard, stating that it had improved their understanding of OOP only to a certain extent. Discussion follows according to these three definitive groups of responses, giving the sub-themes in each category.

Table 6.19 Impact of *Alice* on improving understanding of OOP

Impact of <i>Alice</i> on improving understanding of OOP	Frequency counts (n=55)	%
Yes, working with <i>Alice</i> has improved my understanding of OOP	35	64
I have an improved understanding of OOP concepts such as methods, functions, events, inheritance, properties, parameters, classes, objects, instantiation and polymorphism	19	35
<i>Alice</i> helped me understand OOP visually	7	13
This is better than previous teaching of programming, which has been purely textual in nature	1	2
I can create methods to manipulate and animate objects to perform actions	6	11
Everything in <i>Alice</i> is viewed as an object	6	11
I have a clearer understanding that objects have different parts. It is easier to work with parts of an object	1	2
Reality of objects in <i>Alice</i> helps to understand objects in conventional languages	1	2
This environment increased my motivation and confidence to practice examples in OOP	1	2
Working with <i>Alice</i> has improved my understanding of OOP only to a certain extent	7	13
Cannot relate problem solving in conventional languages with the coding behind <i>Alice</i> . It would be more useful if the coding in <i>Alice</i> was shown	2	4
A two week workshop is insufficient time to see all the benefits of <i>Alice</i>	2	4
<i>Alice</i> would have helped me understand OOP if introduced earlier in the semester/for a new learner	2	4
<i>Alice</i> provided extra means of revision to reinforce concepts on OOP, in a fun way	4	7
No, working with <i>Alice</i> has not improved my understanding of OOP	9	16
OOP is more complex and uses more semantics in conventional languages than can be represented with <i>Alice</i>	2	4
I already have a sound understanding of OOP	5	9
<i>Alice</i> does not relate to second year work	1	2
Total	111	-

Thirty-five of the 55 experimental participants agreed that working with *Alice* had improved their understanding of OOP, representing 64% of the participants. In this response group, 35% indicated that *Alice* enhanced their grasp of specific concepts within the OOP domain, such as methods, functions, events, inheritance, properties, parameters, classes, objects, instantiation and polymorphism. Thirteen percent suggested that this improvement can be attributed to the visual nature of *Alice*. Eleven percent were impressed at how everything in *Alice* is viewed as an object; and emphasised their new ability to create methods that manipulate and animate these objects.

Seven of the 55 participants (13%) partially agreed that *Alice* had improved their understanding of OOP. Four participants (7%) indicated that their interaction with the VPE had been a suitable means of revision on OOP concepts. Moreover, two (4%) found that they could not see the relationship between problem solving in conventional languages and the coding behind *Alice*, due to the coding in *Alice* being hidden. Four percent indicated that a two week workshop had been insufficient to see all the benefits of using *Alice*. Furthermore, two (4%) stated that the *Alice* intervention should have been introduced earlier in the semester in order to gain a better understanding of OOP.

Conversely, nine of the 55 (16%) believed that working with *Alice* had not improved their understanding of OOP. Five expressed having a sound grasp of OOP concepts prior to using *Alice*, thereby indicating that *Alice* had not served to improve his or her understanding. Four percent stated that *Alice* does not represent the complexity and semantics of writing in traditional languages. Another participant had difficulty relating the concepts taught using *Alice* with the concepts covered in the DS2 syllabus.

6.5.2.7 Impact of *Alice* in addressing challenges of OOP

Feedback from the experimental participants concerning the impact of *Alice* in addressing the challenges of OOP, is presented in Table 6.20. Most of the participants (87%) agreed that *Alice* as a VPE can address these challenges, whilst a few (7%) believed that *Alice* could not help to solve the issues. Discussion follows according to these two definitive groups of responses, giving the sub-themes in each category. Considerably more sub-themes emerged from Case Study 2 than from Case Study 1 (see Table 5.18).

Table 6.20 Impact of *Alice* in addressing challenges of OOP

Impact of <i>Alice</i> in addressing challenges of OOP	Frequency counts (n=55)	%
Yes, I agree that <i>Alice</i> as a VPE can help address some of these challenges	48	87
It is easier to learn programming through visualisation/graphics, than having to remember the syntax for coding	13	24
Ability to see the visual effects of every statement of code	6	11
Use of trial-and-error to alter the desired output	1	2
Drag-and-drop feature releases the learner from having to write code and complex syntax	3	5
<i>Alice</i> is fun, engaging and cultivates an interest in programming	6	11
Immediate feedback when program runs	1	2

Don't have to deal with complex error messages	1	2
Coding is simple and straight-forward	3	5
Easy to quickly solve problems encountered	2	4
A visual representation of how objects interact with each other is valuable	5	9
Tutorials help with the coding	1	2
An interactive environment, that represents real-life situations	5	9
Promotes repeated revision of programming concepts	1	2
Improves learner motivation, e.g. with boredom	2	4
It makes programming concepts easy to learn and understand	24	44
No, I do not agree that <i>Alice</i> helps address these challenges	4	7
Preference to learn more advanced concepts such as databases, abstract methods/classes	2	4
Drag-and-drop feature does not assist with the hard-coding required in programming languages	3	5
Total	131	-

A high percentage (87%), namely 48 of the 55 participants, agreed that *Alice's* VPE helped them address some of the challenges they had faced in learning OOP. Furthermore, 24 participants (44%) indicated that *Alice* cultivates an easy approach to learning and understanding programming concepts. Thirteen (24%) felt that programming through visualisation alleviates the learner from having to remember lines of code. Eleven percent appreciated the ability to see the effects of every statement of code, i.e. immediate feedback. Furthermore, eleven percent found the *Alice* environment to be engaging and fun, whilst stimulating a greater interest in programming.

Conversely, four of the 55 participants (7%) doubted that *Alice* had helped to address these challenges. Three (5%) stated that the drag-and-drop feature does not equip the learner with the necessary hard-coding skills that are required in conventional languages. Two (4%) indicated that they would prefer to learn more advanced concepts, such as databases, abstract methods and abstract classes.

6.6 Qualitative data analysis: Interviews

Case Study 2 supplemented the qualitative findings derived from the open-ended responses to the questionnaire with further qualitative results extracted from semi-structured interviews. Interviews were not conducted in Case Study 1. In this way, richer participant responses have been obtained and validity has been enhanced. Furthermore, the data sources have been triangulated (see Section 4.3.2.5). As mentioned in Section 4.3.3.2, the researcher conducted post-questionnaire interviews

with eighteen of the 55 participants who had engaged with the *Alice* visual environment. These interviews were conducted individually with each participant three weeks after the workshop.

The interview protocol (see Appendix C.4) consisted of five open-ended questions on the main themes/domain areas. These five core questions are listed in Table 6.21 and will be referred to during the discussion that follows. The interviews followed a semi-structured format in that the core questions led to the exploration of avenues mentioned by different interviewees.

Table 6.21 Core questions in the interview protocol

Core questions
1. How easy is it to use <i>Alice</i> ? Why?
2. How would you describe your experience with the teaching and learning of programming at DUT? Tell me more.
3. What are the challenges you are faced with in learning OOP?
4. How would you like to see the teaching of OOP improve?
5. How has <i>Alice</i> impacted on your understanding of OOP and addressing the challenges you mentioned?

As was the case with the open-ended section of the questionnaire, participants were encouraged to discuss their positive and negative perceptions of *Alice* during the interviews. This led to discussions on the challenges they face in learning OOP; while suggesting techniques that could improve the teaching of OOP. Further, the impact of *Alice* on the learners' understanding of OOP was addressed, as well as the effectiveness of the *Alice* VPE in helping to solve the identified challenges. For the purposes of uniformity, the questions asked in the open-ended section of the questionnaire and the questions asked in the interviews were very similar. As a means to elicit deeper and more contextualised responses than those acquired from the questionnaire, the interview included a question pertaining specifically to their experience with the teaching and learning of programming at DUT. Many of the interview responses provided similar types of data as the open-ended questions in the questionnaire, therefore the same structure has been used to analyse them.

The main area of discussion in this section is highlighted in Figure 6.3. Section 6.6.1 explains data collection and preparation, followed by data analysis and interpretation in Section 6.6.2.

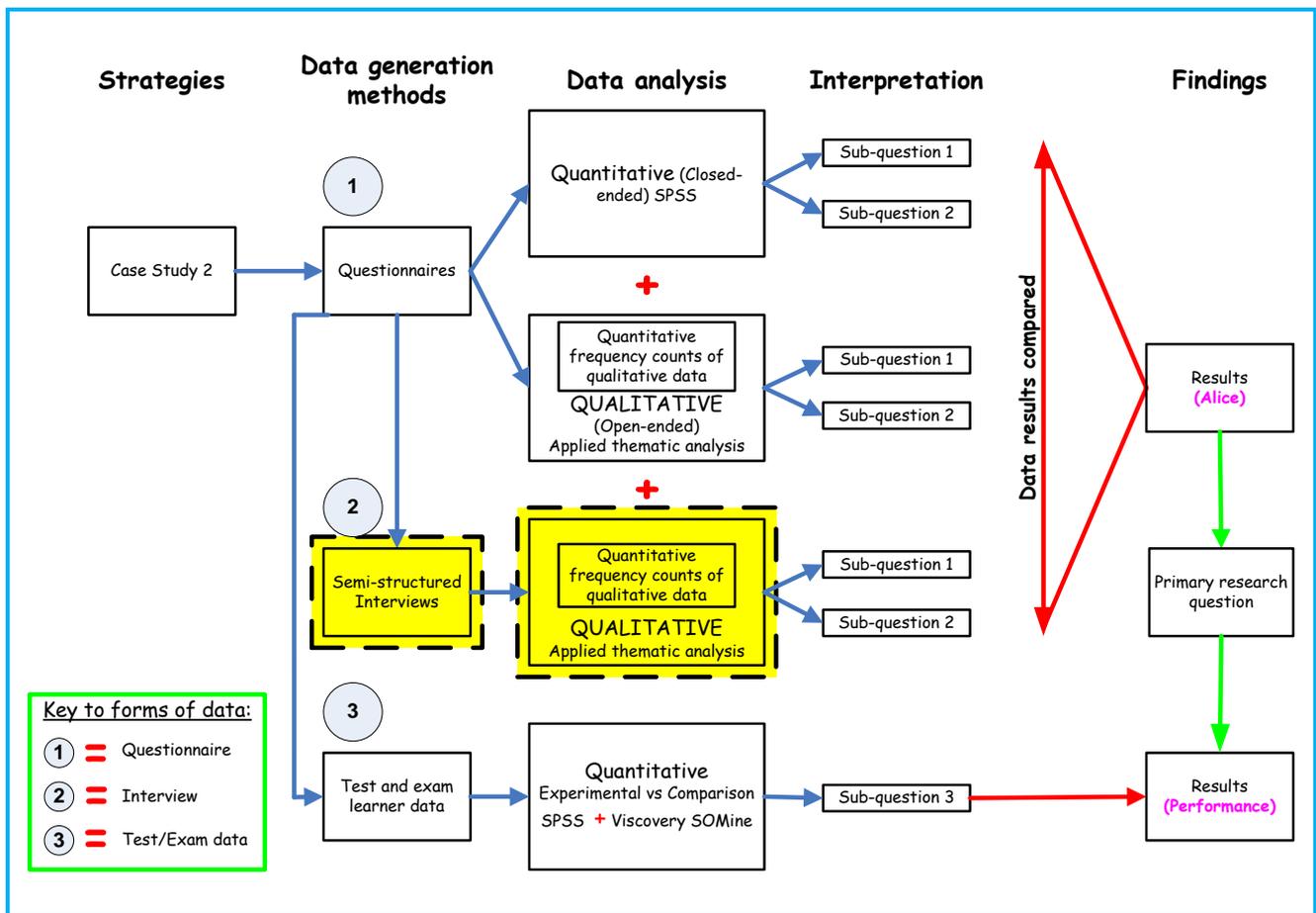


Figure 6.3 Detailed research processes of Case Study 2, highlighting a quantification of qualitative data derived from the responses to the semi-structured interviews

6.6.1 Data collection and preparation: Interviews

As stated previously, the semi-structured interviews consisted of five open-ended questions on the core themes. This led into open discussions, as each interviewee was probed further with more focused questions as the interview progressed and interesting matters were raised. The interviewees were free to speak in detail on the matters included, and also introduced issues of their own that were relevant to the theme at hand. The interviews elicited rich, unanticipated data. Contrary to the written format of the open-ended questions in the questionnaire, the semi-structured interviews were conducted verbally and tape recorded. The researcher transcribed the recorded interviews into transcripts, which were then analysed using ATA. Furthermore, a quantification of the qualitative data derived from the interview transcripts provided statistical findings relating to the textual data. Appendix E.1 depicts the frequency counts of participant responses to the various themes.

6.6.2 Data analysis and interpretation: Interviews

Manual thematic analysis was conducted in the process of studying responses to each interview question, in a quest to establish themes and relationships. During this process, valid and relevant data was transcribed into Appendix E: Qualitative Analysis Tables (the Codebook, as described in Section 4.8.2.3). Once the codebook had been updated, substantive corroborative interpretations were extrapolated from the codebook to reach viable and pertinent conclusions. This will be addressed in Sections 6.6.2.1 to 6.6.2.8, whilst drawing comparisons in each sub-section to the qualitative findings from the open-ended section of the questionnaire (see Section 6.5.2). It must be noted that references to the questionnaire within these sections are relevant to the open-ended questions only. The same structure was used to analyse the interview responses, as was done for the open-ended questions in the questionnaire.

The set of tables on interview findings presented in Sections 6.6.2.1 to 6.6.2.8 are similar to the set of tables already given in Sections 6.5.2.1 to 6.5.2.7, which report the Study 2 responses to the qualitative open-ended questions in the questionnaire. The two sets of tables present different categories, each with sub-themes that emerged from qualitative analysis. Some of the sub-themes are common, i.e. they emerged from both the qualitative responses to the questionnaire and from the interview transcripts, while other sub-themes emerged from either the questionnaire or from the interview transcripts. Thus, the raw categories and themes in the two sets of tables respectively, are not identical. Refer to the codebook in Appendix E.1 for the full details.

Each of Tables 6.13 to 6.20 on qualitative responses to the questionnaire survey, is followed by textual discussion on points in the tables. However, in presenting Tables 6.22 to 6.28 on qualitative interview responses, detailed textual discussion of each point is avoided. Only the most notable points are addressed, as well as aspects where there are different sub-themes or differences between the findings of the two qualitative studies.

6.6.2.1 General overview comments: Interviews

Personal perceptions and general attitudes emerged from interview responses. Table 6.22 shows positive feedback to the first interview question in Table 6.21, “How easy is it to use Alice? Why?”. The general overview comments presented in Table 6.22 did not fall within any of Nielsen’s ten heuristics.

Table 6.22 Positive overview comments on the ease of using *Alice* - Interviews

Positive overview comments	Frequency counts (n=18)	%
I like <i>Alice</i>	3	17
<i>Alice</i> is a fun/exciting/enjoyable/interesting way to learn programming	4	22
Improved understanding of DS2 course work	2	11
Suitable for novice programmers, e.g. high school learners or first-year learners starting OOP	7	39
Promotes learner collaboration	4	22
Promotes self-learning; anticipated future personal use	6	33
It is easier to visualise coding practices in <i>Alice</i> when compared to conventional programs	2	11
Using <i>Alice</i> was easy because of previous knowledge gained from first year studies	1	6
Prior exposure to <i>Alice</i> gave me a slight advantage	1	6
Total	30	-

Whilst a few participants in the questionnaire survey had expressed negative comments regarding their general perceptions on the ease of using *Alice* (see Table 6.14 under ‘General overview comments’), by contrast, none of the interviewees reported any negative observations. All general overview comments made by the interviewees were therefore positive. A small number of participants (three) spontaneously made the comment ‘I like *Alice*’, compared to the prompted 46 responses in the questionnaire, but Table 6.23 indicates that the interviewees almost unanimously found the interface simple and easy to use (see Section 6.6.2.2). With a greater number assertion than the questionnaire, seven (versus three) felt that *Alice* is a suitable tool for novice programmers. This is probably due to the fact that the discussion ethos of an interview gave opportunities to dwell on less obvious matters. Moreover, in other human-related issues, four interviewees (opposed to one in the questionnaire) mentioned the value of learning collaboratively with *Alice* and six participants (in comparison to three) suggested that using *Alice* improves motivation to self-learn and generates interest in further personal use of the VPE. Themes arose that had not emerged in the questionnaire: two interviewees appreciated the power of the visualisations found in *Alice*, which allows human beings to understand concepts better than when using conventional programs. Two others felt that they had found *Alice* easy to use because of their backgrounds – either their programming foundation from first-year studies or prior experience with the VPE.

6.6.2.2 Spontaneous positive responses on usability of *Alice*: Interviews

As mentioned, Nielsen’s (1994) heuristics provided a basis for structuring responses to the questionnaire (see Sections 4.5.2.1, 5.5.2.2 and 6.5.2.2). Similarly, these heuristics were applied in categorising responses to the first interview question “How easy is it to use *Alice*? Why?” (Table 6.21).

Table 6.23 relates to positive learning experiences pertaining to the usability of *Alice*. Themes corresponding to nine of the ten heuristics emerged from the interviews.

Table 6.23 Spontaneous positive responses on usability of *Alice* - Interviews

#	Nielsen’s Heuristics	The heuristic as it applies to <i>Alice</i>	Frequency counts (n=18)	%
1	Visibility of the system status	Immediate feedback after program execution	2	11
2	Match between the system and the real world	Graphics and animation	1	6
		Visualisation, e.g. see creation of methods, see objects move on command	3	17
3	User control and freedom	Learning how to develop video games	2	11
5	Error prevention	Restricting the user to a space during tutorials prevents the occurrence of errors	1	6
6	Recognition rather than recall	Drag-and-drop feature limits typing	10	56
		No complex syntax, only English-like statements	2	11
7	Flexibility and efficiency of use	User-defined methods are created to manipulate objects and can be tested individually	1	6
8	Aesthetic and minimalist design	<i>Alice</i> interface is simple and easy to use	17	94
		Pre-defined methods and variables simplify coding in <i>Alice</i>	7	39
		Dummy-camera helps to recover from loss of focus in the <i>Alice</i> world	1	6
9	Recognition, diagnosis and recovery from errors	Can recover from errors quickly and easily	4	22

10	Help and documentation	Built-in tutorials assist with learning the basic techniques of programming; help teach how to use <i>Alice</i> ; and provide an overview of <i>Alice</i>	3	17
		The built-in demo programs and additional Internet resources assisted in answering my queries	2	11
Total			56	-

Fourteen subthemes occurred in both Table 6.15 (questionnaire responses) and Table 6.23 (interview responses), of which nine were common to both. The number of additional sub-themes mentioned in both the questionnaire and interview were the same, but the response from each data extraction method differed with respect to participant perspectives and thought processes. For example, seven interview participants expressed appreciation of *Alice's* inbuilt features, such as the pre-defined methods and variables, which support coding. Three interviewees liked the built-in tutorials, and two others found the demos and supplementary Internet resources to be useful. The questionnaire elicited sub-themes that related to other aspects, for instance, six participants identified the relationship between the reality of *Alice* objects and real-world objects. Two others enjoyed learning how to create movies through storytelling.

With regard to common sub-themes, most response patterns were fairly similar. For example, a notably high number of interviewees (seventeen) commended *Alice's* simplicity and ease of use, compared to sixteen from the questionnaire. Similarly, ten interviewees and nine questionnaire participants praised the drag-and-drop feature, which reduces tedious typing.

Some common sub-themes relating to the *match between the system and the real world* (Heuristic 2) were more evident in the questionnaire. Thirteen questionnaire responses (versus one) showed a keen interest in graphics and animation, while nine participants (opposed to three interviewees) acknowledged the benefits of visualisation and being able to see the creation of methods and objects moving on command.

6.6.2.3 Spontaneous negative responses on usability of *Alice*: Interviews

Table 6.24 depicts negative perceptions regarding the usability of *Alice* in response to the first interview question “How easy is it to use *Alice*? Why?” (see Table 6.21). Once again, the responses are categorised against related heuristics of Nielsen (1994).

Table 6.24 Spontaneous negative responses on usability of *Alice* - Interviews

#	Nielsen’s Heuristics	The heuristic as it applies to <i>Alice</i>	Frequency counts (n=18)	%
4	Consistency and standards	<i>Alice</i> takes too long to load	1	6
		Transferring a program from a 32-bit OS on one computer to a 64-bit OS on another computer	1	6
6	Recognition rather than recall	Difficulty with the use of Math functions, e.g. avoiding the collision of objects	1	6
		Difficulty with the use of inheritance	1	6
7	Flexibility and efficiency of use	Time consuming – use of the camera to turn the point of view/zoom to a particular object, e.g. set to a new scene; following an object such as a flying bird	4	22
		Rotating an object or repositioning an object	2	11
		Copying statements when duplication of code required	1	6
		Time consuming – movement of objects, e.g. moving single limbs of bodies, so as to propel the object across the screen	2	11
8	Aesthetic and minimalist design	The look and feel of the <i>Alice</i> GUI was juvenile	1	6
9	Recognition, diagnosis and recovery from errors	<i>Alice</i> stalls/crashes intermittently	1	6
		Time consuming – execution of lengthy projects that have errors at the end	1	6
10	Help and documentation	Tutorials are time consuming to complete	1	6
		Tutorials are not as effective in teaching <i>Alice</i> when compared to being taught by the lecturer	2	11
Total			19	-

Thirteen sub-themes emanated from the interview study (see Table 6.24), whereas 24 emerged from the questionnaire survey (see Table 6.16). This ratio is realistic considering that there were only eighteen interviewees in comparison to fifty-five questionnaire participants. Furthermore, it is likely that questionnaire respondents producing a written document felt freer to express negative opinions than interviewees who were in the presence of a lecturer and may have felt reluctant about being candid.

In terms of numbers, a higher response was elicited from the questionnaires with regards to common findings in the two qualitative studies. Nineteen negative perceptions emerged from the open-ended questions in the questionnaires in relation to *flexibility and efficiency of use* (Heuristic 7), while nine such interview responses emerged. However in terms of the percentages, 50% of interviewees were negative in this regard, compared to 35% in the questionnaires. Heuristic 7 was the most dominant heuristic in both qualitative studies and included notable common sub-themes: Four interviewees (and twelve questionnaire participants) indicated that using the camera to turn the point of view or zoom to a particular object had proven to be time consuming. Similarly, two interviewees (and three in the questionnaire) experienced difficulties with rotating or repositioning objects. Other findings revealed that one interviewee (in comparison to four in the questionnaire) struggled with using Math functions to perform basic movements of objects. This interview response was notable, since the participant felt free to discuss this problem personally and openly with the researcher. It was notable that these issues were still fresh in the minds of the interviewees, even although three weeks had elapsed since the *Alice* workshop.

Five additional sub-themes were mentioned during the interviews. Although the number of responses for each was not high, they are nonetheless new and interesting findings. Two interviewees found that it takes time learning how to move and propel objects across the screen, for example, watching a girl walk along a path with her hands and legs moving proportionally and in-sync with each other. The participants must be commended for attempting such advanced animations. Two others appreciated having the researcher present to guide them through a set of lessons, in comparison to self-studying the tutorials, whilst another interview participant felt that it was time consuming to learn from the tutorials. An interviewee suggested that the look and feel of the *Alice* GUI was ‘juvenile’ in comparison to conventional applications. When he was developing *Alice* programs at home, his parents confused the application with a game and were surprised to hear he was learning programming.

6.6.2.4 Challenges faced by learners in learning OOP: Interviews

Interviewees' opinions regarding challenges they experienced when learning OOP, are shown in Table 6.25, in response to the third interview question "What are the challenges you are faced with in learning OOP?" (see Table 6.21).

Table 6.25 Challenges faced by learners in learning OOP - Interviews

Challenges of learning OOP	Frequency counts (n=18)	%
Inheritance	3	17
Methods (calling/overriding/abstract/virtual/no default/event-driven)	3	17
Properties (get/set)	2	11
Parameter passing	1	6
Creating classes and sub-classes (abstract)	4	22
Creating objects and instantiating objects	6	33
SQL Server and database connectivity	1	6
Difficulty with problem-solving and applying concepts, e.g. when to write code for objects and methods	3	17
Logically connecting theoretical concepts of OOP with practical examples	1	6
Lack of visual representation to aid with understanding logic	2	11
Lack of motivation, for reasons such as boredom, intimidation or frustration	2	11
Having to remember syntax	2	11
Understanding compiler errors and debugging	1	6
Language barrier	3	17
OOP is a new concept that has not been dealt with during first year. The objects-first strategy had therefore not been implemented	6	33
Difficulty understanding OOP the first time	4	22
Adjusting from practical to theory-based testing	1	6
Total	45	-

Fourteen sub-themes were common to the questionnaire responses (see Table 6.17) and interview responses (see Table 6.25). In most cases, more responses emanated from the questionnaires regarding common challenges, including: a lack of competence in problem solving and in applying programming concepts; a poor understanding of instantiation; and difficulties with the concept of inheritance and with understanding the logic of methods.

Some of the common challenges received a greater interviewee response, such as difficulties in creating classes and sub-classes and low competencies in dealing with objects. Participants who

were not first-language English speakers were frustrated by issues related to the language barrier. These problems weighed heavily on the minds of the interview participants, and were spontaneously communicated to the researcher. The severity of these challenges was notable. This confirms the worth of personal interaction in eliciting information.

Three additional sub-themes emerged from the interviews, compared with sub-themes in the questionnaire survey. For example, an insightful concern emanated from six interviewees, all of whom had a preference for an objects-first strategy, rather than being lectured OOP for the first time during second year. Furthermore, four interview participants struggled with learning and understanding OOP at first attempt. Another participant could not adjust from practical testing methods to theory-based written papers.

6.6.2.5 Techniques to improve the teaching of OOP: Interviews

In order to address the challenges identified in Section 6.6.2.4, interviewees were required in the fourth interview question (see Table 6.21) to suggest how they would like to see the teaching of OOP improve. Their ideas are presented in Table 6.26.

Table 6.26 Techniques to improve teaching of OOP - Interviews

Techniques to improve teaching of OOP	Frequency counts (n=18)	%
Detailed, interactive explanations of programs with practical examples that enforce the understanding of theoretical concepts, instead of just giving solutions and/or notes	5	28
Pace of lecturing could be less rapid, i.e. avoid moving ahead with new concepts before learners have had enough time to grasp other concepts	2	11
Guiding the learner through the program logic with step-by-step instructions	3	17
Using visual, graphical, interactive environments as part of standard teaching, to improve learner interest and motivation to learn OOP	5	28
Introduce <i>Alice</i> as a supplementary teaching tool	17	94
More examples during lecture time	1	6
Student support systems and resources for IT learners, such as more open lab time, additional tutorials with examples, references and tutors to assist learners one-on-one	2	11

Offering 'Logic' as a first-year semester 1 subject/module, before being introduced to a programming language to help learners improve their logical thinking	1	6
Introducing OOP earlier in the ND: IT programme	3	17
Providing learners with written exercises prior to tests will help learners to grasp the style and format of the test	1	6
Including more fun gaming applications into the syllabus, apart from business applications such as sales, inventory etc.	1	6
Preference for practical lectures in a lab over the theory lectures in a classroom	2	11
Total	43	-

In the questionnaire survey (see Table 6.18), only five participants out of 55, suggested that *Alice* be introduced as a supplementary teaching tool within the ND: IT programme. In a far greater affirmation, an almost unanimous pattern emerged, whereby seventeen of the eighteen interviewees requested for *Alice* to be an official part of ND: IT (see Table 6.26). The interviews elicited new information that did not emerge from the questionnaire. For example, eight participants felt that this intervention would be most appropriate during the second semester of first year; while six suggested the first semester of first year. One participant felt it would be most beneficial during second year; and another proposed that the vacation period after Matric is a suitable time for preparatory classes. In general, it was clear that interviewees preferred that OOP be introduced earlier in the ND: IT programme.

Other sub-themes emerged from the interviews when compared with sub-themes in the questionnaire survey: Two participants preferred lab lectures, with practical hands-on experience, instead of theoretical classes. One interviewee enjoyed the *Alice* experience so much, that she suggested including fun gaming applications into the DS2 syllabus, as well as conventional business applications. Another interviewee requested that written exercises be provided prior to writing tests, in order to familiarise learners with the style and format of theory questions.

6.6.2.6 Impact of *Alice* on improving understanding of OOP: Interviews

Table 6.27 presents spontaneous responses of the interviewees regarding the impact of *Alice* on their personal understanding of OOP. This was in response to the fifth interview question in Table 6.21, "How has *Alice* impacted on your understanding of OOP and addressing the challenges you mentioned?". As indicated by the three pink shaded rows in Table 6.27 and the set of responses following each pink row, responses fell into three distinct response groups. For instance, nearly all the participants reported that working with *Alice* had improved their understanding of OOP. None

of them indicated that *Alice* had failed to make a positive impact on improving their understanding. Others were not entirely convinced and believed that *Alice* had improved their understanding to a certain extent. Discussion follows according to these three definitive response groups.

Table 6.27 Impact of *Alice* on improving understanding of OOP - Interviews

Impact of <i>Alice</i> on improving understanding of OOP	Frequency counts (n=18)	%
Yes, working with <i>Alice</i> has improved my understanding of OOP	16	89
I have an improved understanding of OOP concepts such as methods, functions, events, inheritance, properties, parameters, classes, objects, instantiation and polymorphism	16	89
<i>Alice</i> helped me understand OOP visually	8	44
Everything in <i>Alice</i> is viewed as an object	2	11
I have a clearer understanding that objects have different parts. It is easier to work with parts of an object	1	6
Reality of objects in <i>Alice</i> helps to understand objects in conventional languages	3	17
This environment increased my motivation and confidence to practice examples in OOP	3	17
This environment improved my understanding of basic concepts such as loops and selection statements	1	6
I refer to examples in <i>Alice</i> to reinforce the understanding of concepts in conventional languages	2	11
Working with <i>Alice</i> has improved my understanding of OOP only to a certain extent	1	6
Cannot relate problem solving in conventional languages with the coding behind <i>Alice</i> . It would be more useful if the coding in <i>Alice</i> was shown	2	11
<i>Alice</i> would have helped me understand OOP if introduced earlier in the semester/for a new learner	1	6
Learners followed the examples given by the instructor without applying their minds to solving new problems. Thus, a solid grasp of OOP concepts cannot be made.	1	6
The pre-defined objects in <i>Alice</i> does not assist with creating new objects	1	6
No, working with <i>Alice</i> has not improved my understanding of OOP	-	-
OOP is more complex and uses more semantics in conventional languages than can be represented with <i>Alice</i>	1	6
Total	59	-

Working with Alice had improved my understanding of OOP

An almost unanimous number (16 interviewees out of 18) indicated that *Alice* enhanced their grasp of specific concepts within the OOP domain, such as methods, functions, events, inheritance, properties, parameters, classes, objects, instantiation and polymorphism (see Table 6.27). Although

a greater number of questionnaire participants (nineteen) provided the same response (see Table 6.19), they comprised only 35% of the fifty-five participants, whereas the sixteen interviewees made up 89% of the eighteen participants. Furthermore, eight interviewees (versus seven in the questionnaire) suggested that such an improvement can be accredited to the visual nature of *Alice*. Three interview participants (in comparison to one questionnaire participant) suggested that the reality of objects in *Alice* improves the understanding of objects in conventional languages; and increased their motivation and confidence to practice examples in OOP.

Four additional sub-themes provided new points of discussion. Two interviewees indicated that they refer to examples done in *Alice* to reinforce their understanding of concepts in conventional languages. Another participant felt that the VPE improved his understanding of basic concepts of programming, such as repetition and selection structures.

Alice had improved my understanding to a certain extent

The other two additional sub-themes were raised by those interviewees who believed that *Alice* had improved their understanding only to a certain extent. One suggested that learners tend to follow the examples without applying their minds to solving new problems. Thus, a solid grasp of OOP concepts is difficult to achieve. Another interviewee indicated that learners are restricted in regards to creating new objects, as well as the pre-defined objects cannot be edited.

6.6.2.7 Impact of *Alice* in addressing challenges of OOP: Interviews

Feedback from the interviewees concerning the impact of *Alice* in addressing the challenges of OOP, is presented in Table 6.28. This was in response to the fifth interview question “How has *Alice* impacted on your understanding of OOP and addressing the challenges you mentioned?” (see Table 6.21). When asked whether the *Alice* VPE helped to address the identified challenges, the interviewees responded by mentioning ways in which it had supported them. In contrast, only one mentioned a reservation.

Table 6.28 Impact of *Alice* in addressing challenges of OOP - Interviews

Impact of <i>Alice</i> on improving understanding of OOP	Frequency counts (n=18)	%
<i>Alice</i> as a VPE can help address some of these challenges		
It is easier to learn programming through visualisation/graphics, than having to remember the syntax for coding	2	11
Ability to see the visual effects of every statement of code	2	11
Use of trial-and-error to alter the desired output	1	6
Drag-and-drop feature releases the learner from having to write code and complex syntax	3	17
<i>Alice</i> is fun, engaging and cultivates an interest in programming	7	39
Coding is simple and straight-forward	4	22
A visual representation of how objects interact with each other is valuable	2	11
Improves learner motivation, e.g. with boredom	1	6
It makes programming concepts easy to learn and understand	10	56
No, I do not agree that <i>Alice</i> helps address these challenges		
Drag-and-drop feature does not assist with the hard-coding required in programming languages	1	6
Total	33	-

Alice as a VPE can help address some of these challenges

Ten common findings occurred in the questionnaire (see Table 6.20) and interviews (see Table 6.28). With 24 similar responses in the questionnaire survey, a good response was derived from ten interviewees, indicating that *Alice* facilitates an easy approach to learning and understanding programming concepts. With regard to both methods of qualitative research, the same absolute number of participants (six in the questionnaire and seven in the interviews) found the *Alice* environment to be engaging and fun, whilst cultivating an interest in programming. Similarly, four interviewees (as opposed to three questionnaire participants) enjoyed the simplicity of coding with *Alice*.

I do not agree that Alice helps address these challenges

One interviewee expressed a justifiable concern, namely that the drag-and-drop feature does not equip the learner with the necessary hard-coding skills that are required in conventional languages.

6.6.2.8 Teaching and learning programming at DUT: Interviews

As mentioned previously, to contextualise the research and as a means of eliciting deeper and different responses from those acquired from the questionnaire, the interview contained an additional question pertaining to the participants' specific experiences with the teaching and learning of programming at DUT, i.e. the second interview question "How would you describe your experience with the teaching and learning of programming at DUT? Tell me more." (see Table 6.21). This section discusses the participants' general attitudes, as well as the positive and negative comments in their responses to this question. Special attention was given to the detail of each comment, in terms of the aspect it relates to within DUT. The researcher categorised the comments in each category with respect to the following criteria:

- (a) *Lecturer*;
- (b) *Learning*;
- (c) *Syllabus*;
- (d) *Resources*; and
- (e) *Outcomes*.

General overview comments

Table 6.29 presents the personal perceptions and general attitudes that emerged from interview responses regarding the teaching and learning of programming at DUT.

Table 6.29 General overview comments on the teaching and learning of programming at DUT

Teaching and learning theme	As it applies to DUT	Frequency counts (n=18)	%
Lecturer	Teaching and learning depends on how a lecturer explains a concept in class	5	28
Learning	I am able to learn closer to the test dates	1	6
Syllabus	The structured approach in the first year should be replaced with an objects-first approach	1	6
	Conventional languages are more difficult to understand than programs such as <i>Alice</i>	1	6
Resources	The step-by-step .pdf notes provided during the <i>Alice</i> workshop proved useful for self-study	2	11
Total		10	-

Concerning the lecturer, five interviewees believed that the success of their learning depends on the lecturer's approach to teaching. With regard to resources, two participants found that the detailed notes providing step-by-step instructions, such as those used during the *Alice* workshop, are useful in aiding self-study.

Spontaneous positive responses regarding the teaching and learning of programming at DUT

Table 6.30 relates to positive experiences pertaining to the teaching and learning of programming at DUT.

Table 6.30 Spontaneous positive responses on the teaching and learning of programming at DUT

Teaching and learning theme	As it applies to DUT	Frequency counts (n=18)	%
Lecturer	I am satisfied with the lecturer's teaching styles	12	67
	The lecturers are helpful when you need assistance	6	33
	Lecturers ensure that learners do their work and motivate them to study	2	11
Learning	I spend time practicing programming exercises	13	72
	I am motivated to learn programming	13	72
	Programming is like Maths. The more you practice, the better you know it	2	11
Syllabus	The syllabus is well structured and adequate, so learners should be able to cope if they pace themselves	3	17
	The syllabus is relevant and current to industry requirements	1	6
	The ND: IT course offers a high practical component, which helps me to cope	1	6

Resources	DUT provides learners with extensive resources and information for self-study	1	6
	The tutors were able to assist me with problems	1	6
	I have my own computer facilities	8	44
	I am able to get assistance from fellow students, who I believe are good learners	5	28
	I had prior exposure to C++ Java in school, and this helped me to adapt to C# programming language at DUT	2	11
	I had prior exposure to a computer in school; and this helped me to develop basic computer skills	1	6
	The computer lab facilities are adequate for teaching	1	6
Outcomes	I had prior experience with programming in school, but have an improved understanding of IT since studying at DUT	1	6
	I have an improved understanding of concepts from first year to second year	3	17
Total		76	-

In general the interviews elicited responses that praised the teaching of programming at DUT and that indicate an approach that supports learning.

Lecturer

Twelve interviewees expressed satisfaction with lecturers' teaching styles, of which nine of the twelve were general perceptions and three of the twelve specified that these were first year lecturers only. Furthermore, a third of the participants stated that the lecturers are willing to assist when required. Positive remarks were made by two participants regarding the lecturer's concern over learners completing their study tasks; and the encouragement that lecturers provide.

Learning

Thirteen participants said they spent time practicing programming exercises and were motivated to learn programming. Moreover, two of them likened programming expertise to the skills required to be a good Mathematician, which improves with practice. This affirms the assertion made by Law *et al.* (2010), which posits that in order to develop good programming skills, learners should do a great deal of intensive practice on programming exercises to gain experience in debugging. Furthermore, the learner must be adequately motivated to sustain a high level of competence.

Syllabus

According to three interviewees, the current syllabus within the ND: IT programme at DUT, is well structured and adequate. Learners should therefore be able to cope with the workload if they pace themselves. Furthermore, one participant expressed satisfaction regarding the relevance of the syllabus to industry requirements. Another was comfortable with the high practical component of the course.

It must be noted, however, that these positive comments regarding the syllabus came from only a few interviewees.

Resources

At any institution, the availability of computer resources for programming studies is a key driver for successful learning competency. Eight of the eighteen stated that they possessed their own computer facilities. Five participants found peer-to-peer collaboration to be a good learning resource. Two believed that their prior exposure to programming at school had helped them adjust to learning at DUT.

Outcomes

Three participants acknowledged that their understanding of programming concepts had improved from first year to second year. One stated that, although previously exposed to programming at school, his level of understanding in programming had improved since studying at DUT.

Spontaneous negative responses regarding the teaching and learning of programming at DUT

Table 6.31 depicts negative perceptions regarding the teaching and learning of programming at DUT.

Table 6.31 Spontaneous negative responses on the teaching and learning of programming at DUT

Teaching and learning theme	As it applies to DUT	Frequency counts (n=18)	%
Lecturer	Lecturers do not provide a clear explanation of concepts, leaving the learner to do independent research or seek assistance from third year learners	6	33
	Lecturers want to move ahead with the syllabus before learners have an adequate grasp of the concepts	1	6
	Lecture time is insufficient to grasp the work the lecturer is doing	1	6
	Some lecturers come to class with coded programs, which makes it difficult to follow if you get distracted	2	11
	I prefer to go through the program step by step with the lecturer, instead of being lectured to	2	11
	Due to a poor understanding in class I spend more time trying to grasp a concept than time on practicing programming	1	6
	Some learners are intimidated to ask questions in class	2	11
	I depend on the lecturer for a clear explanation of concepts. Thereafter, I am able to self-learn with notes and example programs	4	22
	There was a lack of interaction and consultation with the lecturer, which left learners uncertain about the correctness of their solutions in tests and exams	1	6
	The DS2 lecturer expected us to know the DS1 syllabus well. New concepts of the syllabus were introduced quickly and students were expected to keep up	2	11
	Preference for a higher practical component during tests and exams	1	6

Learning	I feel demotivated when I cannot solve a problem and there is no immediate feedback	2	11
	I feel demotivated when I cannot understand a concept	1	6
	I focused on syntax and semantics with no understanding of the underlying concepts	1	6
	I had no exposure to computer programming at school and wanted to deregister during first year. I am still finding it difficult and confusing	1	6
	I felt scared and intimidated when I started studying programming, but these fears were dispelled with time	4	22
	I had no exposure to a computer prior to studying at DUT, and found that developing basic computer skills was challenging	1	6
	It was difficult to learn debugging and hard to write the syntax for a calculation	1	6
Resources	There are not enough computers in the open labs	2	11
	Labs are often closed before the scheduled time	1	6
Total		37	-

There are less negative issues than positive factors, i.e. 37 negative responses in Table 6.31, compared to 76 positive ones in Table 6.30, which is encouraging. However, the points raised in Table 6.31 are important and present pertinent concerns that need to be addressed.

Lecturer

Six interviewees voiced concerns regarding the quality of facilitation, which forced them to seek assistance from other sources, such as personal research and third year learners. On the other hand, four indicated their dependence on the lecturers for a clear explanation of concepts, after which they were able to self-learn with notes and example programs.

Learning

It appears that over time, four interviewees gained more confidence and comfort in their programming studies. As mentioned previously, one of the challenges facing learners engaging in computer programming is a lack of immediate feedback after the program has executed. Two participants explicitly mentioned this problem.

6.6.3 Wrap-up discussion on the interviews

The interviews, by their very nature, enriched the findings from the previous qualitative study. Participants were further probed and prompted so as to explore unanticipated sub-themes. The responses elicited during a face-to-face interaction are spontaneous and unsolicited, rather than the contemplation that occurs during a written exercise. No spontaneous, unprompted comment can be discounted in a qualitative study. Therefore, a low number of responses was not regarded as a sign of lack of importance. The converse however, does not hold true: a large number of spontaneous, unprompted responses would necessarily indicate an important issue.

The added value of the interviews was two-fold. Firstly, the interview responses served to confirm the open-ended responses from the questionnaire. This was presented in Sections 6.6.2.1 to 6.6.2.7 as common findings that emanated across both qualitative studies. Secondly, additional sub-themes emerged from the interviews, which did not feature in the open-ended questionnaire responses. These findings brought new and interesting topics to the forefront. The researcher was able to probe the interviewees further with follow-up questions to the core questions, thus information gathered was drilled down to a more detailed level.

Moreover, the additional second interview question “How would you describe your experience with the teaching and learning of programming at DUT? Tell me more.” (see Table 6.21) added enrichment to the study. Participants were able to express personal reflective thoughts about their learning experiences at DUT, as discussed in Section 6.6.2.8.

6.7 Quantitative data analysis: Test and exam learner data

This final section on analysing the data of Case Study 2, presents the quantitative analysis of test marks and exam results, comparing performances of the experimental group and the comparison group. As mentioned in Section 4.4, the researcher hand-selected learners based on their level of success at first-year level, to achieve close uniformity between the comparison group and the experimental group. The component being addressed in this section is highlighted in Figure 6.4.

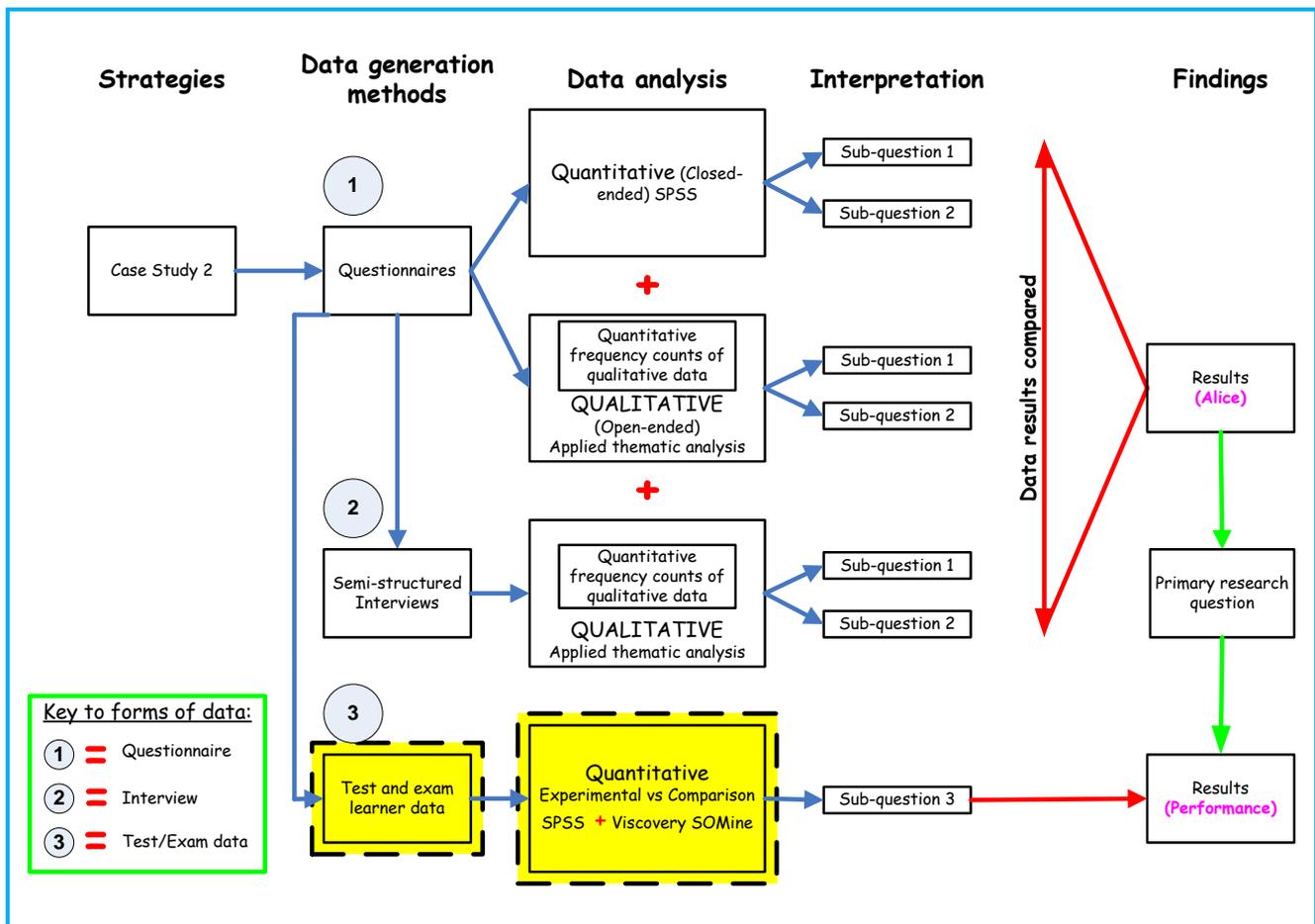


Figure 6.4 Detailed research processes of Case Study 2, highlighting quantitative data analysis of learner data from the tests and exam

6.7.1 Data collection and preparation: Test and exam learner data

As mentioned in Section 6.4.1, the first data set of Case Study 2, contained responses to closed-ended questions in the questionnaire. A second set of quantitative data sets were generated after the experimental group's exposure to learning OOP with an *Alice* intervention. This involved extracting academic performance data from assessments of the experimental group and the comparison group. In line with the method of data extraction mentioned in Section 5.6.1, the 2012 cohort's test and

exam data for both groups was mined from DUT’s ITS database. This database contains, but is not limited to, records of all registered learners at DUT, including their personal details and academic record. Ethical clearance was granted to the researcher from the DUT for the use of sensitive data (see Appendix A.1).

The raw data extracted from the ITS database was originally in text format (.txt files). For the purpose of quantitative data analysis, the assessment data of Study 2 underwent data preparation to ensure that the resulting formatted data sets were free from any data impurities. This was done in line with the processes used on assessment data in Case Study 1. Cleansed data was then imported into a Microsoft Excel spreadsheet that is compatible with SPSS as well as Viscovery SOMine. Considering that both these statistical packages require the data to be in a numerical format, gender and race were converted to numeric formats using the coding pattern shown in the key for Table 6.32 and Table 6.33. Similarly, the subject result symbols (P*, P, F, FE, FR, FS, FA) were converted to numeric equivalents according to the coding pattern in Figure 6.5, which appears as the final column in Table 6.32 and Table 6.33.

Code	Result	Meaning
7	P*	PASS WITH DISTINCTION
6	P	PASS
5	F	FAIL
4	FE	FAIL, SPECIAL EXAM GRANTED
3	FR	FAIL, SUB MINIMUM GRANTED
2	FS	FAIL, SUPPLEMENTARY GRANTED
1	FA	FAIL, NO EXAM ADMITTANCE

Figure 6.5 Codes for results, listed against the associated description

Segments of the data sets for the experimental group and the comparison group are shown in Table 6.32 and Table 6.33 respectively. It must be noted that the final data sets are entirely numeric and anonymised, in that the student number in the ‘Participant’ column has been coded as P1, P2, ..., P50 in Table 6.32 and Q1, Q2, ..., Q50 in Table 6.33. The ‘DS101’ and ‘DS102’ columns give the participants’ first-year results for Development Software 1, Module 1 and Module 2 respectively, which served as prerequisites for DS2. The ‘Test 1’, ‘Test 2’, ‘Exam Mark’ and ‘Final Mark’ columns are performance assessments for Development Software 2. The ‘Result’ column provides each participant’s final result, of which the meaning is given in Figure 6.5. The average marks appear at the bottom of the data sets.

Table 6.32 Data set 1 - Experimental group

Participant	DEMOGRAPHIC DATA			FIRST-YEAR MARKS		SECOND-YEAR MARKS				
	Gender	Race	Age	DS101	DS102	Test1	Test2	Exam Mark	Final Mark	Result
P1	1	4	20	65	56	26	67	71	64	6
P2	1	3	20	79	80	70	61	87	80	7
P3	1	3	19	84	78	70	67	95	85	7
P4	1	4	22	79	81	65	73	83	79	7
P5	0	3	19	87	82	61	67	80	76	7
P6	1	4	36	67	57	50	52	69	65	6
P7	0	3	20	73	70	45	33	59	56	6
P8	1	3	20	64	75	65	45	59	59	6
P9	0	4	23	50	63	48	70	77	69	6
P10	0	4	18	55	60	47	67	58	59	6
P11	1	4	19	55	76	61	77	93	84	7
P12	0	4	20	58	62	45	66	83	75	7
P13	1	4	20	54	59	39	53	77	65	6
P14	1	4	21	56	73	34	37	72	61	6
P15	1	1	21	77	75	75	60	85	79	7
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
P46	1	3	19	84	87	84	70	84	82	7
P47	1	3	20	95	89	80	79	81	81	7
P48	1	3	21	62	64	70	64	90	82	7
P49	1	4	21	65	70	53	70	82	75	7
P50	1	4	19	61	64	73	56	89	76	7
AVERAGES				67.26	69.66	55.94	62.88	77.00	71.24	6.44

Key:

Gender: M = 1, F = 0

Race: Black = 4, Asian = 3, Coloured = 2, White = 1

Result: Pass with distinction = 7, Pass = 6, Fail = 5, Fail, special exam granted = 4,
Fail, sub minimum granted = 3, Fail, supplementary granted = 2, Fail, no exam admittance = 1

As mentioned in Section 4.4, in Case Study 2 the experimental learners chosen to participate (P1, ..., P50 in Table 6.32) were selected from those who had passed both programming subjects (Development Software 1 and Technical Programming 1) together with one of the theoretical subjects (Information Systems 1 or Systems Software 1) at first attempt. The selection of learners for the comparison group (Q1, ..., Q50 in Table 6.33), had a similar success rate at first-year level, and were manually selected from the remaining learners.

Table 6.33 Data set 2 - Comparison group

Participant	DEMOGRAPHIC DATA			FIRST-YEAR MARKS		SECOND-YEAR MARKS				
	Gender	Race	Age	DS101	DS102	Test1	Test2	Exam Mark	Final Mark	Result
Q1	0	2	20	71	68	44	61	61	61	6
Q2	1	3	20	80	87	66	61	84	75	7
Q3	1	3	26	83	83	86	99	100	96	7
Q4	1	1	20	63	68	33	37	36	41	5
Q5	1	4	20	64	61	40	44	68	57	6
Q6	1	3	18	63	69	46	37	58	56	6
Q7	1	3	25	90	86	76	73	67	70	6
Q8	1	4	26	77	67	41	69	24	38	4
Q9	1	4	21	63	53	16	56	64	57	6
Q10	1	3	21	57	70	59	59	85	77	7
Q11	1	4	20	50	52	13	34			1
Q12	1	4	19	55	70	53	67	74	67	6
Q13	0	3	20	83	82	55	77	94	84	7
Q14	1	3	20	50	50	21	51	41	41	5
Q15	1	4	22			76	99	86	85	7
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Q46	0	3	19	75	67	54	66	84	77	7
Q47	1	3	21	62	54	33	50	76	64	6
Q48	1	3	21	63	72	34	60	69	63	6
Q49	1	1	22			61	36	71	65	6
Q50	1	4	19	51	64	29	54	74	60	6
AVERAGES				67.44	66.38	49.08	59.51	71.31	66.15	6.10

Key:

Gender: M = 1, F = 0

Race: Black = 4, Asian = 3, Coloured = 2, White = 1

Result: Pass with distinction = 7, Pass = 6, Fail = 5, Fail, special exam granted = 4,
Fail, sub minimum granted = 3, Fail, supplementary granted = 2, Fail, no exam admittance = 1

The test and exam data in Table 6.32 and Table 6.33 will be used to compare the performance of the experimental group with the comparison group. This is addressed in the next section.

6.7.2 Data analysis and interpretation: Test and exam learner data

As stated, this section compares the performance of the experimental group with the performance of the comparison group. Further comparison must be conducted, namely comparing results of data analysis of the test and examination data in Case Study 1 and in Case Study 2. This is done in Section 7.4.3 of the Conclusion Chapter. In doing so, the impact of the *Alice* intervention on learner performance will be concluded.

6.7.2.1 SPSS data analysis of test and exam learner data

As mentioned in Section 4.8.1.1 and Section 5.6.2.1, SPSS is a statistical analysis tool used for data analysis of the test and exam data of the 2012 cohort. This section presents the findings in the form of tabulated reports, charts, plots of distributions and trends, including a *two-sample comparison*, *frequency plot*, *hypothesis testing* and *independent t-test*.

Two-sample comparison

A two-sample comparison was performed with Sample 1 being the comparison group and Sample 2 being the experimental group. Table 6.34 shows summary statistics for the two samples of data, based on the final mark.

Table 6.34 Summary statistics

	Comparison Group (Sample 1)	Experimental Group (Sample 2)
Count	48	50
Average	66.1458	71.24
Standard deviation	12.8262	8.8215
Coefficient of variation	19.3907%	12.3828%
Minimum	38.0	52.0
Maximum	96.0	86.0
Range	58.0	34.0
Standard skewness	-0.77026	-0.184346
Standard kurtosis	0.0893796	-1.45595

Sample 1 contained 48 values ranging from 38.0 to 96.0 with an average of 66.15%, whilst Sample 2 contained 50 values ranging from 52.0 to 86.0 with an average of 71.24%. Within the original comparison group data set of 50 records, two records were excluded due to missing summative

assessments in cases where certain assessments were not done by two learners. This resulted in a count of 48 for Sample 1. Furthermore, in order to attain a uniform and equitable comparison between Sample 1 and Sample 2, the original experimental group data set of 55 records, was reduced by five records. This resulted in a count of 50 for Sample 2. The minimum mark of the 48 learners in the comparison group was 38% and four learners failed. In the experimental group, by contrast, no participants failed and the minimum mark was 52%.

Other tabular options within this analysis can be used to test whether differences between the statistics from the two samples are statistically significant. This matter is reported later in this section. Of particular interest in Case Study 2 are the standardised skewness and standardised kurtosis, which can be used to determine whether the samples come from normal distributions. Values of these statistics outside the range of -2 to +2 indicate significant departures from normality, which would tend to invalidate the tests which compare the standard deviations. In this case, both standardised skewness values are within the expected range and both standardised kurtosis values are within the expected range. The coefficient of variation also indicates that there is a smaller variation about the mean for the experimental group.

Frequency plot

Figure 6.6 visually depicts the frequency range of the final marks for the comparison group and experimental group.

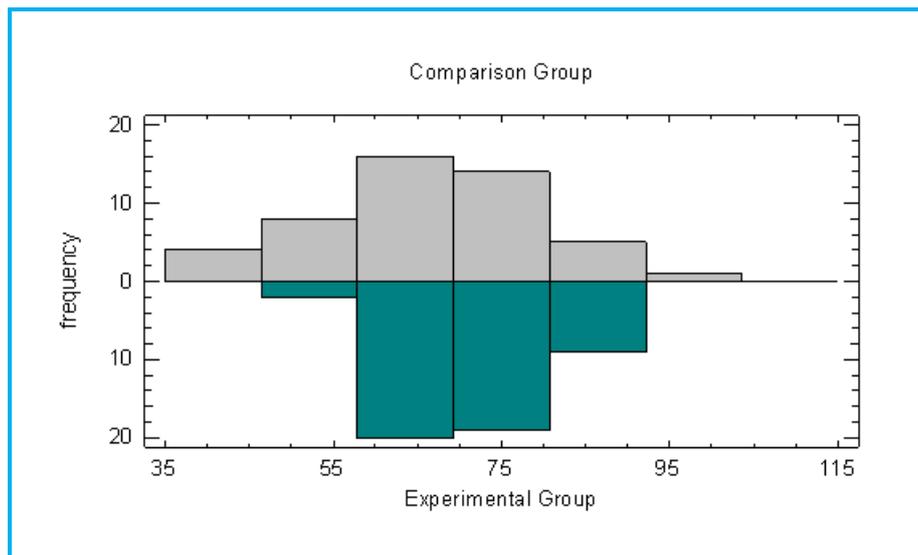


Figure 6.6 Frequency plot for the experimental group vs. the comparison group

With regard to the length of bars, as indicated by the height (comparison group) and depth (experiential group), the length of each bar represents the number of learners who obtained a mark in the sub-range of that bar. Bars for the comparison group and the experimental group are juxtaposed against each other for better visual comparison.

- Range of marks: The marks of the comparison group span a greater range than the experimental group, as can be seen from the left-to-right range of the grey bars.
- Clustering of marks: The longer green bars indicate that the experimental learners' final marks are clustered closer and that most fall within the 57 to 82 range.

Hypothesis testing

As mentioned in Section 4.8.1.1, one of the forms of analysis available in SPSS is inferential statistical analysis, which is concerned with the testing of hypotheses and allows the researcher to draw conclusions about populations from sample data (Lind *et al.*, 2002). Inferential statistical analysis was applied to the final marks to compare the performance of learners from the experimental group with those from the comparison group. The independent t-test was used to compare the mean values between the two groups.

The result of the t-test for the comparison of the means is given below:

Null hypothesis: difference = 0

t statistic = -2.28

Two-sided p-value = 0.0251

The null hypothesis claims that there is no significant difference in the mean values between the experimental and comparison groups. The alternate hypothesis states that the difference between the mean values between the groups is significant. Since the p-value for the t-test is less than 0.05, it implies that there is indeed a significant difference between the mean values. The direction of the difference is given by the mean scores (with respect to the standard deviation), implying that the experimental group had a higher mean than the control group.

Independent t-test

Table 6.35 presents a test of differences between the experimental and comparison groups for each of the scoring assessments. The intervention of *Alice* is indicated by the red border prior to Test 2.

Table 6.35 t-test for equality of means, mean scores and standard deviation for the experimental group and the comparison group

Assessment	Sig. (2-tailed)	Group	N	Mean	Std. Deviation	Std. Error Mean
DS101	0.942	Comparison	48	67.4375	12.73779	1.83854
		Experimental	50	67.2600	11.37633	1.60886
DS102	0.121	Comparison	48	66.3750	12.01351	1.73400
		Experimental	50	69.6600	8.56097	1.21070
Test 1	0.039	Comparison	49	49.0816	17.86971	2.55282
		Experimental	50	55.9400	14.55939	2.05901
Test 2	0.258	Comparison	49	59.5102	15.98974	2.28425
		Experimental	50	62.8800	13.37320	1.89126
Exam Mark	0.043	Comparison	48	71.3125	16.12736	2.32778
		Experimental	50	77.0000	10.88905	1.53994
Final Mark	0.024	Comparison	48	66.1458	12.82615	1.85130
		Experimental	50	71.2400	8.82150	1.24755

Although, in general, the performance of the experimental group was better than that of the comparison group, the mean of the experimental group was better than that of the comparison group in all instances, after the *Alice* intervention (indicated by the red border). This is indicated in the ‘Mean’ column for the assessments conducted after exposure to *Alice*. In particular, the examination performance was six percent higher in the experimental group. This is indicated by bold red font.

6.7.2.2 Findings derived using Viscovery SOMine

As mentioned in Section 4.8.1.2 and Section 5.6.2.2, Viscovery SOMine is a high-potential data mining tool used in this study for the visual analysis and exploration of numerical data sets. The process of acquiring knowledge is facilitated by analysing cluster maps and component pictures, which were discussed in detail in Section 5.6.2.2. The discussion that follows in this section is aided by the use of component pictures to visually present the analysis findings of the 2012 assessment marks for the experimental group and the comparison group.

The use of Viscovery SOMine component pictures helps the human eye to identify visual patterns and trends, which according to Du (2010) can assist interpretation. The scale at the bottom of each component picture ranges from the minimum to the maximum node value of the respective component.

Figure 6.7 depicts the component pictures for student number, age, race and gender in the experimental group. With respect to the ‘Gender’ component picture, one can see nodes that represent the male learners in dark blue and those representing the female learners coloured red. The dominant number of male learners depicted in Viscovery SOMine corresponds to the high number of males in the male:female ratio derived from SPSS. Similarly, the red nodes in the ‘Race’ component picture represent black learners and those representing the white learners are dark blue, and so on. *Note:* The colours of nodes are generated by Viscovery SOMine itself.

Furthermore, the component pictures are directly related to each other. Overlaying them will allow one to visually judge, for example, the correspondence between the ‘Student Number’ component picture and all other components. There is a strong correlation between the red nodes in the ‘Race’ component picture, the dark blue nodes in the ‘Gender’ component picture and the dark/light blue nodes in the ‘Age’ component picture. This is indicated by the intersected area of the map on which these nodes are distributed. This observation reaffirms the statistics presented in Table 6.4, Table 6.5 and Table 6.6, which indicate that the highest number of learners in the gender-race, gender-age and race-age categories was a set of black males aged between nineteen and twenty-one years.

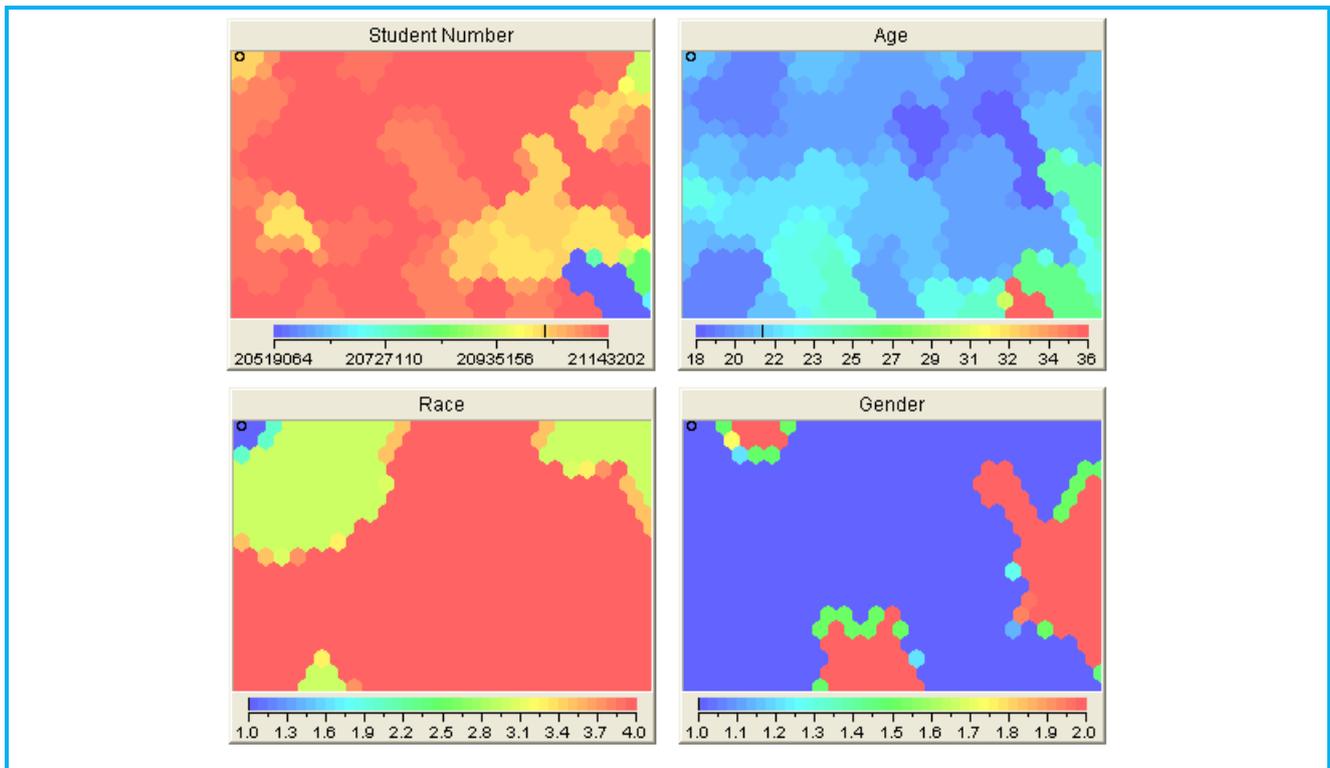


Figure 6.7 Component pictures for student number, age, race and gender in the experimental group

Key:

- Student number:** Ascending order of learner identification (blue to red nodes)
- Age:** Ascending order of age (blue to red nodes)
- Race:** Black = red nodes (4 on scale), Asian = green nodes (3 on scale),
Coloured = light blue nodes (2 on scale), White = dark blue nodes (1 on scale)
- Gender:** Male = dark blue nodes (1 on scale), Female = red nodes (2 on scale)

Figure 6.8 and Figure 6.9 depict the component pictures for ‘Exam Mark’, ‘Final Mark’ and ‘Result’ of the experimental group and comparison group respectively. The ‘Exam Mark’ and ‘Final Mark’ range from blue to red coloured nodes in ascending order of marks from 0 to 100%. There is a visual higher frequency of red, orange and yellow nodes in the component pictures pertaining to the experimental group, when compared to the comparison group. This indicates superior performance on the part of the experimental group.

Similarly, in relation to the ‘Result’ component picture of the experimental group (see Figure 6.8), learner results range from dark blue (6 on scale) to red (7 on scale) indicating that all learners who were exposed to the *Alice* intervention had either passed or passed with a distinction. On the other hand, the ‘Result’ component picture representing the comparison group (see Figure 6.9) did not reflect such a satisfactory performance. This can be observed with nodes ranging from learners that failed (1 to 5 on scale) to those who either passed or passed with a distinction (6 and 7 on scale).

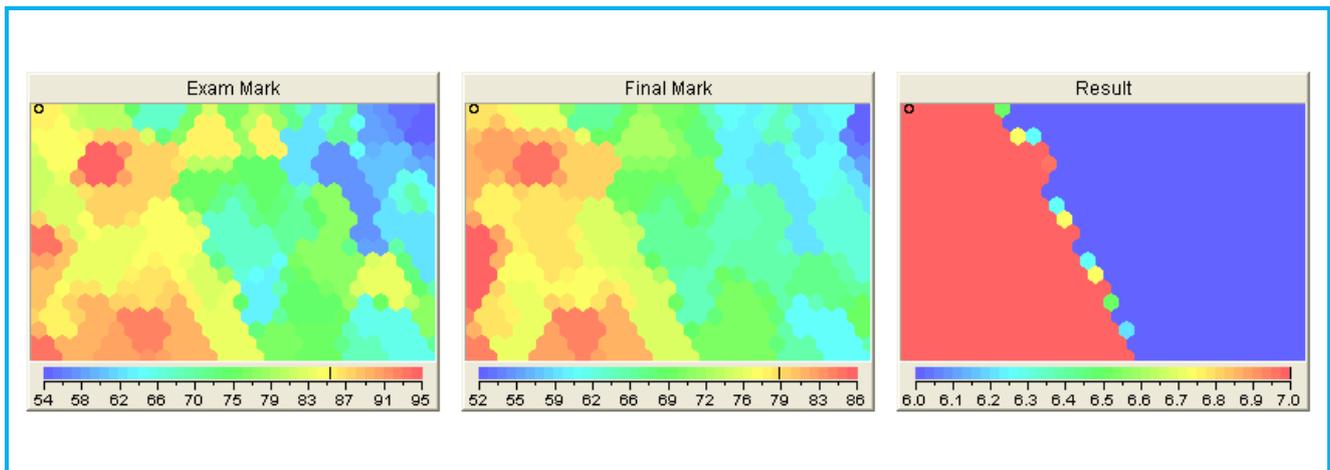


Figure 6.8 Experimental group – 2012

Key:

Exam mark: Ascending order of exam marks from 0 to 100% (blue to red nodes)

Final mark: Ascending order of final marks from 0 to 100% (blue to red nodes)

Result: Pass with distinction = red nodes (7 on scale),
Pass = dark blue nodes (6 on scale)

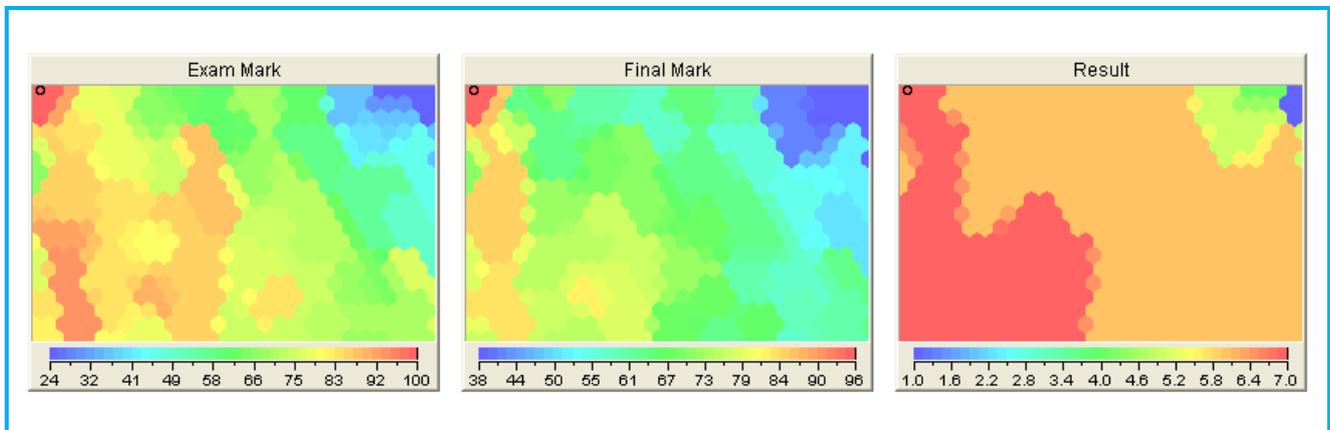


Figure 6.9 Comparison group – 2012

Key:

Exam mark: Ascending order of exam marks from 0 to 100% (blue to red nodes)

Final mark: Ascending order of final marks from 0 to 100% (blue to red nodes)

Result: Pass with distinction = red nodes (7 on scale), Pass = dark/light orange nodes (6 on scale),
Fail = light green nodes (5 on scale), Fail, special exam granted = dark green nodes (4 on scale),
Fail, sub minimum granted = light blue nodes (3 on scale),
Fail, supplementary granted = medium blue nodes (2 on scale),
Fail, no exam admittance = dark blue nodes (1 on scale)

6.8 Triangulation between the quantitative and qualitative findings in Case Study 2

As stated in Section 6.1, this section compares the quantitative results (closed-ended questions in the questionnaire) and two sets of qualitative findings (open-ended questions in the questionnaire and interviews) for Case Study 2.

The quantitative aspect comprises the 72 closed questions administered to the experimental group. The qualitative aspect of Study 2 comprised the six open questions in the questionnaire and the qualitative data that emerged from the five questions in the supplementary interviews. The quantitative data was analysed by studying the Likert scale ratings, while the qualitative was interpreted by using the codebook (see Appendix E.1) to extract themes and sub-themes from the learners' responses, adopting an applied thematic analysis approach (see Section 4.8.2).

In general, the findings of the two qualitative methods, namely the questionnaire and interviews, were similar. This was discussed in detail in Section 6.6. The emphasis in this section, therefore, is to study the relationship between the findings of the quantitative and qualitative research.

In a comparative study, one overviews the common findings and varying findings, which span across responses to both quantitative and qualitative methods. Thus, interpretations are presented in light of the study as a whole:

- (a) Commonalities arise where similar findings occur across two or three methods, whether positive or negative.
- (b) Varying findings occur when methods give contrasting results.

In the qualitative research, learners provided *spontaneous, unprompted* responses. Therefore, it is reasonable to expect that many aspects highlighted in the quantitative study were not mentioned in the qualitative studies. Each learner mentioned only those matters that, for them, were the most profound. In order to obtain some measure of distinction within the qualitative responses elicited from learners for the questionnaire and interviews, a threshold was computed to determine whether these qualitative responses were highly rated. This threshold was calculated by taking an average number of responses as a function of the total number of responses for all themes and sub-themes divided by the number of participants. There were 55 participants that responded to the questionnaire and 18 interviewees. Consequently, for the open-ended questions in the questionnaire, five (9%) or more spontaneous responses was regarded as a high rating of that aspect. By contrast,

for the interview questions, three (17%) or more spontaneous responses were viewed as a high rating.

The data analysis presented in this chapter was discussed in relation to the core domains used to formulate the questions in the questionnaire and interview instruments, namely:

- (a) Usability
- (b) Challenges faced by learners in learning OOP
- (c) Techniques to improve the teaching of OOP
- (d) Impact of *Alice* on improving understanding of OOP
- (e) Impact of *Alice* in addressing challenges of OOP
- (f) Teaching and learning programming at DUT.

Common and varying findings were observed within each of these domains, for at least two or all three methods. This process began by identifying the highest number of responses among all themes and sub-themes that exceeded the established threshold, and was conducted independently for both the open-ended responses to the questionnaire and interview responses. A comparison was drawn between these qualitative methods in search of similarities or differences. Thereafter, a further comparison was made against the quantitative findings according to the percentage ratings i.e. between one (Strongly Disagree) and 5 (Strongly Agree) on the Likert scale.

Discussions of the common findings and varying findings follow in Section 6.8.1 and Section 6.8.2.

6.8.1 Common findings

Table 6.36 presents the common findings between the quantitative and qualitative findings for Case Study 2, which is discussed in this section. The findings highlighted in turquoise exist across all three methods, while the findings highlighted in green are common to at least two methods. The findings highlighted in grey are those that reiterate and support the discussion of a finding.

Table 6.36 Common findings between quantitative and qualitative data for Case Study 2

Finding No.	Domain	Theme/Sub-theme	Quan	Qual	
			A + SA (%)	Open-ended Q's in Q'aire (Frequency spontaneous responses) Threshold=5 N=55	Interview Q's (Frequency spontaneous responses) Threshold=3 N=18
1	Usability	<i>Alice</i> is a fun/exciting /enjoyable/interesting way to learn programming	80% average for all heuristics (Section 6.4.2.1)	17 (Section 6.5.2.1)	4 (Section 6.6.2.1)
	<i>Alice</i> Impact	<i>Alice</i> is fun, engaging and cultivates an interest in programming	-	6 (Section 6.5.2.7)	7 (Section 6.6.2.7)
2	Usability	Match between the system and the real world	92.7% (Section 6.4.2.1)	30 (Section 6.5.2.2)	4 (Section 6.6.2.2)
3	Usability	Aesthetic and minimalist design	65.5% (Section 6.4.2.1)	16 (Section 6.5.2.2)	25 (Section 6.6.2.2)
4	Challenges	Creating objects and instantiating objects	69.1% (Section 6.4.2.3)	11 (Section 6.5.2.4)	6 (Section 6.6.2.4)
5	Challenges	The objects-first strategy had not been implemented during the first year of study	58.2% (Section 6.4.2.4)	1 (Section 6.5.2.4)	6 (Section 6.6.2.4)
6	Techniques	Using visual, graphical, interactive environments, such as <i>Alice</i> , to improve learner interest and motivation to learn OOP	91.4% (Section 6.4.2.4)	15 (Section 6.5.2.5)	5 (Section 6.6.2.5)
7	Techniques	Detailed, interactive explanations of programs with practical examples that enforce the understanding of theoretical concepts, instead of just giving solutions and/or notes	1 (Section 6.4.2.5)	14 (Section 6.5.2.5)	5 (Section 6.6.2.5)
	Teaching and learning at DUT (T&L@DUT)	Lecturers do not provide a clear explanation of concepts, leaving the learner to do independent research or seek assistance from third year learners	-	-	6 (Section 6.6.2.8)
8	Techniques	Introduce <i>Alice</i> as a supplementary teaching tool	92.8% (Section 6.4.2.5)	5 (Section 6.5.2.5)	17 (Section 6.6.2.5)

9	AliceImpact	I have an improved understanding of OOP concepts such as methods, functions, events, inheritance, properties, parameters, classes, objects, instantiation and polymorphism	81.8% (Section 6.4.2.5)	19 (Section 6.5.2.6)	16 (Section 6.6.2.6)
		Alice makes programming concepts easy to learn and understand	-	24 (Section 6.5.2.7)	10 (Section 6.6.2.7)
10	AliceImpact	Alice helped me understand OOP visually	94.6% (Section 6.4.2.4)	7 (Section 6.5.2.6)	8 (Section 6.6.2.6)
11	T&L@DUT	I am motivated to learn programming	96.4% (Section 6.4.2.3)	-	13 (Section 6.6.2.8)
	Challenges	Lack of motivation, for reasons such as boredom, intimidation or frustration	-	3 (Section 6.5.2.4)	2 (Section 6.6.2.4)
12	T&L@DUT	I spend time practicing programming exercises	60% (Section 6.4.2.3)	-	13 (Section 6.6.2.8)
	AliceImpact	Alice increased my motivation and confidence to practice examples in OOP	70.9% (Section 6.4.2.5)	1 (Section 6.5.2.6)	3 (Section 6.6.2.6)
13	T&L@DUT	I am able to get assistance from fellow students, who I believe are good learners	83.6% (Section 6.4.2.5)	-	5 (Section 6.6.2.8)

In four cases (findings 5, 11, 12 and 13) the interviews on their own produced qualitative findings that confirmed data from the quantitative study. This attests to the particular value of face-to-face interviews in obtaining interesting data. In no case did qualitative data from the open questions in the questionnaire confirm quantitative data on their own, although in thirteen cases they did so along with data from interviews.

The second research sub-question of this study relates to the learners' experience with using Alice. Some of the findings presented in this section, namely the first three common findings, address this research question.

Finding 1:

Seventeen spontaneous responses were elicited from the open-ended questions in the questionnaire, indicating that *Alice* is a fun, enjoyable, engaging and interesting tool used to learn programming. This was reiterated by four positive responses from the interviewees. Moreover, when questioned

about the impact of *Alice* on addressing the challenges of OOP, six responses from the questionnaire and seven responses from the interviewees emanated in relation to this theme. Confirming the responses to the qualitative methods, the quantitative findings regarding the usability of *Alice* indicate a high degree of conformance to Nielsen's ten heuristics. This claim is supported with a category rating of 4.00 – 4.55 for seven of the ten interface design heuristics, as shown in Table 6.8. Furthermore, the lowest-rated heuristic had a category rating of 3.56, which is still relatively high and the average percentage of (Agree + Strongly Agree) across all heuristics is 80%. It follows that, when learners experience user-friendly interaction and interfaces that are easy to use, as is the case with *Alice*, their enjoyment of programming can be enhanced.

Finding 2:

The open-ended questions in the questionnaire elicited 30 positive responses from the participants regarding the *match between the system and the real world*, ranking Heuristic 2 as the highest-rated heuristic. Four positive responses in relation to Heuristic 2 emerged from the interviewees. A notable observation is that none of the interviewees expressed negative concerns with respect to Heuristic 2 (Section 6.6.2.3), whilst only one participant mentioned a negative concern in the open-ended responses to the questionnaire (Section 6.5.2.3). In line with these positive findings, in the closed-ended questions in the questionnaire, the *match between the system and the real world* is also the highest-rated heuristic with a category rating of 4.55. This verifies that the *Alice* VPE helps learners to easily associate programming objects with real-world objects, such as rabbits, trees etc.

Finding 3:

Regarding *aesthetic and minimalist design* (Heuristic 8), sixteen positive responses from the open-ended questions in the questionnaire indicated that the *Alice* interface is simple and easy to use. This was the second highest-rated heuristic. In addition, the interviews elicited 25 positive responses that related to Heuristic 8. Twenty-five positive responses emerged from eighteen interviewees, because several interviewees mentioned more than one positive factor. In line with these positive qualitative findings, the closed-ended questionnaire responses reflected a combined agreement of 65.5%. These results indicate that the design of *Alice* has resulted in aesthetics that appeal to users.

The remaining common findings in this section address the first research sub-question, which investigates the effectiveness, as perceived by learners, of using the Alice visual programming environment in addressing the challenges facing novice programming learners within the object-oriented domain.

Finding 4:

In response to the open-ended questions in the questionnaire a notable eleven experimental participants claimed to have a poor understanding of creating and instantiating objects. Six interviewees affirmed the pertinence of this challenge. Similarly, a good percentage (69.1%) of participants confirmed in the closed-ended questions in the questionnaire, that learning the basic concepts of OOP would become easier, if the learner was alleviated from dealing with complex syntax. This implies that learners are experiencing difficulties with applying basic OOP concepts, such as creating objects and instantiating them.

Finding 5:

Having being taught OOP for the first time during second year, six interviewees expressed a preference for an objects-first strategy to be implemented at first-year level. It must be borne in mind that these qualitative responses were unprompted and unsolicited, thereby strongly validating the quantitative closed-ended questionnaire findings presented in Section 6.4.2.4. These quantitative findings revealed that a fair percentage (58.2%) of the participants felt confident that it would be easier to learn OOP during the first year of study, and later learn the conventional control structures such as loops, if statements etc. As mentioned previously, the implementation of OOP in DS2 occurred during the second semester of 2012 in the Department of Information Technology at DUT. This study shows that participants felt overwhelmed with the exposure to OOP concepts only during second year. The decision to introduce OOP into the first year syllabus was therefore taken to bridge the learning gap between first and second year.

Finding 6:

Participants expressed a desire to learn OOP through the use of a visual, graphical environment, such as *Alice*. They felt that such methods of teaching would increase their interest in the OOP domain and improve their motivation to learn the subject matter. By means of qualitative feedback, fifteen such responses were elicited from the open-ended questions in the questionnaire. Five interviewees confirmed these assertions. Common findings also emerged from the quantitative closed-ended questions, with a combined agreement of 91.4% in favour of using 3D visual tools such as *Alice* to improve learners' understanding of OOP.

Finding 7:

In relation to the teaching of OOP, a high number of participants (fourteen open-ended questionnaire responses and five interview responses) expressed a preference for detailed, interactive explanations of programs. Participants believed that such explanations, coupled with

practical examples, can enforce the understanding of theoretical concepts. Similarly, in response to the interview question related to the learners' experience with the teaching and learning of programming at DUT, six interviewees suggested that lecturers spend more time providing clear explanations of concepts. This aspect had not been addressed in the closed quantitative questions, so the unprompted suggestions demonstrate the value of open-ended inquiry.

Finding 8:

The open-ended questionnaire responses of five participants suggested that *Alice* be introduced as a supplementary teaching tool in the ND: IT programme at DUT. An almost unanimous number of interviewees (seventeen of the eighteen) elicited the same response. Moreover, in response to the closed-ended questions in the questionnaire, a very high percentage (92.8%) of the participants expressed a keen interest in learning and working more with the *Alice* VPE. These findings validate that *Alice* is a useful tool that learners enjoy using and interacting with, while learning OOP.

Finding 9:

Across the board, there was a strong response regarding learners' improved understanding of OOP concepts after exposure to the *Alice* environment. As many as nineteen open-ended questionnaire responses and sixteen interview responses suggested an enhanced grasp in one or more aspects of OOP, including concepts such as methods, functions, events, inheritance, properties, parameters, classes, objects, instantiation and polymorphism. These findings are similar to the quantitative closed-ended responses elicited from 81.8% of the experimental participants, all of whom agreed that *Alice* impacted positively on their understanding of OOP concepts, including inheritance, methods, properties and functions. The feedback derived qualitatively and quantitatively confirms that *Alice* can be used successfully to teach the basic concepts within the OOP domain. Another qualitative sub-theme affirmed these findings, stating that *Alice* makes programming concepts easy to understand. Twenty-four participants in the open-ended questionnaire and ten interviewees experienced this. This high response rate verifies that *Alice* is indeed an effective tool for learning programming.

Finding 10:

Apart from an improved understanding of OOP, participants acknowledged that *Alice* allowed them to learn these concepts *visually*. Seven open-ended responses and eight interview responses mentioned this. Furthermore, a high percentage (94.6%) of the experimental participants responding to the closed-ended questionnaire, agreed that the *visual effects* in *Alice* provide a meaningful

context for understanding classes, objects, methods and events. It can be clearly deduced that learners appreciate the visual nature of the *Alice* environment.

Finding 11:

The *lack of motivation to learn programming* was presented in Section 2.3.2.1 as one of the core challenges in this study. Contrary to these literature findings, thirteen of the eighteen interviewees found that they were motivated to learn programming whilst studying at DUT. Moreover, in response to the quantitative closed-ended questions, an almost unanimous percentage (96.4%) of participants agreed that they were motivated to learn programming. Similarly, when questioned about the challenges facing learners in learning OOP, only three participants expressed a lack of motivation in the open-ended questionnaire responses and only two in the interviews. It seems that the learners at DUT are confident and motivated in learning programming, and that these findings are consistent.

Finding 12:

In response to the experience of the learner in learning programming at DUT, thirteen of the eighteen interviewees indicated that they spend time practicing programming exercises. Although this did not emerge from open-ended responses to the questionnaire, this good interview response correlates to the 60% combined agreement in the closed-ended questionnaire, stating that they spend a lot of time intensively practicing programming exercises. It is interesting to note that after participating in the *Alice* workshop, three interviewees believed that their exposure to the VPE had enhanced their motivation to practice examples in OOP. This assertion was verified quantitatively within the closed-ended questions, with 70.9% of the participants agreeing that they used *Alice* during their personal time after attending the first lesson of the *Alice* workshop. It can therefore be inferred that a good percentage of the participants were motivated to practice programming exercises. They also have a vested interest in expanding their programming skills through the use of supplementary learning tools, such as *Alice*.

Finding 13:

Five interviewees mentioned that their fellow students were able to assist them in their experience of learning programming at DUT. Similarly, in response to the closed-ended questionnaire, 83.6% agreed that they could interact and communicate successfully with other learners. These findings imply that most learners are able to collaborate and support each other with respect to teaching and learning at DUT.

6.8.2 Varying findings

Table 6.37 presents the findings that vary between the quantitative and qualitative components of Case Study 2. The findings highlighted in yellow vary between at least two methods. The findings highlighted in turquoise relate to a finding that arose only in one method, and was not addressed in the other two methods. The findings highlighted in pink reflect two differing findings that exist across all three methods.

Table 6.37 Varying findings between quantitative and qualitative data for Case Study 2

Finding No.	Domain	Theme/Sub-theme	Quan	Qual	
			A + SA (%)	Open-ended Q's in Q'aire (Frequency spontaneous responses) Threshold=5 N=55	Interview Q's (Frequency spontaneous responses) Threshold=3 N=18
1	Usability	Consistency and standards	93.4% (Section 6.4.2.1)	0 pos (Section 6.5.2.2)	0 pos (Section 6.6.2.2)
2	Usability	Flexibility and efficiency of use	76.4% (Section 6.4.2.1)	19 neg (Section 6.5.2.3)	9 neg (Section 6.6.2.3)
3	Challenges	Difficulty with problem-solving and applying concepts, e.g. when to write code for objects and methods	78.2% (Section 6.4.2.3)	14 (Section 6.5.2.4)	3 (Section 6.6.2.4)
		Methods (calling/overriding /abstract/virtual/no default/event-driven)	96.3% 90.9% (Section 6.4.2.4)	7 (Section 6.5.2.4)	3 (Section 6.6.2.4)
4	Techniques	Guiding the learner through the program logic with step-by-step instructions	40% (Section 6.4.2.4)	4 (Section 6.5.2.5)	3 (Section 6.6.2.5)
5	Challenges	Creating objects and instantiating objects	69.1% (Section 6.4.2.4)	11 (Section 6.5.2.4)	6 (Section 6.6.2.4)

6	T&L@DUT	I am satisfied with the lecturer's teaching styles	85.5% (Section 6.4.2.2)	1	12 (Section 6.6.2.8)
		Lecturers do not provide a clear explanation of concepts, leaving the learner to do independent research or seek assistance from third year learners	1	14 (Section 6.5.2.5)	6 (Section 6.6.2.8)

The second research sub-question of this study relates to the learners' experience with using Alice. Some of the findings presented in this section, namely the first two varying findings, address this research question.

Finding 1:

In terms of usability, *consistency and standards* (Heuristic 4) was ranked the third highest-rated heuristic in the quantitative closed-ended questions, with an average rating of 4.40. Contradictory to these positive results, none of the participants spontaneously gave positive responses pertaining to Heuristic 4 in either of the two qualitative methods. This invalidates the notion that *Alice* maintains a good level of consistency and standards.

Finding 2:

Flexibility and efficiency of use (Heuristic 7) obtained a category rating of 76.4% in response to the closed-ended questions. Specific to each sub-theme, 83.6% of the experimental participants agreed that *Alice* caters for beginner to expert users, whilst 69.1% claimed to having used various views and tools embedded in *Alice* to successfully move, align and reposition objects on the screen. The percentages for Heuristic 7 varied considerably in the qualitative feedback. Nineteen negative responses were obtained from the open-ended questions regarding Heuristic 7. Similarly, nine negative responses emerged from the interviews. Participants found that the basic use of the camera to focus on an object; the rotation, positioning and movement of objects; and the use of various views had proven to be time consuming and arduous. This should not have presented a problem if the VPE had catered for beginner users.

The remaining varying findings in this section relate to the first research sub-question, which investigates the effectiveness, as perceived by learners, of using the Alice visual programming environment in addressing the challenges facing novice programming learners within the object-oriented domain.

Finding 3:

One of the core challenges presented in Section 2.3.2.4 of this study relates to the difficulties of understanding compound logic and the application of algorithmic problem-solving skills. It emerged in some cases, that the main reason why learners struggle to create algorithms, is because they are unable to solve basic problems. Attesting to these findings, fourteen participants indicated in the questionnaires that they experienced difficulties with problem-solving and applying concepts. For example, some were unable to discern when to write code for objects and methods. Three interviewees expressed having to deal with the same challenge. A further seven open-ended questionnaire responses and three interview responses indicated that writing methods, in particular, posed a challenge. The quantitative responses to the closed-ended questions, however, differ from the qualitative feedback. Firstly, a good percentage (78.2%) of the participants agreed that they are capable of applying basic problem-solving techniques to create algorithms that solve problems. Secondly, an almost unanimous percentage of participants (96.3%) agreed that they are able to read and write in a formal programming language. Thirdly, 90.9% of the participants agreed that they can use creative thinking and programming concepts to write programs. These positive quantitative findings contradict the qualitative concerns regarding inability to problem-solve and write methods. The researcher attaches more importance to the spontaneous, unprompted qualitative data than the Likert-scale responses.

Finding 4:

Four open-ended questionnaire responses and three interview responses affirmed that participants preferred being guided through the program logic with step-by-step instructions. Conversely, 40% of the closed-ended questionnaire responses indicated that participants are able to write a program with a clear understanding of every line of code. Although these statistics are not significantly high, it is likely that some participants may have contradicted their initial assertions in this regard.

Finding 5:

A fair number of participants (eleven open-ended questionnaire responses and six interview responses) felt that the ability to create objects and instantiate them, posed a challenge in learning OOP. The closed-ended questionnaire responses contradict this claim, with a 69.1% combined agreement that they had acquired a good level of understanding of objects during their first year of study. Such a problem arises when learners are unable to follow through from one year to another on the knowledge gained.

Finding 6:

Regarding the teaching and learning at DUT, twelve interviewees expressed satisfaction with the lecturers' teaching styles. In response to the closed-ended questionnaire, a high percentage of participants (85.5%) agreed that the current teaching methods and techniques used in programming subjects at DUT helped them to program successfully. The qualitative and quantitative findings correspond with each other in relation to this sub-theme. However, a different stream of thought arose from another sub-theme. Fourteen open-ended questionnaire responses and six interview responses indicated that lecturers do not provide a clear explanation of concepts, placing the responsibility on the learner to self-study or seek assistance from other learners. There appears to be a distinct difference in the learners' thought processes regarding the lecturing style at DUT.

There is a further comparison of quantitative findings and qualitative findings in Sections 7.4.1 and 7.4.2 of the Conclusion Chapter, namely an overview of the most important aspects emerging from the quantitative and qualitative results of the whole study, i.e. findings from both Case Study 1 and Case Study 2.

The next section provides a summative overview of the findings revealed for Study 2.

6.9 Summary and conclusion

As discussed in Sections 4.3.2 and 5.8, the mixed-methods approach employed in this research involved quantitative and qualitative studies. As was the case in Study 1, this triangulated approach to data collection and analysis, contributed to validating the findings for Study 2.

Several challenges emerged from Study 1, for which recommendations were implemented to address these problems. These were presented in Section 6.2. Section 6.3 addressed the reliability of the questionnaire.

The findings of the quantitative data analysis of the closed-ended section in the questionnaire using SPSS, are presented in Section 6.4. With reference to Section 6.5 and Section 6.6, textual analysis was undertaken respectively of the qualitative data from the questionnaires and interviews to identify the themes that emerged. In sub-sections of Section 6.5, i.e. Sub-sections 6.5.2.1 to 6.5.2.7, the sub-themes that emanated from the open-ended responses to the questionnaire in Case Study 2 were discussed. In some instances, this was done by highlighting interesting comparisons to findings in Study 1. In sub-sections of Section 6.6, i.e. Sub-sections 6.6.2.1 to 6.6.2.7, the sub-

themes that emanated from the interviews and the findings of the open-ended responses to the questionnaire, were discussed in combination. In general, the findings were in line with each other.

Section 6.7 presented the quantitative analysis of test marks and exam results, comparing performances of the experimental learners who had been exposed to the *Alice* intervention and the comparison learners, who had been taught OOP by conventional teaching methods only. Both SPSS and Viscovery SOMine tools were used in the analysis process. The results revealed that the experimental group had performed significantly better than the comparison group.

Section 6.8 provided a comparison of the quantitative results (closed-ended questions in the questionnaire) and two sets of qualitative findings (open-ended questions in the questionnaire and interviews) for Case Study 2. The strength of a quantitative Likert scale is that participants are prompted throughout and the subject is addressed comprehensively in a short time. This brings numerous aspects to mind, whereas in an open-ended qualitative investigation, only the most outstanding attributes come to mind, given the limited time in which the questions must be answered. A qualitative study, therefore, provides a narrow range of information. The implication is that, for a comprehensive study, it is necessary to ask a large set of quantitative questions so as to elicit feedback from all participants on each matter. Qualitative research is, however, enriching because it elicits unprompted, and sometimes unanticipated, findings. These results make it clear that mixed-methods research, combining quantitative and qualitative data, is a sound approach.

Chapter 7 concludes the research, with a comparison and overview of the findings from Case Study 1 and Case Study 2.

Chapter 7: Conclusion

7.1 Introduction

This study set out to improve learners' understanding of programming within the domain of OOP, aided by a visual programming environment called *Alice*. *Alice* seeks to engage and motivate learners to grasp concepts of OOP, while they are creatively applying themselves to adapting animated movies and video games. During *Alice* interventions, participants' attention is drawn to new and exciting ways of programming that can enhance their skills, and stimulate their higher-order critical thinking capabilities. The ultimate aim of this study was to answer the primary research question:

To what extent can the implementation of the Alice visual programming environment in a second-level programming course at the Durban University of Technology improve the performance and learning experience of learners?

The three research sub-questions that emerged from the primary research question were presented in Section 1.6.2, and revisited in Sections 4.2 and 5.1. The sub-questions, which are illustrated in Figure 7.1, are:

1. *What is the effectiveness, as perceived by learners, of using the Alice visual programming environment in addressing the challenges facing novice programming learners within the object-oriented domain?*
2. *How do learners experience the usability of Alice?*
3. *To what extent do the test and exam results of participating learners relate to those of similar learners who were not exposed to the Alice intervention?*

This chapter endeavours to cohesively collate and link points from Chapters 1 through to Chapter 6, and consolidate the meaning of it all. Chapter 7 differs from Chapter 5 and Chapter 6, on Case Study 1 and Case Study 2 respectively, in that it presents qualitative quotations. These are extracted from participants' open-ended responses to the questions in the questionnaires across both case studies, and from interviews in Case Study 2. The quotations are used in Section 7.3 to emphasise the value of the parallel qualitative studies. As such, the mixed-methods methodology employed in this study to answer the three research sub-questions and the effectiveness of this approach, is evaluated in Section 7.3.

In Section 6.8 the quantitative and qualitative findings of Case Study 2 were compared. Chapter 7 provides a further comparison of quantitative and qualitative findings, but is an overview of the most important aspects emerging from the results of the whole study, i.e. it considers findings from both Case Study 1 and Case Study 2. This is discussed in Section 7.4, while re-visiting the research questions by integrating the empirical findings from Case Study 1 and Case Study 2 with respect to the individual research sub-questions. In the process, references are made to the quantitative and qualitative analyses, while interpretations are drawn from information in Chapter 5 and Chapter 6, and the literature studies.

Two diagrams, namely Figure 7.1 and Figure 7.2, are used to visually aid these discussions. A brief description of each diagram follows in Section 7.2.

The validity and reliability of the findings in the present study are addressed in Section 7.5. Section 7.6 provides direction and areas for future research, before presenting a selection of recommendations in Section 7.7. Finally, the chapter concludes with an overall summary of the study in Section 7.8.

The next section provides a visual representation of the study.

7.2 A visual representation of the study

Two diagrams have been generated to facilitate the discussion of the findings in this chapter.

Figure 7.1 provides a simple visual representation of the road map that binds the entire study together. It depicts the flow of the research processes from the inception of the primary question via the sub-questions to the determinations of its solution, as will be discussed in Section 7.4. The same figure was originally presented as Figure 4.1 in Chapter 4, the Research Design and Methodology chapter, whilst discussing the research flow. The arrows emanating from Sub-question 1 and Sub-question 2 in Figure 7.1 merge and lead into both Case Study 1 and Case Study 2, the results of which contribute to the final answer. Similarly Sub-question 3 is addressed in both studies, contributing further to the final answer to the primary research question.

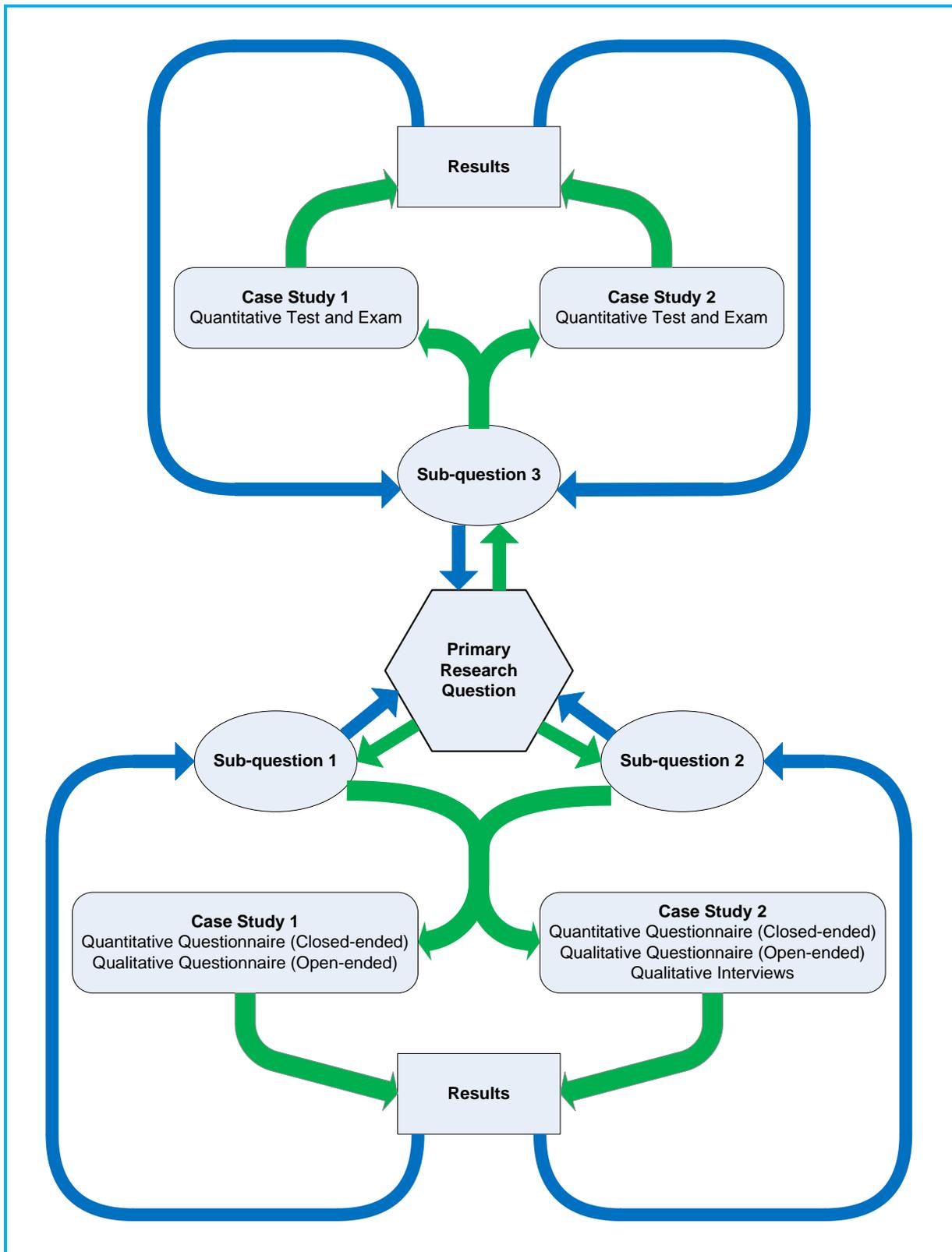


Figure 7.1 Road map that binds the entire study together

Figure 7.2 provides a comprehensive visual representation of the research process. It demonstrates continuity, cohesion and development of the logic adopted in this study. The diagram represents visualisation in action, as the more one looks at it, the more one sees.

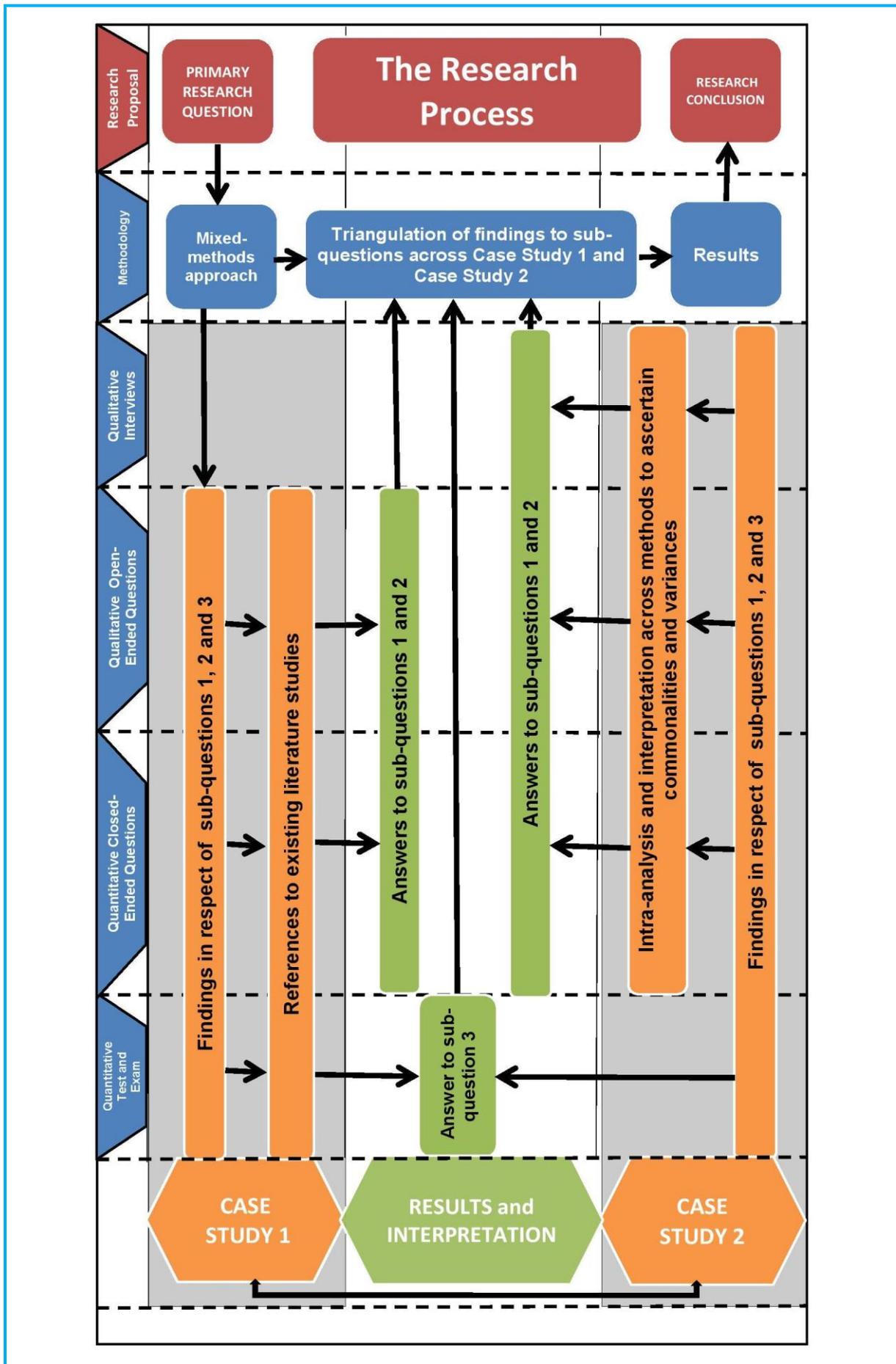


Figure 7.2 The research process model

To mention a few pertinent observations from Figure 7.2:

- (a) The varying block sizes in the diagram are used to emphasise the extent to which research methods were adopted within each study when finding answers to the sub-questions. This is demonstrated with a larger orange block for findings of Sub-questions 1, 2 and 3 in Case Study 2 when compared to the smaller one for findings of Sub-questions 1, 2 and 3 in Case Study 1. Similarly, the answers depicted in the green block for Case Study 2 are larger than that of the smaller green block for Case Study 1.
- (b) This can be justified by observing the research methods in blue blocks aligned along the extreme left margin, where the additional interview sessions extend the findings for Case Study 2.
- (c) The blue blocks further illustrate how the answers to Sub-questions 1 and 2 span across the questionnaires and interviews research methods, while Sub-question 3 is restricted only to the test and exam data.
- (d) Furthermore, the discussions of findings for Case Study 1 were enriched by references made to existing literature studies (see Sections 5.4, 5.5 and 5.6 in Chapter 5). This is depicted using the orange block within Case Study 1. On the other hand, intra-analysis and interpretations of commonalities and variances from quantitative and qualitative findings occurred within Case Study 2 only, and portrayed in orange (see Section 6.8 in Chapter 6).
- (e) The triangulation of the findings to sub-questions across Case Study 1 and Case Study 2 leads to the results and interpretation of the study and is illustrated by the blue blocks at the top of the diagram. The final answer to the primary research question is shown using the red blocks (see Section 7.4 in Chapter 7).

The next section explains the effectiveness of the mixed-methods approach used in this study.

7.3 Effectiveness of a mixed-methods approach

As discussed in Section 4.3.2, the mixed-methods approach employed in this research involved quantitative and qualitative studies. This triangulated the data collection and analysis methods, and proved to be an effective approach that strengthened the conclusive findings, as depicted in Figure 7.2.

Questionnaires were successfully administered to the participants of the *Alice* workshop during both case studies. The quantitative approach was effective in measuring their level of agreement or

disagreement in response to the closed-ended questions in the questionnaire (Likert scale integer values, average rating, category rating).

The qualitative open-ended questionnaire responses contributed positively towards eliciting rich, spontaneous findings and provided some interesting unanticipated findings. As shown in Figure 7.2, Case Study 2 strengthened the research by also using semi-structured interviews to elicit qualitative data relating to the participants' experiences with *Alice*. For example, in relation to Nielsen's (1994) Usability Heuristic 8 (*Aesthetic and minimalist design*), seven out of the eighteen interviewees praised the pre-defined methods and variables available on the *Alice* interface, which simplify the coding process. These comments emerged only during the one-on-one post-questionnaire interviews, and it is unlikely they would have surfaced from the questionnaire alone (see Appendix E.1- *Usab1.8.2*). As mentioned in the previous section, a few pertinent quotations made by the interviewees are provided to elaborate and illustrate:

Criterion: Pre-defined methods and variables simplify coding in Alice

“*Alice is so simple that there are options where you can already create your functions, your methods, your parameters and for me it's really... it's like enjoying programming!*” (Interviewee 16).

“OK, *Alice was so easy because uhhh... so, like the methods, they were done for us and so it was easy for us to copy or drag from the tree to the coding bar*” (Interviewee 2).

With reference to Figure 7.2, triangulation of data was conducted between the quantitative findings (closed-ended questionnaire) and the two sets of qualitative findings in Case Study 2 (open-ended questionnaire and interview). Common findings and varying results were encountered (see Section 6.8). A number of the issues raised in the quantitative analysis, were confirmed in the qualitative data. For example, with reference to Finding 2 in Table 6.36, a high percentage of participants (92.7%) strongly agreed that the *Alice* system easily related to real world objects and activities (*Heuristic 2 - Match between the system and the real world*). This was reinforced by detailed, unprompted comments in the qualitative data. Notable comments mentioned by the participants in relation to qualitative data from the open-ended data, as well as qualitative data from interviews in Case Study 2, are quoted below (see Appendix E.1- *Usab1.2.1* and *Usab1.2.3*):

Note: Quotations from the questionnaire are written in red font and interview responses are in blue font.

Criterion: Graphics and animation

“It taught me how to use methods and inheritance to animate and make objects do real life activities that are done by people and animals” (Questionnaire respondent 5, Case Study 1)

“I like *Alice* because it teaches how computer graphics and animation works, *Alice* is more concerned in programming, therefore I found it good for me because I love programming” (Questionnaire respondent 12, Case Study 2)

“For me it was extremely easy, simply because like there were pictures like graphics involved in using it and there’s no typing out of code, instead you drag your code onto like your work pane and because it was a lot of graphics, it’s easy to remember like...the concepts are easier to grasp so it made it really easy for me to use *Alice*” (Interviewee 6)

Criterion: Visualisation, e.g. see objects move on command

“...With *Alice* you get to see animated objects move on your command” (Questionnaire respondent 28, Case Study 2)

“What I enjoyed the most is to see cartoon or object moving...” (Questionnaire respondent 11, Case Study 1)

“You can do and see what’s the results for it. Because the time you do, you drag the code and you see the object moves” (Interviewee 2)

In addition, certain inconsistent and contradictory findings emerged from the two different methods. For example, with reference to Finding 6 in Table 6.37, varying thought processes emerged amongst participants regarding their satisfaction with lecturers’ teaching styles at DUT. It is unlikely that the richness of these observations could have been achieved through quantitative analysis alone.

As mentioned in Section 4.3.2.2, this study used data transformation, which was defined by Creswell (2009) as the quantification of qualitative data. With reference to the recently written book by Guest *et al.* (2012), this study adopted applied thematic analysis (see Section 4.8.2), which involved developing a codebook to quantify the qualitative responses derived from the open-ended questionnaire and interview questions.

With reference to Section 3.4.2, *Alice* allows learners to instantly visualise the execution of each animation program. Concomitantly, this study sought to provide the reader with a visual representation of findings. SPSS and Viscovery SOMine were used to perform quantitative analysis on learner results achieved from tests and examinations. Both statistical packages provided a graphical representation of findings, in the form of graphs, charts, maps and other diagrams.

The next section re-visits the findings that emerged in response to the three sub-questions that arose from the primary research question. This overview is conducted in the context of the empirical findings of the two case studies. It affirms that the research conducted, has adequately answered each of these sub-questions.

7.4 Re-visiting the research questions: Integration of empirical findings from Case Study 1 and Case Study 2

A non-comprehensive study was conducted in 2011, which provided preliminary answers to all three sub-questions (*Chapter 5: Data Collection and Analysis, Case Study 1*). The main empirical findings however, emanated from the 2012 study (*Chapter 6: Data Collection and Analysis, Case Study 2*). As mentioned in Section 7.1, this section provides a synthesis of the quantitative and qualitative findings of both case studies so as to answer the three research sub-questions shown in Figure 7.1 and Figure 7.2.

7.4.1 Sub-question 1: What is the effectiveness, as perceived by learners, of using the *Alice* visual programming environment in addressing the challenges facing novice programming learners within the object-oriented domain?

With reference to Figure 7.1 and Figure 7.2, Sub-question 1 was addressed in both Case Study 1 and Case Study 2. Findings of both case studies are summarised in this section, taking both the quantitative and qualitative data into consideration.

7.4.1.1 The teaching and learning of programming at DUT

The teaching and learning of programming at DUT was addressed in Sections 5.4.2.2, 6.4.2.2 and 6.6.2.8. The salient discoveries are as follows:

- (a) With reference to Table 5.7 and Table 6.9, the average ratings for each of the closed-ended questions on the teaching and learning of programming were similarly high for both studies, but mostly a little higher in Case Study 2, i.e. the overall average rate for Case Study 1 was 3.96 and 4.32 in the second study.
- (b) Previous studies conducted within other institutions by Moskal *et al.* (2004) and Stiller (2009) suggest that large volumes of course work in programming subjects, inadequate experience in computer usage and poor teaching methods and techniques contribute towards learner attrition. However, high percentages of participants in the present study indicated adequate satisfaction pertaining to both the breadth and depth of material and the teaching methods and techniques used at DUT. Study participants further expressed relative confidence in their prior experience of computer usage. High rates of attrition exist at DUT (this is supported with statistical data given in Section 5.4.2.2), but due to the mainly positive responses with regard to the factors mentioned above, all these factors can be disregarded as likely reasons for learner attrition. Attrition did indeed occur in Case Study 1 but pro-active measures taken by the researcher in Case Study 2 alleviated the problem, as mentioned in Section 6.2;

- (c) Albeit an average-to-good number of participants expressed high levels of comfort with OOP prior to the *Alice* workshop, around a third were unsure about their level of understanding. This implied that at least 33% of the participants were prime candidates for the proposed intervention;
- (d) Participants almost unanimously agreed that learning OOP in a fun, engaging and interactive way was more appealing to them. This supports the assertion that *Alice* appears to increase learner motivation, thereby addressing one of the key challenges in OOP that this research sought to investigate, and will be addressed in the next section;

7.4.1.2 Challenges faced by learners in learning OOP

The challenges facing learners in learning OOP were addressed in Sections 5.4.2.3, 5.5.2.4, 6.4.2.3, 6.5.2.4 and 6.6.2.4. The core findings are highlighted below:

- (a) *Lack of motivation for programming*: In Case Study 1, the motivation to learn programming on the part of the participants was found to be notably high, however, insufficient stratification existed in the questionnaire to ascertain whether this could be attributed to the *Alice* intervention. Case Study 2 included a further closed-ended question, which verified that more than 80% of the participants acknowledged an increase in motivation after being exposed to the visual tool. This reinforces that *Alice* does contribute positively to uplifting learner motivation. As a matter of concern, both studies revealed that a third of the *Alice* participants spent insufficient time on practicing programming exercises. It is proposed that learners need to spend more time practicing programming in order to maintain the confidence obtained from an intervention of this nature;
- (b) *Complex syntax, logic and semantics*: Participants found the drag-and-drop feature of *Alice* to be a pleasant release from the complexity of using syntax and semantics. Contradictorily, it was discovered in both studies that a fair number of participants possessed a high level of comfort with the textual nature of conventional learning methods. An assertion made by Carlisle (2009), that the textual nature of most programming environments work against typical learning styles, was not supported by this study;
- (c) *The need for immediate feedback and identifying results of computation as a created program runs*: In response to Questions 14.1, 14.2 and 14.3 in the questionnaire, participants in Study 1 stated that they were able to identify errors and correct them using immediate feedback when such was given by the program they were using. By adding a further question, namely Question 14.4, Study 2 queried the feedback provided specifically by *Alice*. Eighty percent of the participants felt that *Alice* provides immediate feedback as the program executes.

- (d) *Difficulties in understanding compound logic and the application of algorithmic problem-solving skills*: The quantitative data indicated that many participants were capable of problem-solving and had a good understanding of pseudocode. Contradictorily, only a few of them stated that they used pseudocode to outline the logic of a program. This inconsistency was evident across both case studies. The qualitative interviews confirmed that learners did indeed experience difficulty with problem-solving and applying OOP concepts. However, an additional closed-ended question in Case Study 2 clearly indicated that most participants found that *Alice* supported them in focusing on problem-solving.

The responses above imply that the VPE is beneficial for learning OOP, which will be further addressed in Sections 7.4.1.4 and 7.4.1.5.

7.4.1.3 Techniques to improve the teaching of OOP

The techniques used to improve the teaching of OOP were addressed in Sections 5.4.2.4, 5.5.2.5, 6.4.2.4, 6.5.2.5 and 6.6.2.5. The prominent results are discussed below:

- (a) *Algorithmic thinking and expression*: Participants almost unanimously agreed that they were able to read and write in a formal language.
- (b) *Abstraction*: As a matter of concern, a high number of participants acknowledged that they sometimes write programs without understanding every line of code, which implies that they are unable to grasp programming concepts on an abstract level. This needs further addressing.
- (c) *Objects-first strategy*: A high percentage of participants stated that *Alice* helps them to see everything as an object. Furthermore, about half of them felt that it would be easier to learn OOP during the first year of study. It is encouraging that an introduction to basic OOP concepts has now been implemented into the DS1 syllabus as of 2012. However this was done during the second semester and had no impact on the DS2 participants of 2011 and 2012 study.
- (d) *3D animation authoring tools and visualisation*: It emerged from the quantitative data that participants rated their improved understanding of programming concepts highly. They attributed this to the visual effects in *Alice*. In qualitative data, without being spurred, a high number motivated the introduction of *Alice* into the syllabus as a supplementary teaching tool. The acquisition of such findings can be attributed to the mixed-methods approach adopted in this study.

7.4.1.4 Impact of *Alice* in addressing the challenges and improving the understanding of OOP

The impact of *Alice* in addressing the challenges and improving the understanding of OOP was addressed in Sections 5.4.2.5, 5.5.2.6, 5.5.2.7, 6.4.2.5, 6.5.2.6, 6.5.2.7, 6.6.2.6 and 6.6.2.7. Several points regarding the positive influence that *Alice* had made to this end, have been mentioned in Sections 7.4.1.2 and 7.4.1.3. In addition to this, other findings emerged:

- (a) Participants expressed an interest in learning more about computer graphics and animation, after being exposed to the *Alice* intervention.
- (b) Others had used *Alice* during their personal time and were eager to work more with the VPE, whilst most were motivated to learn more about OOP.
- (c) The participants felt that it was easier to learn programming through visualisation, rather than having to remember the syntax for coding.
- (d) Participants found that the *Alice* workshop related directly to OOP concepts covered in the DS2 syllabus.
- (e) A large positive response affirmed that *Alice* improves the understanding of OOP concepts, and that *Alice* can help address the challenges of OOP.
- (f) Nine groups of participants demonstrated such enthusiasm and motivation to learn, that they voluntarily used the VPE during their personal time to develop collaborative projects.
- (g) With reference to the various learning theories discussed in Chapter 2, it is proposed that *Alice* promotes constructivist learning. Wulf (2005) described constructivism as a learner-centered pedagogy, which typically entails experiential discovery learning through exploration. The *Alice* VPE is designed to familiarise learners with real-world objects whilst being able to tell a story. Users can nurture the skills needed to create something that is personal and deeply expressive.

7.4.1.5 Criteria based on the researchers' experience

A summary of criteria that emerged from the researchers' personal experience in teaching OOP to IT learners, is given below (see Sections 5.4.2.5 and 6.4.2.5):

- (a) A high percentage of the experimental groups affirmed that *Alice* had taught them to program incrementally. Moreover, a fair number of used bricolage (trial-and-error) to 'try out' individual animation instructions, when creating methods using *Alice*. It would be rewarding to see learners applying the knowledge gained from interacting with *Alice*, whilst coding methods and functions in conventional programs.

- (b) It emerged from the qualitative data, that many participants found it challenging to learn OOP concepts such as inheritance, methods, properties and functions. The responses to the closed-ended questions, however, indicated that *Alice* helped most participants to improve their understanding of these concepts.
- (c) It was encouraging to note that learner collaboration was fostered amongst most participants of the *Alice* workshops. The ability to interact and communicate well with other learners, are skills that learners should be able to transpose to their third-year course work.

7.4.1.6 The answer to Sub-question 1

The findings presented from Sections 7.4.1.1 to 7.4.1.5 support the answer to the first sub-question.

The answer to Sub-question 1 is that *Alice* showed itself to be an effective tool that addressed challenges faced by novice programming learners within the object-oriented domain.

7.4.2 Sub-question 2: How do learners experience the usability of *Alice*?

As shown in Figure 7.1 and Figure 7.2, Sub-question 2 was addressed in both Case Study 1 and Case Study 2. The participants' experiences on the usability of *Alice* was successfully probed through the use of closed-ended and open-ended questions in the questionnaire during the *Alice* workshops, and for Case Study 2, usability was investigated further during the post-questionnaire interviews. The findings on usability from Case Study 1 were presented in Chapter 5 (see Sections 5.4.2.1, and 5.5.2.1 to 5.5.2.3) whilst the results of Case Study 2 were addressed in Chapter 6, with some references made to Case Study 1 (see Sections 6.4.2.1, 6.5.2.1 to 6.5.2.3, and 6.6.2.1 to 6.6.2.3). The core findings from both case studies are highlighted in this section, in order to answer Sub-question 2.

The quantitative approach to assessing the participants' perceptions on the usability of *Alice* was done using the ten closed-ended questions of the questionnaire that were formulated using Nielsen's (1994) ten interface design heuristics (see Section 4.5.2.1). The ranking of conformance of *Alice* to these heuristics yielded the following:

- (a) Case Study 1 resulted in a general agreement of 70.7 – 92.9% from the participants for seven of the heuristics, and a category rating of 3.83 – 4.50 (see Table 5.6);
- (b) Congruently, Case Study 2 followed with a general agreement of 74.4 – 92.7% from the participants for seven of the heuristics, and a category rating of 4.00 – 4.55 (see Table 6.8).

Six of the seven heuristics had corresponding positive results across both case studies. These are:

- (a) *Match between the system and the real world*;
- (b) *Visibility of the system status*;
- (c) *Consistency and standards*;
- (d) *User control and freedom*;
- (e) *Help and documentation*; and
- (f) *Recognition, diagnosis and recovery from errors*.

Two of the heuristics, namely, *Aesthetic and minimalist design* and *Recognition rather than recall*, were rated less positively in both case studies. There is thus a strong similarity between the quantitative data of Case Study 1 and Case Study 2. This indicates a notable degree of **consistency over time and participants**, confirming that the methodology adhered to the principles of validity and reliability, and will be addressed in Section 7.5.

Based on the majority of quantitative findings, it appears that *Alice* has good usability.

The findings presented thus far, were derived from the quantitative data on usability. Findings with regard to the qualitative data, are now discussed. This is done by combining positive and negative assertions that emerged from open-ended responses in the questionnaire and interviews. The researcher used the frequency counts in the codebook (see Appendix E.1) to find the highest rated qualitative responses.

During analysis of responses to the question “Explain your experience with using *Alice*”, notably positive responses on the usability of *Alice* occurred for the following heuristics:

- (a) *Match between the system and the real world*: Participants enjoyed working with graphics and animations whilst programming. They suggested that visualisation allowed them to see objects move on command. Others appreciated how real 3D objects bring coding into reality, whilst reflecting real-world scenarios.
- (b) *Aesthetic and minimalist design*: Most participants found the *Alice* interface simple and easy to use. A few interviewees explained that this was due mainly to the pre-defined methods and variables, which simplify the coding process;
- (c) *Recognition rather than recall*: a high number of participants agreed that *Alice*’s drag-and-drop feature relieved them from having to type code, as is required with conventional software. This released them to focus on problem-solving. Others preferred coding with *Alice*’s English-like statements and appreciated not having to deal with complex syntax;

- (d) *User control and freedom*: Participants enjoyed the freedom to develop video games and movies; and
- (e) *Visibility of the system status*: Participants found the VPE interactive. *Alice* was able to generate immediate feedback upon program execution.

Whilst the above-mentioned comments are favourable with regard to the usability of *Alice*, some negative perceptions arose in response to the question, “Name the problems you encountered in using *Alice*”. The negative responses are listed along with the associated heuristic:

- (a) *Consistency and standards*: Participants indicated that *Alice* takes too long to load;
- (b) *Recognition rather than recall*: Amongst other issues related to methods and functions, participants experienced difficulty with the use of *Alice*’s built-in Math function;
- (c) *Flexibility and efficiency of use*: A number of participants found it time consuming to use the camera to focus on a particular object and to zoom in and out of a scene;
- (d) *Aesthetic and minimalist design*: A few would have preferred having the option to customise the user interface, since *Alice* does not allow learners to adjust the GUI. Others suggested that the user interface was cluttered; and
- (e) *Recognition, diagnosis and recovery from errors*: Some participants complained that *Alice* stalled or crashed intermittently, and some were unable to recover from their errors.

Some of these negative comments suggest areas of improvement, which the developers of *Alice* could address in later versions of the software. However, some of the problems are related to learners’ lack of skills. This calls for practice, since skills need to be developed over time. Although certain negative perceptions emerged regarding *Alice*, they were outweighed by the number of positive comments, and the number of times that each comment was made.

In the qualitative findings, the preponderance of positive perceptions affirms the quantitative findings that *Alice* has good usability.

The three heuristics that emerged from the triangulated data analysis with only positive comments in both the quantitative and qualitative results are:

- (a) *Match between the system and the real world*;
- (b) *User control and freedom*; and
- (c) *Visibility of the system status*.

With reference to Table 6.8, two of these heuristics, (a) and (c), are rated as the top two heuristics for Case Study 2, the main empirical study in this research. This reinforces the assertion that a satisfactory level of **consistency across methodologies exists** between the quantitative and the qualitative data.

The answer to Sub-question 2 is that learners found *Alice* to have good usability.

7.4.3 Sub-question 3: To what extent do the test and exam results of participating learners relate to those of similar learners who were not exposed to the *Alice* intervention?

With reference to Figure 7.1 and Figure 7.2, Sub-question 3 was addressed in both Case Study 1 and Case Study 2. SPSS was used to analyse the test and exam data of the participants who attended the *Alice* workshops (i.e. the experimental group), in comparison to the data of a similar set of learners who were not exposed to the *Alice* intervention (i.e. the comparison group) (see Sections 5.6.2.1 and 6.7.2.1). The test and exam data was processed using a data mining tool called Viscovery SOMine, to assist with deeper interpretations, such as the identification of at-risk learners (see Sections 5.6.2.2 and 6.7.2.2).

In Case Study 1, the findings in answer to Sub-question 3, did not demonstrate a statistically significant difference in learner performance from using *Alice*, when compared to the comparison group. As mentioned previously, Case Study 1 was not a comprehensive study, since only 21 participants remained for the duration of the *Alice* workshop. However, the mean scores for three of the four assessments conducted after the *Alice* intervention were higher for the experimental group, indicating a positive difference, even if not a significant difference. Table 5.21 indicates that both the examination performance and the final mark were approximately 5% higher in the experimental group, with a 10% higher difference for Test 2. Moreover, responses to Questions 25.2, 25.3 and 25.5 (Table 5.10) indicate that ***Alice* had notably increased motivation amongst the participants in Case Study 1.**

Case Study 2, however, successfully maintained a full complement of 55 participants until the workshop ended. The main empirical findings indicated that the p-value for the t-test was less than 0.05. This demonstrates that there was a statistically significant difference between the mean values. Moreover, Table 6.35 indicates that in 2012 both the examination performance and the final mark were approximately five to seven percent higher in the experimental group than in the control group of 2012, with a three percent higher difference for Test 2. It is clear that interaction with the *Alice*

VPE spawned a significant difference in performance. Similar to Case Study 1, responses to Questions 25.2, 25.3 and 25.5 (Table 6.12) verify that *Alice* had notably increased motivation amongst the participants in Case Study 2.

The answer to Sub-question 3 is that the test and exam performance of learners who are taught *Alice*, is better than those of similar learners who were not exposed to the *Alice* intervention.

Having re-visited Sub-questions 1, 2 and 3, attention is now paid to the answer to the primary research question.

7.4.4 Primary research question: To what extent can the implementation of the *Alice* visual programming environment in a second-level programming course at the Durban University of Technology improve the performance and learning experience of learners?

The learning experience of DS2 learners at DUT was enhanced and improved with the *Alice* intervention. *Alice* showed itself to be an effective tool that addressed challenges faced by novice programming learners within the object-oriented domain, as was supported by the answer to Sub-question 1. Furthermore, the answer to Sub-question 2 confirmed that learners found *Alice* to have good usability.

With regard to learner performance, the answer to Sub-question 3 indicated that the test and exam performance of the experimental participants in Case Study 2 of 2012, who were taught *Alice* in a large study, was statistically better than those of similar learners in the comparison group, who were not exposed to the *Alice* intervention. Furthermore, the mean scores for the experimental learners were consistently higher in Case Study 2 when compared to the comparison learners. The participants also indicated that they had an increase in motivation.

With regard to these aspects in the smaller-scale study, Case Study 1 of 2011, a significant difference was not found between the 2011 experimental participants and the comparison group. However, the mean scores of the 2011 experimental group for assessments conducted after the *Alice* intervention were, in general, higher than those of the comparison group, and the participants reported increased motivation.

The answer to the primary research question is that the implementation of the *Alice* visual programming environment in a second-level programming course at the Durban University of Technology successfully improved the performance of learners and their learning experience.

The next section demonstrates that this study adheres to principles of validity and reliability.

7.5 Validity and reliability

In Section 4.7, the terms *validity* and *reliability* were defined. It was the role of the researcher to ensure the validity and reliability of the findings of the present study, as discussed in the Data Collection and Analysis chapters (Chapters 5 and 6).

Creswell (2009) states that data analysis in mixed-methods research should consider dual facets, namely checking both the validity of the quantitative data and the accuracy of the qualitative findings. Cohen *et al.* (2011:179) define validity as “a demonstration that a particular instrument in fact measures what it purports to measure”. Guest *et al.* (2012) state that for any given study, if the same trends and themes emerge within data from different participant groups and data collection methods, then the validity of the findings is increased considerably. Reliability refers to “consistency and replicability over time, over instruments and over groups of respondents. It is concerned with precision and accuracy.” (Cohen *et al.*, 2011:199).

Table 7.1 re-visits these aspects of validity and reliability introduced in Section 4.7, and shows how they were implemented within the context of the present study. The table provides a summary of the types of validity and reliability, as well as the locations in the dissertation where they were addressed.

Table 7.1 Types of validity and reliability and how these were implemented within the context of the present study

Types of validity and reliability	Location in dissertation	How these aspects were implemented within the context of the present study
Quantitative reliability and validity	Section 4.6	The use of more than one data analysis tool contributed towards confirming the findings pertaining to learner performance. The primary tool used for the quantitative analysis was SPSS and the secondary tool was Viscovery SOMine.
	Section 4.7	
	Section 4.8.1.1	
	Section 5.3	The pilot study increased the reliability, validity and practicability of the questionnaire.
	Section 6.3	The reliability statistics of the questionnaires used in both case studies were computed by using Cronbach’s Alpha.

Qualitative reliability	Section 4.7	The data presented in this study is an accurate reflection of what occurred in the natural setting. Furthermore, the researcher's approach was consistent across Case Study 1 and Case Study 2.
		A codebook was used for analysing qualitative textual data derived from the open-ended responses from the questionnaires across both case studies and the interviews in Case Study 2.
		Each interview transcript in Case Study 2 was transcribed verbatim without losing the richness and accuracy of the participant responses, and was double-checked by the researcher.
Qualitative validity	Section 4.3.2.5 Section 4.7 Section 6.8	As mentioned in Section 7.3, the triangulation of open- and closed-ended questions, as well as the use of interview data and methods allowed the researcher to establish themes from multiple sources by compiling converging ideas and perspectives of the participants; thus increasing the validity of the findings.
		The use of rich, thick descriptions connected the findings of the literature as discussed in Chapters 2 and 3 to the empirical evaluation work discussed in Chapters 5 and 6, thereby contributing to the validity.
		In cases where the opinion of the researcher was expressed, it was indicated that such statements were contributions made by the researcher.
		Quotes made by participants were used verbatim to support themes and interpretations, thereby increasing the validity of the findings.
		The dual-case study was conducted over a two-year period, allowing comparison of the findings from two different case studies and thus increasing validity.

The methodology employed in this study and the results achieved, while answering the research questions, have adhered to the principles of validity and reliability. These were further enhanced by triangulation, which was addressed in Section 7.3.

The following section suggests areas that require further research as a follow-up to this study.

7.6 Future research

Following on the findings of the present study, areas for possible future research are proposed:

- (a) A broad study of the learner attrition issue is required on a wider selection of the DS2 population. The learners used in this study were 'good learners' who had passed all first-year subjects at first attempt, hence it is assumed that they were more likely to complete the ND: IT qualification;
- (b) Thirty-three percent of the participants in the interview voiced unprompted concerns regarding the quality of facilitation (see Section 6.6.2.8), which resulted in them seeking assistance from

other sources, such as third-year learners and personal research. With reference to Finding 6 in Table 6.37, these findings did not emerge in quantitative studies, but became clear from qualitative data. It must be borne in mind that a distinction exists between good teaching methods and techniques versus quality of facilitation. Further research and investigation needs to be undertaken with regards to these perceptions;

- (c) Following from the previous finding that some participants expressed reservations regarding the quality of instruction at DUT, it is suggested that the current system for lecturer development be formalised. A research study could be conducted to ascertain the feasibility of implementing a Continuous Professional Development (CPD) programme within the DUT IT department. This is elaborated in the next section, 'Recommendations';
- (d) Both case studies revealed that a third of the *Alice* participants were spending insufficient time practicing programming exercises to gain experience in debugging. Further investigation is required to establish the reasons behind the lack of practice. Strategies and interventions should be proposed that could be employed by tertiary institutions to reduce this problem;
- (e) The problem of learner inability to grasp programming concepts on an abstract level requires further scrutiny. This assertion is based on the finding that many learners write programs without understanding each piece of the code within the program.
- (f) Section 6.2 describes the measures the researcher took in Case Study 2 to address the issue of the learner attrition that had occurred in Case Study 1. These pro-active measures also constitute a contribution to knowledge in the field, however, the issue of attrition and safeguards against it could be investigated in a future study.

The next section makes recommendations that could be considered for implementation.

7.7 Recommendations

Based on the results in this study, the researcher proposes the following:

- (a) The positive results presented in this study provide grounds to incorporate *Alice* as a component of the DS1 or DS2 syllabus. It is recommended that the IT department at DUT consider this.
- (b) The researcher recommends that a further study should be conducted in the 2014 academic year, using the latest *Alice* 3 version. This version contains more explicit support for transitioning to Java - a programming languages taught within the ND: IT programme.
- (c) Human beings are known to learn from pictures. As mentioned in Section 2.4.4, Gomes and Mendes (2007) point out that many tools for solving programming complexities use animation and simulation techniques, aiming to take advantage of the potential of the human visual

system. The enthusiastic response received from the experimental groups in this study, in relation to visualisation, was indicative of a positive impact. It is therefore recommended that no matter which visual programming environment is implemented, at least one such intervention should be used to supplement the teaching and learning of OOP at DUT.

- (d) Robotic software, and in particular the *Lego Mindstorm NXT* robot (see Section 3.3.10), would provide an affordable, flexible and fun learning experience for the learners at DUT. This could contribute to solving two challenges previously highlighted namely, *lack of motivation for programming* and *the need for immediate feedback and response*.
- (e) It appears that the *Alice* intervention provided a platform for learners to collaborate with other learners, whilst learning in an engaging environment. It is recommended that other support programmes, such as formal peer to peer programming and paired-programming, be adopted to foster further collaborative studying amongst learners. The informal system currently in place to assist DS2 learners by their DS3 counterparts, could become a highly constructive and rewarding initiative. It is widely believed that an excellent way to learn and to consolidate learning, is to teach. In so doing, DS3 learners would solidify their prior-year knowledge. This could be implemented as early as the first year of study.
- (f) It is further recommended that an *Alice* workshop be conducted with first-year learners, during their orientation sessions at the beginning of the academic year. This would provide an interesting learning experience, whilst providing a gentle introduction to basic programming concepts.
- (g) It emerged that learners experienced difficulties in problem solving. Good, logical programming is developed through sound pre-code planning and organisation. This is assisted by the use of tools such as flowcharts, pseudocode and algorithms. It is proposed that the introduction of an additional subject on 'Logic' into the ND: IT programme will benefit the learners by improving their motivation and confidence to write programs.
- (h) Tertiary institutions should strive continuously to ensure that the quality of the education they offer is of the highest standards. The DUT IT Department has various commendable initiatives in place to incentivise, prompt and monitor staff performance with regards to continuous research and development. International and national best practice recommendations around quality dictate that professionals, such as doctors and engineers, spend a minimum number of hours on Continuing Professional Development (CPD). It is proposed that such a formal compulsory intervention is of equal significance for lecturers, as they are custodians of knowledge and the transfer of skills. The informal systems currently in place to ensure that lecturers embark on training courses in order to maintain their knowledge and skills, could be

formalised discipline-wide into a structured CPD system. This could assist in providing an irrefutable, transparent and high quality delivery mechanism for the IT classroom.

The final section below provides a brief summary to this study.

7.8 Summary and conclusion

The study set out to improve learners' understanding of programming within the domain of OOP, aided by a visual programming environment called *Alice*. A mixed-methods approach was adopted, with a view to extrapolating pertinent commonalities and variances across the quantitative and qualitative findings. This proved to be highly effective.

It was established that *Alice* is an effective tool that addressed challenges faced by novice programming learners within the object-oriented domain. Learners found *Alice* to have good usability. It was further established that the test and exam performance of learners who were taught using *Alice*, is better than those of similar learners who were not exposed to the *Alice* intervention. Moreover, the experimental learners indicated that their motivation had increased due to their experience with the *Alice* VPE.

The implementation of the *Alice* visual programming environment in a second-level programming course at the Durban University of Technology successfully improved the performance of learners and their learning experience. The dissertation contributes to dispelling the notion by Charles M. Schulz, "Try not to have a good time...this is supposed to be educational."

“The best thing for being sad,” replied Merlin, beginning to puff and blow, “is to learn something. That’s the only thing that never fails. You may grow old and trembling in your anatomies, you may lie awake at night listening to the disorder of your veins, you may miss your only love, you may see the world about you devastated by evil lunatics, or know your honour trampled in the sewers of baser minds. There is only one thing for it then – to learn. Learn why the world wags and what wags it. That is the only thing which the mind can never exhaust, never alienate, never be tortured by, never fear or distrust, and never dream of regretting. Learning is the only thing for you. Look what a lot of things there are to learn.” ~ T.H. White, The Once and Future King

~ ~ ~ ~ ~

*The Mock Turtle went on: “We had the best of educations – in fact, we went to school every day...”
I’ve been to day-school too,” said Alice; “you needn’t be as proud as all that.”*

“With extras?” asked the Mock Turtle a little anxiously.

“Yes,” said Alice, “we learned French and music,”

*“And washing?” said the Mock Turtle. ... “Now at ours they had at the end of the bill, ‘French, music and **washing – extra.**’”*

“You couldn’t have wanted it much,” said Alice; “living at the bottom of the sea.”

“I couldn’t afford to learn it,” said the Mock Turtle with a sigh. “I only took the regular course.”

~ Lewis Carroll, Alice in Wonderland

The situation above need not be the case for DUT learners if the *Alice* VPE becomes part of the ‘regular course’, to the benefit of both learners and academics.

References

- AGARWAL, R., HARRINGTON, D. & GUSMAN, C. 2012. Lego Mindstorm NXT controller with peer-to-peer video streaming in Android. *Journal of Computing Sciences in Colleges*, 27, 243-252.
- ALESSI, S. M. & TROLLIP, S. R. 2001. *Multimedia for learning: methods and development*, Needham Heights, Massachusetts, Allyn & Bacon.
- ANNIROOT, J. & DE VILLIERS, M. R. A study of *Alice*: A visual environment for teaching object-oriented programming. Proceedings of the IADIS International Conference Information Systems 2012, 10-12 March 2012 Berlin, Germany. IADIS Press.
- ANNIROOT, J. & DEAN, E. J. Using self-organizing maps to analyse first year IT results in relation to the student's matriculation results. Proceedings of the 35th Conference of SACLA, 3-6 July 2005 Kasane, Botswana. University of Botswana, 22-27.
- BALDWIN, R. G. 2007. *Learning to program using Alice* [Online]. Available: <http://www.dickbaldwin.com/alice/Alice0150.htm> [Accessed 26 December 2010].
- BECKER, B. W. Teaching CS1 with Karel the robot in Java. SIGCSE '01: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, 2001 Charlotte, NC, USA. ACM, 50-54.
- BEN-ARI, M. Constructivism in computer science education. SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education, 1998 Atlanta, GA, USA. ACM, 257-261.
- BENNEDSEN, J. & CASPERSEN, M. E. Abstraction ability as an indicator of success for learning computing science? ICER '08: Proceedings of the Fourth international Workshop on Computing Education Research, 6-7 September 2008 Sydney, Australia. ACM, 15-26.
- BRADY, K., GUZDIAL, M., STECKEL, M. & SOLOWAY, E. Whorf: A visualization tool for software maintenance. Proceedings of the IEEE Workshop Visual Languages, 15-18 September 1992 Seattle, WA, USA. IEEE, 148-154.
- BROOKS, R. 1999. Towards a theory of the cognitive processes in computer programming. *International Journal of Human-Computer Studies*, 51, 197-211.
- CARBONE, A., HURST, J., MITCHELL, I. & GUNSTONE, D. An exploration of internal factors influencing student learning of programming. ACE '09: Proceedings of the Eleventh Australasian Conference on Computing Education, January 2009 Wellington, New Zealand. Australian Computer Society, Inc., 25-34.
- CARLISLE, M. C. 2009. Raptor: a visual programming environment for teaching object-oriented programming. *Journal of Computing Sciences in Colleges*, 24, 275-281.

- CARNEGIE MELLON UNIVERSITY. 2012. *Alice: an educational software that teaches students computer programming in a 3D environment* [Online]. Carnegie Mellon Foundation. Available: http://www.alice.org/index.php?page=what_is_alice/what_is_alice [Accessed 08 August 2010].
- CLARK, R. E. 1994. Media will never influence learning. *Educational Technology Research and Development*, 42, 21-29.
- CLIBURN, D. C. Student opinions of Alice in CS1. FIE '08: Proceedings of the 38th Annual Frontiers in Education Conference, 22–25 October 2008 Saratoga Springs, NY. IEEE, T3B-1-T3B-6.
- COHEN, L., MANION, L. & MORRISON, K. 2007. *Research methods in education*, New York, USA, Routledge.
- COHEN, L., MANION, L. & MORRISON, K. 2011. *Research methods in education*, New York, USA, Routledge.
- CONWAY, M., AUDIA, S., BURNETTE, T., COSGROVE, D. & CHRISTIANSEN, K. Alice: lessons learned from building a 3D system for novices. CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems, 1-6 April 2000 The Hague, Amsterdam. ACM, 486-493.
- COOPER, S., DANN, W., LEWIS, D., LAWHEAD, P., RODGER, S., SCHEP, M. & STALVEY, R. A pre-college professional development program. ITiCSE '11: Proceedings of the 16th annual joint conference on Innovation and technology in computer science education, 27-29 June 2011 Darmstadt, Germany. ACM, 188-192.
- COOPER, S., DANN, W. & PAUSCH, R. Teaching objects-first in introductory computer science. SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education, February 2003. ACM, 191-195.
- CRESWELL, J. W. 2009. *Research design: Qualitative, quantitative, and mixed methods approaches*, Thousand Oaks, California, Sage Publications, Inc.
- CRESWELL, J. W. & PLANO CLARK, V. L. 2011. *Designing and conducting mixed methods research*, Thousand Oaks, California, Sage Publications, Inc.
- DANN, W. P., COOPER, S. & PAUSCH, R. 2009. *Learning to program with Alice*, Upper Saddle River, New Jersey, Pearson Education, Inc.
- DANN, W. P., COOPER, S. & PAUSCH, R. 2012. *Learning to program with Alice*, Boston, Massachusetts, Prentice Hall, Pearson Education, Inc.
- DE VILLIERS, M. R. 2005. e-Learning artifacts: are they based on learning theory? *Alternation*, 12.1b, 345-371.
- DEBOECK, G. & KOHONEN, T. 1998. *Visual explorations in finance: with self-organizing maps*, New York, Springer.
- DICKEY, M. D. 2003. Teaching in 3D: Pedagogical affordances and constraints of 3D virtual worlds for synchronous distance learning. *Distance Education*, 24, 105-121.

- DIX, A., FINLAY, J., ABOWD, G. D. & BEALE, R. 2004. *Human-computer interaction*, Harlow, England, Pearson Education Limited.
- DU, H. 2010. *Data mining techniques and applications*, Andover, Hampshire, Cengage Learning.
- EDWARDS, H. K., GERSTING, J. L. & TANGARO, T. Teaching Alice in Hawai'i: Cultural perspectives. FIE '07: Proceedings of the 37th Annual Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports, 10-13 October 2007 Milwaukee, WI. IEEE, T3A-1-T3A-5.
- ELMASRI, R. & NAVATHE, S. B. 2000. *Fundamentals of database systems*, Reading, MA, Addison-Wesley.
- ESTEVES, M., FONSECA, B., MORGADO, L. & MARTINS, P. Contextualization of programming learning: A virtual environment study. FIE '08: Proceedings of the 38th Annual Frontiers in Education Conference, 22-25 October 2008 Saratoga Springs, NY. IEEE, F2A-17-F2A-22.
- EUDAPTICS SOFTWARE 1999-2001. Viscovery SOMine. 4.0 ed.
- FOLK, M. 1981. Review of "Mindstorms: Children, Computers, and Powerful Ideas by Seymour Papert", Basic Books: New York, 1980. *ACM SIGCUE Outlook*, 15, 23-24.
- GADDIS, T. 2011. *Starting out with Alice: A visual introduction to programming*, Boston, Massachusetts, Addison-Wesley, Pearson Education, Inc.
- GÁLVEZ, J., GUZMÁN, E. & CONEJO, R. 2009. A blended E-learning experience in a course of object oriented programming fundamentals. *Knowledge-Based Systems*, 22, 279-286.
- GÁRCIA-MATEOS, G. & FERNÁNDEZ-ALEMÁN, J. L. A course on algorithms and data structures using on-line judging. ITiCSE '09: Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education, July 2009 Paris, France. ACM, 45-49.
- GEORGE, J. F., BATRA, D., VALACICH, J. S. & HOFFER, J. A. 2004. *Object-oriented systems analysis and design*, Upper Saddle River, New Jersey, Prentice Hall.
- GERHARDT-POWALS, J. 1996 Cognitive engineering principles for enhancing human-computer performance. *International Journal of Human-Computer Interaction*, 8, 189-211.
- GIORDANO, J. C. & CARLISLE, M. Toward a more effective visualization tool to teach novice programmers. SIGITE '06: Proceedings of the 7th conference on Information technology education, 19–21 October 2006 Minneapolis, Minnesota, USA. ACM, 115-122.
- GLINERT, E. P. (ed.) 1990. *Visual programming environments: Paradigms and systems*, Los Alamitos, CA, USA: IEEE Computer Society Press.
- GOMES, A. & MENDES, A. J. An environment to improve programming education. CompSysTech '07: Proceedings of the 2007 international conference on Computer systems and technologies, June 2007. ACM, IV.19-1-IV.19-6.

- GUEST, G., MACQUEEN, K. M. & NAMEY, E. E. 2012. *Applied thematic analysis*, Thousand Oaks, California, Sage Publications, Inc.
- HADJERROUIT, S. A constructivist approach to object-oriented design and programming. ITiCSE '99: Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education, June 1999 Cracow, Poland. ACM, 171-174.
- HADJERROUIT, S. 2008. Towards a blended learning model for teaching and learning computer programming: A case study. *Informatics in Education*, 7, 181-210.
- HAMER, J., LUXTON-REILLY, A., PURCHASE, H. C. & SHEARD, J. 2011. Tools for contributing student learning. *ACM Inroads*, 2, 78-91.
- HARVARD-MIT DATA CENTER. *Guide to SPSS* [Online]. Available: http://www.hmdc.harvard.edu/projects/SPSS_Tutorial/spsstut.shtml.
- HAVENGA, M., DE VILLIERS, M. R. & MENTZ, E. 2011. Thinking processes used by high-performing students in a computer programming task. *TD The Journal for Transdisciplinary Research in Southern Africa*, 7, 25-40.
- HAVENGA, M., MENTZ, E. & DE VILLIERS, R. 2008. Knowledge, skills and strategies for successful object-oriented programming: A proposed learning repertoire. *South African Computing Journal*, 42, 1-8.
- HERBERT, C. W. 2011. *An introduction to programming using Alice 2.2*, United States of America, Course Technology, Cengage Learning.
- HTIKE, K. K. & KHALIFA, O. O. Comparison of supervised and unsupervised learning classifiers for human posture recognition. ICCCE '10: Proceedings of the International Conference on Computer and Communication Engineering, 11-13 May 2010 Kuala Lumpur, Malaysia. IEEE, 1-6.
- INTRODUCTION TO SAS. UCLA: ACADEMIC TECHNOLOGY SERVICES STATISTICAL CONSULTING GROUP. *Cronbachs Alpha* [Online]. Available: www.ats.ucla.edu/stat/sas/notes2/ [Accessed 24 November 2007].
- IRAN-NEJAD, A. & HOMAIFAR, A. Biofunctional learning and performance. Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, 10-12 October 2005. IEEE, 411-417 Vol. 1.
- JENKINS, T. The motivation of students of programming. ITiCSE '01: Proceedings of the 6th annual conference on Innovation and technology in computer science education, June 2001 Canterbury, UK. ACM, 53-56.
- JOHNSGARD, K. & MCDONALD, J. Using Alice in overview courses to improve success rates in Programming 1. CSEET '08: Proceedings of the 21st Conference on Software Engineering Education and Training, 14-17 April 2008. IEEE, 129-136.
- JOINT TASK FORCE ON COMPUTING CURRICULA. IEEE 2001. Computing Curricula 2001: Computer Science. *ACM Journal of Educational Resources in Computing (JERIC)*, 1, 1-236.

- JONES, M. E., KISTHARDT, M. & COOPER, M. A. Interdisciplinary teaching: Introductory programming via creative writing. SIGCSE '11: Proceedings of the 42nd ACM technical symposium on Computer science education, 09-12 March 2011 Dallas, Texas, USA. ACM, 523-528.
- KATO, Y. Splish: A visual programming environment for arduino to accelerate physical computing experiences. C5 '10: Proceedings of the Eighth International Conference on Creating Connecting and Collaborating through Computing, 25-28 January 2010 La Jolla, CA. IEEE, 3-10.
- KAUČIČ, B. & ASIČ, T. Improving introductory programming with Scratch? MIPRO '11: Proceedings of the 34th International Convention, 23-27 May 2011 Opatija, Croatia. IEEE, 1095-1100.
- KELLEHER, C. & PAUSCH, R. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)*, 37, 83-137.
- KELLEHER, C. & PAUSCH, R. Lessons learned from designing a programming system to support middle school girls creating animated stories. VL/HCC '06: Proceedings of the Symposium on Visual Languages and Human-Centric Computing, 4-8 September 2006. IEEE, 165-172.
- KERR, A. W., HALL, H. K. & KOZUB, S. A. 2002. *Doing statistics with SPSS*, London, Sage Publications Ltd.
- KIESMUELLER, U., SOSSALLA, S., BRINDA, T. & RIEDHAMMER, K. Online identification of learner problem solving strategies using pattern recognition methods. ITiCSE '10: Proceedings of the fifteenth annual conference on Innovation and technology in computer science education, 26-30 June 2010 Bilkent, Ankara, Turkey. ACM, 274-278.
- KIESMÜLLER, U. 2009. Diagnosing learners' problem-solving strategies using learning environments with algorithmic problems in secondary education. *ACM Transactions on Computing Education (TOCE)*, 9, 17.1-17.26.
- KLASSNER, F. A case study of Lego Mindstorms suitability for artificial intelligence and robotics courses at the college level. SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education, 27 February - 3 March 2002 Covington, Kentucky, USA. ACM, 8-12.
- KOENEMANN, J. & ROBERTSON, S. P. Expert problem solving strategies for program comprehension. CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems: Reaching through technology, March 1991. ACM, 125-130.
- KOLLER, A. & KRUIJFF, G. M. Talking robots with Lego MindStorms. COLING '04: Proceedings of the 20th international conference on Computational Linguistics, August 2004 Geneva, Switzerland. Association for Computational Linguistics.
- KÖLLING, M. & ROSENBERG, J. Guidelines for teaching object orientation with Java. ITiCSE '01: Proceedings of the 6th annual conference on Innovation and technology in computer science education, June 2001 Canterbury. ACM, 33-36.

- KORDAKI, M. 2010. A drawing and multi-representational computer environment for beginners' learning of programming using C: Design and pilot formative evaluation. *Computers & Education*, 54, 69-87.
- LARASON, K. 1995. Using Karel the robot as a classroom motivator. *3C ON-LINE* [Online], 2.
- LAW, K. M. Y., LEE, V. C. S. & YU, Y. T. 2010. Learning motivation in e-learning facilitated computer programming courses. *Computers & Education*, 55, 218-228.
- LEE, Y. 2011. Empowering teachers to create educational software: A constructivist approach utilizing Etoys, pair programming and cognitive apprenticeship. *Computers & Education*, 56, 527-538.
- LIND, D. A., MARCHAL, W. G. & MASON, R. D. 2002. *Statistical techniques in business & economics*, New York, McGraw-Hill.
- LUI, A. K., NG, S. C., CHEUNG, Y. H. Y. & GURUNG, P. 2010. Facilitating independent learning with Lego Mindstorms robots. *ACM Inroads*. ACM.
- MALONEY, J., BURD, L., KAFAI, Y., RUSK, N., SILVERMAN, B. & RESNICK, M. Scratch: A sneak preview. C5 '04: Proceedings of the Second International Conference on Creating, Connecting and Collaborating through Computing, 29-30 January 2004. IEEE, 104-109.
- MASON, R., COOPER, G. & COMBER, T. 2011. Girls get it. *ACM Inroads*, 2, 71-77.
- MATHISON, S. 1988. Why Triangulate? *Educational Researcher*, 17, 13-17.
- MATTHEWS, R., HIN, H. S. & CHOO, K. A. Multimedia learning object to build cognitive understanding in learning introductory programming. MoMM '09: Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia, 14-16 December 2009 Kuala Lumpur, Malaysia. ACM, 396-400.
- MCNALLY, M., GOLDWEBER, M., FAGIN, B. & KLASSNER, F. Do Lego Mindstorms robots have a future in CS education? SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education, 1-5 March 2006 Houston, Texas, USA. ACM, 61-62.
- MELVILLE, S. & GODDARD, W. 1996. *Research methodology: An introduction for science and engineering students*, Kenwyn, Juta & Co. Ltd.
- MILES, M. B. & HUBERMAN, A. M. 1994. *An expanded sourcebook: Qualitative data analysis*, Thousand Oaks, California, Sage Publications.
- MONTERO, S., DÍAZ, P., DÍEZ, D. & AEDO, I. Dual instructional support materials for introductory object-oriented programming: Classes vs. objects. Proceedings of the IEEE Education Engineering (EDUCON), 14-16 April 2010 Madrid, Spain. IEEE, 1929-1934.
- MOSKAL, B., LURIE, D. & COOPER, S. Evaluating the effectiveness of a new instructional approach. SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education, 3-7 March 2004 Norfolk, Virginia, USA. ACM, 75-79.

- MOUTON, J. 2001. *How to succeed in your master's & doctoral studies: A South African guide and resource book*, Hatfield, Pretoria, Van Schaik Publishers.
- NAGAOKA, K., OSAWA, N., MOCHIZUKI, K., TAKAHASHI, H., NISHINA, E. & SAITO, F. Evaluation of a 3-D visual programming environment in an introductory course of object-oriented programming. FIE '00: Proceedings of the 30th Annual Frontiers in Education Conference, 18-21 October 2000. IEEE, T4C-12 Vol.1.
- NIELSEN, J. Enhancing the explanatory power of usability heuristics. CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems: celebrating interdependence, 24-28 April 1994 Boston, Massachusetts, USA. ACM, 152-158.
- NIELSEN, J. 2003. *Usability Engineering*, San Diego, CA, Academic Press.
- NORMAN, D. 1990. *The design of everyday things*, New York, Doubleday.
- OATES, B. J. 2006. *Researching information systems and computing*, London, Sage Publications Ltd.
- OLIVIER, M. S. 2004. *Information technology research: A practical guide for Computer Science and Informatics*, Hatfield, Pretoria, Van Schaik Publishers.
- OWUSU, K. A., MONNEY, K. A., APPIAH, J. Y. & WILMOT, E. M. 2010. Effects of computer-assisted instruction on performance of senior high school biology students in Ghana. *Computers & Education*, 55, 904-910.
- PAUSCH, R., BURNETTE, T., CAPEHART, A. C., CONWAY, M., COSGROVE, D., DELINE, R., DURBIN, J., GOSSWEILER, R., KOGA, S. & WHITE, J. 1995. A brief architectural overview of Alice, a rapid prototyping system for virtual reality. *IEEE Computer Graphics and Applications* [Online]. Available: <http://www.cs.cmu.edu/~stage3/publications/95/journals/IEEEcomputer/CGandA/paper.html>.
- PHELPS, A. M., EGERT, C. A. & BIERRE, K. J. MUPPETS: Multi-user programming pedagogy for enhancing traditional study: An environment for both upper and lower division students. FIE '05: Proceedings of the 35th Annual Conference Frontiers in Education, 19–22 October 2005 Indianapolis, IN. IEEE, S2H-8-S2H-15.
- QIDWAI, U. A. 2007. A LAMP-Lego experience of motivating minority students to study engineering. *ACM SIGCSE Bulletin*, 39, 41-44.
- QUEVEDO-TORRERO, J. U. Learning theories in computer science education. ITNG '09: Proceedings of the Sixth International Conference on Information Technology: New Generations, 27-29 April 2009. IEEE, 1634-1635.
- RALSTON, A., REILLY, E. D. & HEMMENDINGER, D. 2000. *Encyclopedia of Computer Science*. Fourth ed. London, United Kingdom: Nature publishing group.
- REEVES, T. C. Enhancing the worth of instructional technology research through "Design Experiments" and other development research strategies. International Perspectives on Instructional Technology Research for the 21st Century, 27 April 2000 New Orleans, LA, USA. Annual Meeting of the American Educational Research Association, 1-15.

- REILLY, E. D. 2004. Concise encyclopedia of Computer Science. Chichester, England: John Wiley and sons, Ltd.
- ROB, P. & CORONEL, C. 1997. *Database systems: Design, implementation and management*, Cambridge, MA, USA, Course Technology PTR.
- ROMERO, C., VENTURA, S. & GARCÍA, E. 2008. Data mining in course management systems: Moodle case study and tutorial. *Computers & Education*, 51, 368-384.
- ROSNOW, R. L. & ROSENTHAL, R. 1996. *Beginning behavioral research: A conceptual primer*, Upper Saddle River, New Jersey, Prentice Hall.
- SAJANIEMI, J., KUITTINEN, M. & TIKANSALO, T. 2008. A study of the development of students' visualizations of program state during an elementary object-oriented programming course. *ACM Journal on Educational Resources in Computing (JERIC)*, 7, 3:1-3:31.
- SALIM, A., HASSAN, S., HAMDY, S., YOUSSEF, S., ADEL, H., KHATTAB, S. & EL-RAMLY, M. On using 3D animation for teaching computer programming in Cairo University. INFOS '10: Proceedings of the 7th International Conference on Informatics and Systems, 28-30 March 2010 Cairo. IEEE, 1-5.
- SANDOVAL-REYES, S., GALICIA-GALICIA, P. & GUTIERREZ-SANCHEZ, I. Visual learning environments for computer programming. CERMA '11: Proceedings of the Electronics, Robotics and Automotive Mechanics Conference, 15-18 November 2011. IEEE, 439-444.
- SPSS. *Data analytics and reporting with IBM SPSS* [Online]. Available: <http://www.spss.co.in/>.
- SPSS SOFTWARE 2008. Cronbach's Alpha. 17.0 ed.
- STARR, C. W., MANARIS, B. & STALVEY, R. H. Bloom's taxonomy revisited: specifying assessable learning objectives in computer science. SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education, 12-15 March 2008 Portland, Oregon, USA. ACM, 261-265.
- STEYN, A. G. W., SMIT, C. F., DU TOIT, S. H. C. & STRASHEIM, C. 1994. *Modern statistics in practice*, Pretoria, J.L. van Schaik Publishers.
- STILLER, E. 2009. Teaching programming using bricolage. *Journal of Computing Sciences in Colleges*, 24, 35-42.
- STOREY, M. 2006. Theories, tools and research methods in program comprehension: past, present and future. *Software Quality Journal*, 14, 187-208.
- SWELLER, J. 1994. Cognitive load theory, learning difficulty, and instructional design. *Learning and Instruction*, 4, 295-312.
- VICKERS, P. 2009. *How to think like a programmer: Program design solutions for the bewildered*, United Kingdom, Cengage Learning.
- VISCOVERY. *Viscovery SOMine* [Online]. Available: <http://www.eudaptics.com/somine/index.php?sprache=en> [Accessed 14 September 2004].

- WANG, T., MEI, W., LIN, S., CHIU, S. & LIN, J. M. Teaching programming concepts to high school students with Alice. FIE '09: Proceedings of the 39th IEEE Frontiers in Education Conference, 18-21 October 2009 San Antonio, TX. IEEE, T3H-1-T3H-6.
- WEBB, H. C. & ROSSON, M. B. Exploring careers while learning *Alice* 3D: A summer camp for middle school girls. SIGCSE '11, 09-12 March 2011 Dallas, Texas, USA. ACM, 377-382.
- WILLEMSE, I. 2009. *Statistical methods and calculation skills*, Cape Town, Juta and Co Ltd.
- WINSLOW, L. A structured introductory computer science course. SIGCSE '77: Proceedings of the seventh SIGCSE technical symposium on Computer science education, February 1977. ACM, 165-167.
- WINSLOW, L. E. 1996. Programming pedagogy-a psychological overview. *ACM SIGCSE Bulletin*, 28, 17-22.
- WRIGHT, T. & COCKBURN, A. Writing, reading, watching: A task-based analysis and review of learners' programming environments. IWALT '00: Proceedings of the International Workshop on Advanced Learning Technologies, 2000. IEEE, 167-170.
- WULF, T. Constructivist approaches for teaching computer programming. SIGITE '05: Proceedings of the 6th conference on Information technology education, 20-22 October 2005 Newark, New Jersey, USA. ACM, 245-248.
- YIN, R. K. 2003. *Case study research: Design and methods*, Thousand Oaks, CA, Sage Publications.
- YU, P. & YANG, L. Programming skills training in programming language courses. ICEIT '10: Proceedings of the International Conference on Educational and Information Technology, 2010. IEEE, V3-14-V3-16.
- YU, S., LI, H., XU, Q. & WU, X. Fuzzy self-organizing maps for data mining with incomplete data sets. ICCASM '10: Proceedings of the International Conference on Computer Application and System Modeling, 22-24 October 2010. IEEE, V14-336-V14-340.
- ZACCONE, R., COOPER, S. & DANN, W. Using 3D animation programming in a core engineering course seminar. FIE '03: Proceedings of the 33rd Annual Frontiers in Education, 5-8 November 2003 Boulder, CO. IEEE, F4D-14-F4D-17 Vol. 2.

Appendix A – Ethical Clearance

A.1 Ethical Clearance – Durban University of Technology



D U R B A N
UNIVERSITY of
TECHNOLOGY

*Research Management and Development
Durban University of Technology
Tromso Annexe, Steve Biko Campus
P.O. Box 1334, Durban 4000
Tel.: 031-3732576/7
Fax: 031-3732946
E-mail: moyos@dut.ac.za*

20th April 2011

Ms. J. Anniroot
c/o Department of Information Technology
Durban University of Technology

Dear Ms. Anniroot

PERMISSION TO CONDUCT RESEARCH AT THE DUT

Your email correspondence dated 14th April 2011 in respect of the above refers. I am pleased to inform you that the Institutional Research Committee (IRC) will grant permission to you to conduct your research at the Durban University of Technology. However, kindly note that the committee requires you to provide proof of ethical clearance prior to you commencing with your research at the DUT.

We would be grateful if a summary of your key research findings can be submitted to the IRC on completion of your studies.

Kindest regards.
Yours sincerely

A handwritten signature in black ink, appearing to read 'S. Moyo'.

PROF. S. MOYO
DIRECTOR: RESEARCH MANAGEMENT AND DEVELOPMENT (ACTING)

cc.: Prof. P. Singh - Department of Information and Corporate Management



A.2 Ethical Clearance – University of South Africa



Ms J Anniroot (3371-639-0)
School of Computing (Student)
UNISA
Pretoria

Date

Permission to conduct MSc(Information Systems) research project
Ref:014/JA/2011

The request for ethical approval for your MSc (Information Systems) research project entitled "Alice: A visual programming environment for teaching object-oriented programming" refers.

The College of Science, Engineering and Technology's (CSET) Research and Ethics Committee (CREC) has considered the relevant parts of the studies relating to the abovementioned research project and research methodology and is pleased to inform you that ethical clearance is granted for your study as set out in your proposal and application for ethical clearance.

Therefore, involved parties may also consider ethics approval as granted. However, the permission granted must not be misconstrued as constituting an instruction from the CSET Executive or the CSET CREC that sampled interviewees (if applicable) are compelled to take part in the research project. All interviewees retain their individual right to decide whether to participate or not.

We trust that the research will be undertaken in a manner that is respectful of the rights and integrity of those who volunteer to participate, as stipulated in the UNISA Research Ethics policy. The policy can be found at the following URL:

http://cm.unisa.ac.za/contents/departments/res_policies/docs/ResearchEthicsPolicy_apprvCounc_21Sept07.pdf

Please note that if you subsequently do a follow-up study that requires the use of a different research instrument, you will have to submit an addendum to this application, explaining the purpose of the follow-up study and attach the new instrument along with a comprehensive information document and consent form.

Yours sincerely

A handwritten signature in black ink, appearing to read "H H Lotriet".

Prof HH Lotriet

Acting Chair: School of Computing Ethics Sub-Committee



University of South Africa
College of Science, Engineering and Technology
Preller Street, Muckleneuk Ridge, City of Tshwane
PO Box 392 UNISA 0003 South Africa
Telephone + 27 12 429 6122 Facsimile + 27 12 429 6848
www.unisa.ac.za/cset

Appendix B – Case Study 1 Documentation

B.1 Learner consent form – *Alice* workshop and questionnaire



Learner Consent Form:

An evaluation of the *Alice* visual programming environment to be undertaken by the Development Software 2 learners at the Durban University of Technology

Background information about the research study:

The researcher, Miss Jeraline Anniroot, is a lecturer at the Department of Information Technology from the Durban University of Technology (DUT). She is currently studying towards a Masters in Science degree in Information Systems at the University of South Africa (UNISA) under the supervision of Prof Ruth de Villiers of UNISA's School of Computing.

There are several challenges facing novice programming learners, including: the lack of motivation for programming; having to deal with complex syntax and semantics of a programming language; the need for immediate feedback; the difficulty in understanding compound logic; and being able to apply algorithmic problem-solving skills. This study proposes the use of a 3D visual programming environment called *Alice* to help improve the understanding of fundamental programming concepts and object-oriented programming (OOP). This will be achieved through a 3-to-4 week workshop and will be conducted by means of hands-on practical interaction with the *Alice* software installed in the Department of Information Technology labs. Learners will be taught how to design simple video games and animated movies.

The study is conducted with learners registered for the subject Development Software 2 (DS2). The benefit of participating in this workshop is that learners can closely associate the *Alice* software to the corresponding course work covered in DS2. The sections on OOP will be lectured using Microsoft C#.Net and supplemented with the knowledge gained from the teaching of OOP through

the 3D *Alice* visual programming environment. The intention of this research initiative is to improve the performance of learners and to help them better understand the concepts of OOP.

Learners were selected based on criteria which indicate a first-time exposure to OOP. Learners ought to have passed all first year modules at first attempt during their first year of registration in the ND: IT programme and should be attempting all second year modules for the first time.

Any queries or questions may be directed to Miss Jeraline Annroot, either personally, via email or via telephonic discussion.

Email: annrootj@dut.ac.za

Telephone number: (031) 373 5583

Learner Consent:

I, _____ (First Name(s) and Surname) and student number _____, a learner at the Durban University of Technology, state that I have not been put under any pressure to participate in the *Alice* workshop and this evaluation exercise. I have willingly agreed to participate in it. I acknowledge that this study guarantees my confidentiality and anonymity, together with the assurance that I can withdraw at any time. I have been requested to complete a questionnaire on completion of the *Alice* workshop, and have agreed to do so.

I understand that the findings of this evaluation will be used for the purpose of research and that the findings may be published in academic publications.

- My name will not be published or disclosed.
- My contributed input will be used purely for academic purposes.

I am aware that following the completion of the study, data will be stored securely on a CD or an alternative secondary storage medium, which will be kept by the researcher for a period of five years.

Signed

Date

B.2 Questionnaire



Questionnaire:

An evaluation of the *Alice* visual programming environment to be undertaken by the Development Software 2 learners at the Durban University of Technology

Note: All the information you provide in this questionnaire is confidential and will only be used for research purposes.

Please mark your responses with an X in the appropriate blocks.

General Details										
Student Number:					Gender:	Male		Female		
Surname:										
First Name(s):										
Race:	Black			White			Asian		Coloured	
First year subjects passed at first attempt:	DS1			TP1			IS1		SS1	
Registered for these second year subjects:	DS2			TP2			IS2		SS2	

Please mark your response with an X. Options are from left to right SA=STRONGLY AGREE, A =AGREE, N =NEUTRAL, D =DISAGREE, SD =STRONGLY DISAGREE

General interface design heuristics, based on Nielsen (1994) and (Dix <i>et al.</i> , 2004:325) used to assess the usability of the <i>Alice</i> visual programming environment						
	Criteria	Rating				
1	<i>Visibility of the system status</i>	SA	A	N	D	SD
	1.1 I am always aware of what is going on in the system.					
	1.2 Whenever <i>Alice</i> is opened, the system indicates that it is in the process of loading the software.					
	1.3 When I save a world in <i>Alice</i> , the system indicates that files are being saved.					
2	<i>Match between the system and the real world</i>	SA	A	N	D	SD
	2.1 The system uses words, terms and phrases that I can easily understand.					
	2.2 The templates used for new worlds and the objects in the system gallery, relate to real-world objects that I encounter in my day-to-day experiences.					

3	<i>User control and freedom</i>	SA	A	N	D	SD
	3.1 I am comfortable with the level of control that I have over the system.					
	3.2 <i>Alice</i> allows me the flexibility to use the environment to perform a task.					
	3.3 I can recover from mistakes quickly and easily.					
4	<i>Consistency and standards</i>	SA	A	N	D	SD
	4.1 The <i>Alice</i> interface maintains a consistent look and feel.					
	4.2 The startup dialog box, play button, main menu and tab controls are clearly and consistently displayed.					
	4.3 The use of the mouse controls buttons and camera navigation controls are always available to move and arrange objects.					
5	<i>Error prevention</i>	SA	A	N	D	SD
	5.1 The <i>Alice</i> software allows me to focus on the objects and their associated properties and methods, rather than dealing with complex syntax.					
	5.2 The <i>Alice</i> interface design causes me to make errors.					
6	<i>Recognition rather than recall</i>	SA	A	N	D	SD
	6.1 The actions to be taken and options available for selection are clear and visible at all times.					
	6.2 I have to remember the information from a previous screen in order to proceed with the next one.					
7	<i>Flexibility and efficiency of use</i>	SA	A	N	D	SD
	7.1 <i>Alice</i> caters for novice to expert users.					
	7.2 I often use shortcut keys and/or combination of keyboard and mouse use to control the movement of objects.					
	7.3 I often use both the single view and the quad view to move, align and reposition objects on the screen.					
8	<i>Aesthetic and minimalist design</i>	SA	A	N	D	SD
	8.1 There is irrelevant information in the <i>Alice</i> interface design that distracts me and slows me down.					
9	<i>Recognition, diagnosis and recovery from errors</i>	SA	A	N	D	SD
	9.1 <i>Alice</i> crashes while I'm using it.					
	9.2 In cases where I encounter system errors, the system provides an appropriate error message in simple language.					
10	<i>Help and documentation</i>	SA	A	N	D	SD
	10.1 The four tutorials in the startup dialog box are useful in helping me to learn how to use <i>Alice</i> .					
	10.2 The example worlds in the startup dialog box are useful.					
The teaching and learning of programming						
11	<i>Learner attrition</i>	SA	A	N	D	SD
	11.1 I had a high level of comfort with computer usage prior to studying programming.					
	11.2 I am able to cope with the breadth and depth of the course work covered in the programming subjects of this course.					
	11.3 The current teaching methods and techniques used in programming subjects helped me to program successfully.					
	11.4 I had a high level of comfort with object-oriented programming prior to the <i>Alice</i> workshop.					

	11.5	I like to learn object-oriented programming in a more fun and interactive way.					
Challenges faced by learners in learning object-oriented programming							
12	<i>Lack of motivation for programming</i>		SA	A	N	D	SD
	12.1	I am motivated to learn programming.					
	12.2	I spend a lot of time intensively practicing programming exercises.					
13	<i>Fragile mechanics of program creation, particularly syntax</i>		SA	A	N	D	SD
	13.1	Learning the syntax and semantics of a programming language is challenging.					
	13.2	It would be easier for me to learn how to solve a problem and learn basic concepts of object-orientation without having to deal with brackets, commas and semicolons.					
	13.3	I have felt intimidated by direct exposure to programming syntax, such as do-while, if-then-else, switch-case, and for-next.					
	13.4	The textual nature of the conventional programming environments I use, makes it easy for me to learn how to program.					
14	<i>Identifying results of computation as the program runs</i>		SA	A	N	D	SD
	14.1	I am able to identify errors and correct them using the feedback given by the program.					
	14.2	I am able to work independently on a program, from coding to testing.					
	14.3	My experience from running and debugging programs allows me to solve similar problems.					
15	<i>Difficulty of understanding compound logic</i>		SA	A	N	D	SD
	15.1	I am able to apply basic problem-solving techniques to create algorithms that solve problems.					
	15.2	I have a poor understanding of pseudocode.					
	15.3	I use pseudocode to outline and understand the logic of a program before I start coding.					
	15.4	I am able to break down a large and complex programming task into smaller subtasks.					
How to improve the teaching of object-oriented programming							
16	<i>Algorithmic thinking and expression</i>		SA	A	N	D	SD
	16.1	I am able to read and write in a formal language.					
17	<i>Abstraction</i>		SA	A	N	D	SD
	17.1	I am able to use creative thinking and programming concepts to write programs.					
	17.2	I sometimes write a program without understanding every line of code.					
18	<i>Objects-first strategy</i>		SA	A	N	D	SD
	18.1	I have a good level of understanding of objects, gained from my first year of study.					
	18.2	It would be easier to learn object-oriented programming during the first year of study, and later learn other control structures such as loops, If statements etc.					
	18.3	<i>Alice</i> helps me to see everything as an object.					

19	<i>3D animation authoring tools and visualisation</i>	SA	A	N	D	SD
	19.1 A visual representation improves my understanding of programming concepts.					
	19.2 The visual effects in <i>Alice</i> provide a meaningful context for understanding classes, objects, methods, and events.					
	19.3 Three-dimensionality makes objects seem real.					
	19.4 I am able to use <i>Alice</i> to write a new method to make objects perform animated tasks, such as hop, fly, swim etc.					
Criteria from the Researcher's experience						
20	<i>Appreciation of trial and error</i>	SA	A	N	D	SD
	20.1 When using <i>Alice</i> , I use trial and error to 'try out' individual animation instructions as I create new methods.					
	20.2 I cannot visibly see the effect that each new animation instruction has on the animation.					
21	<i>Incremental construction approach</i>	SA	A	N	D	SD
	21.1 <i>Alice</i> has taught me how to program incrementally i.e. I write one method at a time, testing and running each piece.					
22	<i>Alice has improved my understanding of these OOP concepts:</i>	SA	A	N	D	SD
	22.1 Inheritance					
	22.2 Methods					
	22.3 Properties					
	22.4 Functions					
23	<i>Alice has improved my understanding of these basic programming concepts:</i>	SA	A	N	D	SD
	23.1 Loops					
	23.2 If..statements					
	23.3 Data types					
	23.4 Event-driven programming					
24	<i>Ability to collaborate</i>	SA	A	N	D	SD
	24.1 The <i>Alice</i> workshop has provided the opportunity to work in pairs with other learners and I have chosen to do so.					
	24.2 I have been able to interact and communicate well with other learners.					
	24.3 The experience of working with other learners has helped me to learn the <i>Alice</i> programming environment.					
25	<i>Impact of Alice on DS2 learners</i>	SA	A	N	D	SD
	25.1 The <i>Alice</i> workshop relates directly to the sections on object-oriented programming covered in the Development Software 2 syllabus.					
	25.2 I am interested in learning more about computer graphics and animation.					
	25.3 I am interested in learning and working more with the <i>Alice</i> visual programming environment.					
	25.4 I used <i>Alice</i> during my personal time after attending the first lesson of the <i>Alice</i> workshop.					
	25.5 I am interested in learning more about object-oriented programming.					

26	Write down your answers to the following questions:
	26.1 Do you like <i>Alice</i> ? Explain your experience with using <i>Alice</i> ?
	26.2 Has working with <i>Alice</i> improved your understanding of object-oriented programming? Give reasons for your answer above.
	26.3 Name the problems you encountered in using <i>Alice</i> .

	<p>26.4 Briefly outline the challenges that you face in learning object-oriented programming.</p>
	<p>26.5 Do you agree that <i>Alice</i> as a visual programming environment can help address some of these challenges, and why.</p>
	<p>26.6 Explain what changes you would like to see in improving the teaching of object-oriented programming.</p>

Thank you for your participation in completing this questionnaire.

The **Alice** Workshop

An exciting opportunity for **Development Software 2 Learners** to learn **Object-Oriented Programming** in a fun and interactive way using a 3D visual programming environment called *Alice*. Here's your chance to enhance your learning of fundamental programming concepts while creating animated movies and simple video games.

This is a research initiative aimed at helping learners better understand the complex concepts of OOP. There will therefore be **No exams** and **No tests**. The top three projects will be awarded a **special prize** each.



Only the learners who appear on the list below are eligible to attend this *Alice* workshop. Only **55** learners will be allowed to attend, on a first come-first serve basis. Learners must meet **Miss J. Annirroot** to confirm their place, details follow:

Date: **09 March 2011**

Venue: **RNO-1**

Time: **12h00 (Lunch Break)**

The **Alice** Workshop

ATTENTION : DS2 Learners

The 55 DS2 learners listed below are **FINALISED** attendees to the *Alice* workshop. Each learner has indicated their willingness to participate in the workshop. Any learner who **CANNOT ATTEND** must see Miss J. Anniroot by 14:00 on Monday, 14 March 2011, so that your place can be offered to another learner.

The workshop will commence tomorrow, details follow:

Date: **15 March 2011**

Venue: **Lab 4**

Time: **12h00 (Lunch Break)**



"You're not the same as you were before. You were much more muchier; you've lost your muchness." -Mad Hatter "

My muchness?" - Alice

"In there (your heart/core), something is missing." -Mad Hatter

Appendix C – Case Study 2 Documentation

C.1 Learner consent form – *Alice* workshop and questionnaire



Learner Consent Form:

(*Alice* Workshop and Questionnaire)

An evaluation of the *Alice* visual programming environment to be undertaken by the Development Software 2 learners at the Durban University of Technology

Background information about the research study:

The researcher, Miss J. Anniroot, is a lecturer at the Department of Information Technology (IT) from the Durban University of Technology (DUT). She is currently studying towards a Masters in Science degree in Information Systems at the University of South Africa (UNISA) under the supervision of Professor M.R. (Ruth) de Villiers of UNISA's School of Computing.

There are several challenges facing novice programming learners, including: the lack of motivation for programming; having to deal with complex syntax and semantics of a programming language; the need for immediate feedback; the difficulty in understanding compound logic; and being able to apply algorithmic problem-solving skills. This study proposes the use of a 3D visual programming environment called *Alice* to help improve the understanding of fundamental programming concepts and object-oriented programming (OOP). This will be achieved through a 2 week workshop and will be conducted by means of hands-on practical interaction with the *Alice* software installed in the Department of IT labs. Learners will be taught how to design simple video games and animated movies.

The study is conducted with learners registered for the subject Development Software 2 (DS2). The benefit of participating in this workshop is that learners can closely associate the *Alice* software to the corresponding course work covered in DS2. The sections on OOP will be lectured using Microsoft C#.Net and supplemented with the knowledge gained from the teaching of OOP through

the 3D *Alice* visual programming environment. The intention of this research initiative is to improve the performance of learners and to help them better understand the concepts of OOP.

Learners were selected based on criteria which indicate a first-time exposure to OOP. Learners ought to have passed both programming subjects (Development Software 1 and Technical Programming 1) together with one of the theoretical subjects (Information Systems 1 or Systems Software 1) at first attempt during their first year of registration in the National Diploma: IT programme.

The questionnaire will be used to obtain responses from the learners in relation to the effectiveness of using the *Alice* visual environment in addressing the challenges facing novice programming learners within the object-oriented domain. The questionnaire will elicit closed-ended and open-ended questions.

Any queries or questions may be directed to Miss J. Annirroot, either personally, via email (annirrootj@dut.ac.za) or via telephonic discussion ((031) 373 5583).

Learner Consent (Questionnaire Evaluation):

I, _____ (first name(s) and surname) and student number _____, a learner at the Durban University of Technology, state that I have not been put under any pressure to participate in the *Alice* workshop and this evaluation exercise. I have willingly agreed to participate in it. I acknowledge that this study guarantees my confidentiality and anonymity, together with the assurance that I can withdraw at any time. I have been requested to complete a questionnaire on completion of the *Alice* workshop, and have agreed to do so.

I understand that the findings of this evaluation will be used for the purpose of research and that the findings may be published in academic publications.

- My name will not be published or disclosed.
- My contributed input will be used purely for academic purposes.

I am aware that following the completion of the study, data will be stored securely on a CD or an alternative secondary storage medium, which will be kept by the researcher for a period of five years.

Signature: _____

Date: _____

Print Name: _____

C.2 Learner consent form – Interview



Learner Consent Form: (Interview)

**An evaluation of the *Alice* visual programming environment to be undertaken
by the Development Software 2 learners at the Durban University of Technology**

Background information about the research study:

The researcher, Miss J. Anniroot, is a lecturer at the Department of Information Technology (IT) from the Durban University of Technology (DUT). She is currently studying towards a Masters in Science degree in Information Systems at the University of South Africa (UNISA) under the supervision of Professor M.R. (Ruth) de Villiers of UNISA's School of Computing.

There are several challenges facing novice programming learners, including: the lack of motivation for programming; having to deal with complex syntax and semantics of a programming language; the need for immediate feedback; the difficulty in understanding compound logic; and being able to apply algorithmic problem-solving skills. This study proposes the use of a 3D visual programming environment called *Alice* to help improve the understanding of fundamental programming concepts and object-oriented programming (OOP). This will be achieved through a 2 week workshop and will be conducted by means of hands-on practical interaction with the *Alice* software installed in the Department of IT labs. Learners will be taught how to design simple video games and animated movies.

The study is conducted with learners registered for the subject Development Software 2 (DS2). The benefit of participating in this workshop is that learners can closely associate the *Alice* software to the corresponding course work covered in DS2. The sections on OOP will be lectured using Microsoft C#.Net and supplemented with the knowledge gained from the teaching of OOP through the 3D *Alice* visual programming environment. The intention of this research initiative is to improve the performance of learners and to help them better understand the concepts of OOP.

Learners were selected based on criteria which indicate a first-time exposure to OOP. Learners ought to have passed both programming subjects (Development Software 1 and Technical Programming 1) together with one of the theoretical subjects (Information Systems 1 or Systems Software 1) at first attempt during their first year of registration in the National Diploma: IT programme.

Semi-structured interviews will be conducted as a follow-up to the questionnaire evaluation, with a selected subsample of learners. This will allow the researcher to probe more detailed themes and aspects from the learners. A set of core questions will be asked to all learners and each learner will be allowed to further explain their experiences and opinions.

Any queries or questions may be directed to Miss J. Anniroot, either personally, via email (annirootj@dut.ac.za) or via telephonic discussion ((031) 373 5583).

Learner Consent (Interview Evaluation):

I, _____ (first name(s) and surname) and student number _____, a learner at the Durban University of Technology, state that I have not been put under any pressure to participate in the *Alice* workshop and this evaluation exercise. I have willingly agreed to participate in it. I acknowledge that this study guarantees my confidentiality and anonymity, together with the assurance that I can refuse to answer any questions and can terminate the interview at any time.

I understand that the findings of this evaluation will be used for the purpose of research and that the findings may be published in academic publications.

- My name will not be published or disclosed.
- My contributed input will be used purely for academic purposes.

I will grant permission to the researcher to tape-record the session. Yes No

I am aware that following the completion of the study, data will be stored securely on a CD or an alternative secondary storage medium, which will be kept by the researcher for a period of five years.

Signature: _____

Date: _____

Print Name: _____

C.3 Questionnaire



Questionnaire

An evaluation of the *Alice* visual programming environment to be undertaken by the Development Software 2 learners at the Durban University of Technology

Note: All the information you provide in this questionnaire is confidential and will only be used for research purposes.

Please mark your responses with an X in the appropriate blocks.

General Details										
Student Number:			Gender:		Male		Female			
Surname:				Class Group:						
First Name(s):										
Email:				Contact Tele:						
Age:		Race:	Black		White		Asian		Coloured	

Please mark your response with an X. Options are from left to right SA=STRONGLY AGREE, A =AGREE, N =NEUTRAL, D =DISAGREE, SD =STRONGLY DISAGREE

General interface design heuristics, based on Nielsen (1994) and (Dix <i>et al.</i> , 2004:325) used to assess the usability of the <i>Alice</i> visual programming environment						
	Criteria	Rating				
1	<i>Visibility of the system status</i>	SA	A	N	D	SD
	1.2 I am always aware of what is going on in the system.					
	1.2 Whenever <i>Alice</i> is opened, the system indicates that it is in the process of loading the software.					
	1.3 When I save a world in <i>Alice</i> , the system indicates that files are being saved.					
2	<i>Match between the system and the real world</i>	SA	A	N	D	SD
	2.1 The system uses words, terms and phrases that I can easily understand.					
	2.2 The templates used for new worlds and the objects in the system gallery, relate to real-world objects that I encounter in my day-to-day experiences.					

3	<i>User control and freedom</i>	SA	A	N	D	SD
	3.1 I am comfortable with the level of control that I have over the system.					
	3.2 <i>Alice</i> allows me the flexibility to use the environment to perform a task.					
4	<i>Consistency and standards</i>	SA	A	N	D	SD
	4.1 The <i>Alice</i> interface maintains a consistent look and feel.					
	4.2 The startup dialog box, play button, main menu and tab controls are clearly and consistently displayed.					
	4.3 The use of the mouse controls buttons and camera navigation controls are always available to move and arrange objects.					
5	<i>Error prevention</i>	SA	A	N	D	SD
	5.1 The <i>Alice</i> software always gives error messages to prevent errors from occurring.					
	5.2 The <i>Alice</i> interface design causes me to make errors.					
6	<i>Recognition rather than recall</i>	SA	A	N	D	SD
	6.1 The actions to be taken and options available for selection are clear and visible at all times.					
	6.2 I have to remember the information from a previous screen in order to proceed with the next one.					
7	<i>Flexibility and efficiency of use</i>	SA	A	N	D	SD
	7.1 <i>Alice</i> caters for beginner to expert users.					
	7.2 I often use both the single view and the quad view to move, align and reposition objects on the screen.					
8	<i>Aesthetic and minimalist design</i>	SA	A	N	D	SD
	8.1 There is irrelevant information in the <i>Alice</i> interface design that distracts me and slows me down.					
9	<i>Recognition, diagnosis and recovery from errors</i>	SA	A	N	D	SD
	9.1 <i>Alice</i> crashes while I'm using it.					
	9.2 In cases where I encounter system errors, the system provides an appropriate error message in simple language.					
	9.3 I can recover from mistakes quickly and easily.					
10	<i>Help and documentation</i>	SA	A	N	D	SD
	10.1 The four tutorials in the startup dialog box are useful in helping me to learn how to use <i>Alice</i> .					
	10.2 The example worlds in the startup dialog box are useful.					
The teaching and learning of programming						
11	<i>Learner attrition</i>	SA	A	N	D	SD
	11.1 I had a high level of comfort with computer usage prior to studying programming.					
	11.2 I am able to cope with the breadth and depth of the course work covered in the programming subjects of this course.					
	11.3 The current teaching methods and techniques used in programming subjects helped me to program successfully.					
	11.4 I had a high level of comfort with object-oriented programming prior to the <i>Alice</i> workshop.					
	11.5 I like to learn object-oriented programming in a more fun and interactive way.					

Challenges faced by learners in learning object-oriented programming						
12	<i>Lack of motivation for programming</i>	SA	A	N	D	SD
	12.1 I am motivated to learn programming.					
	12.2 I spend a lot of time intensively practicing programming exercises.					
	12.3 Alice has improved my motivation for programming					
13	<i>Fragile mechanics of program creation, particularly syntax</i>	SA	A	N	D	SD
	13.1 Learning the syntax and semantics of a programming language is challenging.					
	13.2 It would be easier for me to learn how to solve a problem and learn basic concepts of object-orientation without having to deal with brackets, commas and semicolons, as is the case with Alice.					
	13.3 I have felt intimidated by direct exposure to programming syntax, such as do-while, if-then-else, switch-case, and for-next.					
	13.4 The textual nature of the conventional programming languages I use, makes it easy for me to learn how to program.					
14	<i>Identifying results of computation as the program runs</i>	SA	A	N	D	SD
	14.1 I am able to identify errors and correct them using the feedback given by the program.					
	14.2 I am able to work independently on a program, from coding to testing.					
	14.3 My experience from running and debugging programs allows me to solve similar problems.					
	14.4 Alice provides immediate feedback as the program runs.					
15	<i>Difficulty of understanding compound logic</i>	SA	A	N	D	SD
	15.1 I am able to apply basic problem-solving techniques to create algorithms that solve problems.					
	15.2 I have a poor understanding of pseudocode.					
	15.3 I use pseudocode to outline and understand the logic of a program before I start coding.					
	15.4 I am able to break down a large and complex programming task into smaller subtasks.					
	15.5 Alice allows me to focus on problem-solving.					
How to improve the teaching of object-oriented programming						
16	<i>Algorithmic thinking and expression</i>	SA	A	N	D	SD
	16.1 I am able to read and write in a formal language.					
17	<i>Abstraction</i>	SA	A	N	D	SD
	17.1 I am able to use creative thinking and programming concepts to write programs.					
	17.2 I sometimes write a program without understanding every line of code.					
18	<i>Objects-first strategy</i>	SA	A	N	D	SD
	18.1 I have a good level of understanding of objects, gained from my first year of study.					
	18.2 It would be easier to learn object-oriented programming during the first year of study, and later learn other control structures such as loops, If statements etc.					

	18.3	<i>Alice</i> helps me to see everything as an object.					
19	<i>3D animation authoring tools and visualisation</i>		SA	A	N	D	SD
	19.1	A visual representation improves my understanding of programming concepts.					
	19.2	Three-dimensionality makes objects seem real.					
	19.3	The visual effects in <i>Alice</i> provide a meaningful context for understanding classes, objects, methods, and events.					
	19.4	I am able to use <i>Alice</i> to write a new method to make objects perform animated tasks, such as hop, fly, swim etc.					
Criteria from the Researcher's experience							
20	<i>Appreciation of trial and error</i>		SA	A	N	D	SD
	20.1	When using <i>Alice</i> , I use trial and error to 'try out' individual animation instructions as I create new methods.					
	20.2	I cannot visibly see the effect that each new animation instruction has on the animation.					
21	<i>Incremental construction approach</i>		SA	A	N	D	SD
	21.1	<i>Alice</i> has taught me how to program incrementally i.e. I write one method at a time, testing and running each piece.					
22	<i>Alice has improved my understanding of these OOP concepts:</i>		SA	A	N	D	SD
	22.1	Inheritance					
	22.2	Methods					
	22.3	Properties					
	22.4	Functions					
23	<i>Alice has improved my understanding of these basic programming concepts:</i>		SA	A	N	D	SD
	23.1	Loops					
	23.2	If..statements					
	23.3	Data types					
	23.4	Event-driven programming					
24	<i>Ability to collaborate</i>		SA	A	N	D	SD
	24.1	The <i>Alice</i> workshop has provided the opportunity to work in pairs with other learners and I have chosen to do so.					
	24.2	I have been able to interact and communicate well with other learners.					
	24.3	The experience of working with other learners has helped me to learn the <i>Alice</i> programming environment.					
25	<i>Impact of Alice on DS2 learners</i>		SA	A	N	D	SD
	25.1	The <i>Alice</i> workshop relates directly to the sections on object-oriented programming covered in the Development Software 2 syllabus.					
	25.2	I am interested in learning more about computer graphics and animation.					
	25.3	I am interested in learning and working more with the <i>Alice</i> visual programming environment.					
	25.4	I used <i>Alice</i> during my personal time after attending the first lesson of the <i>Alice</i> workshop.					
	25.5	I am interested in learning more about object-oriented programming.					

26	Write down your answers to the following questions:
	26.1 Do you like <i>Alice</i> ? Explain your experience with using <i>Alice</i> ?
	26.2 Has working with <i>Alice</i> improved your understanding of object-oriented programming? Give reasons for your answer above.
	26.3 Name the problems you encountered in using <i>Alice</i> .

	<p>26.4 Briefly outline the challenges that you face in learning object-oriented programming.</p>
	<p>26.5 Do you agree that <i>Alice</i> as a visual programming environment can help address some of these challenges, and why.</p>
	<p>26.6 Explain what changes you would like to see in improving the teaching of object-oriented programming.</p>

Thank you for your participation in completing this questionnaire.



Interview Questions

An evaluation of the *Alice* visual programming environment to be undertaken by the Development Software 2 learners at the Durban University of Technology

Semi-structured Interview Questions	
1	How easy is it to use <i>Alice</i> ? Why?
2	How would you describe your experience with the teaching and learning of programming at DUT? Tell me more.

3	What are the challenges you are faced with in learning object-oriented programming?
4	How would you like to see the teaching of object-oriented programming improve?
5	How has Alice impacted on your understanding OOP and addressing the challenges you mentioned?

Thank you for your participation in this interview.

The **Alice** Workshop

An exciting opportunity for **Development Software 2** learners to learn **Object-Oriented Programming (OOP)** in a **fun** and **interactive** way using a **3D** visual environment called **Alice**. Here's your chance to enhance your learning of fundamental programming concepts while creating **animated movies** and simple **video games**.

This research initiative will be conducted by **Miss J. Anniroot** and aims at helping learners better understand the complex concepts of **OOP**. There will be **No exams** and **No tests**. Learners will be required to **complete a project**. The top three projects will be awarded **book prizes** (First prize is the D52 prescribed textbook, second and third prizes are SQL Server and ASP.NET C# textbooks).

The workshop will take place over a period of two weeks, from the **16 March - 30 March 2012** in **Lab 2**, during the **lunch breaks**. **Refreshments** will be provided. Learners who **attend every lesson** will receive a **DUT certificate of attendance** issued by DUTs' **Enterprise Development Unit**, and will be useful for your curriculum vitae.



Only the learners who appear on the list below are eligible to attend this **Alice** workshop. Only **55** learners will be allowed to attend, on a first come-first serve basis. Learners who are interested in attending the two-week workshop must meet **Miss J. Anniroot** on

Friday, 24 February 2012 in **RNO-1** at **12h00 (Lunch Break)**

for a detailed explanation of the **Alice** workshop. A **demo** of the **Alice** visual programming environment will also be given. **See you there!**

The Workshop

ATTENTION : DS2 Learners

The 55 DS2 learners listed below are **FINALISED** attendees to the *Alice* workshop. Each learner has indicated their willingness to participate in the workshop.

The workshop will run from the **19 to 30 March 2012**, details follow:

<u>Date</u>	<u>Venue</u>	<u>Time</u>
19 March 2012 (Monday)	Lab 2	12:00 - 13:00 (1 hour)
20 March 2012 (Tuesday)	Lab 2	12:00 - 13:00 (1 hour)
23 March 2012 (Friday)	Lab 2	12:00 - 14:00 (2 hour)
24 March 2012 (Saturday)	Lab 2 (Additional class)	09:00 - 12:00 (3 hour)
26 March 2012 (Monday)	Lab 2	12:00 - 13:00 (1 hour)
27 March 2012 (Tuesday)	Lab 2	12:00 - 13:00 (1 hour)
28 March 2012 (Wednesday)	Lab 2	12:00 - 13:00 (1 hour)
30 March 2012 (Friday)	Lab 2	12:00 - 14:00 (2 hour)

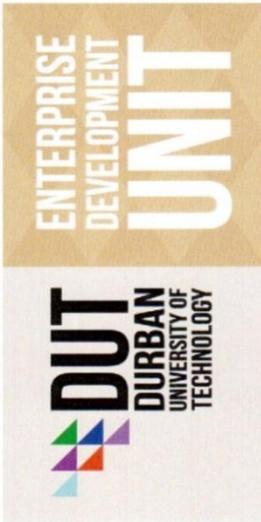
Please download the software, Alice 2.2, which is freely available at:
<http://www.alice.org/>

It is recommended that you practice using the software and go through the tutorials (available within the software) prior to the *Alice* workshop.

Any learners who have **any queries** must please email Miss J. Anniroot at annirootj@dut.ac.za. If a learner **CANNOT ATTEND**, please email Miss J. Anniroot by the 13 March 2012, so that your place can be offered to another learner.

See you there!

C.7 Example certificate of attendance



This is to certify that

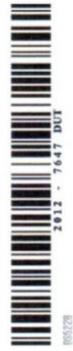
JOHN DOE

Attended a 8 day Workshop - March 2012

The Alice Workshop

19 - 30 March 2012

.....
Mr Surendra Thakur
Director: Enterprise Development Unit



.....
Trainer/Facilitator
Miss J Anniroot

C.8 Example letter of achievement



17 April 2012

Re: Prize award for completion of Alice project: John Doe (Student Number: 20912345)

To whom it may concern

I had conducted a research study that investigated the effectiveness of a 3D visual programming environment called *Alice* to help improve learner performance and to help learners better understand object-oriented programming.

This was achieved through a 7 hour workshop, which was conducted by means of practical interaction with the *Alice* software. Learners were taught how to design simple video games and animated movies. The study was conducted with a selection of learners registered for the subject Development Software 2 within the National Diploma: Information Technology.

I am pleased to confirm that John Doe (Student number: 20912345) had received a 2nd prize award for having completed an outstanding project using *Alice*.

Any queries or questions may be directed to me either via email or telephonically.

Yours Sincerely,

A handwritten signature in black ink, appearing to read "Anniroot", written over a horizontal line.

Miss Jeraline Anniroot

Lecturer

Department of Information Technology

Durban University of Technology

Email: annirootj@dut.ac.za

Office: +27313735583



Appendix D – Quantitative Analysis Tables

D.1 Data collection questionnaire instrument with results: Case Study 1



Case Study 1: Questionnaire

Results of Case Study 1 extracted from raw data in questionnaire

An evaluation of the *Alice* visual programming environment to be undertaken
By the Development Software 2 learners at the Durban University of Technology

Note: All the information you provide in this questionnaire is confidential and will only be used for research purposes.

Please mark your responses with an X in the appropriate blocks.

General Details							
Student Number:				Gender:	Male		Female
Surname:							
First Name(s):							
Race:	Black		White		Asian		Coloured
First year subjects passed at first attempt:	DS1			TP1		IS1	SS1
Registered for these second year subjects:	DS2			TP2		IS2	SS2

Please mark your response with an X. Options are from left to right SA=STRONGLY AGREE, A =AGREE, N =NEUTRAL, D =DISAGREE, SD =STRONGLY DISAGREE

General interface design heuristics, based on Nielsen (1994) and (Dix et al., 2004:325) used to assess the usability of the <i>Alice</i> visual programming environment (n=21)							
Criteria		Frequency of Ratings					
1	<i>Visibility of the system status</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	1.1 I am always aware of what is going on in the system.	7	10	4	0	0	4.14
	1.2 Whenever <i>Alice</i> is opened, the system indicates that it is in the process of loading the software.	9	8	1	3	0	4.10

	1.3	When I save a world in <i>Alice</i> , the system indicates that files are being saved.	12	9	0	0	0	4.57
2	<i>Match between the system and the real world</i>		SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	2.1	The system uses words, terms and phrases that I can easily understand.	10	8	3	0	0	4.33
	2.2	The templates used for new worlds and the objects in the system gallery, relate to real-world objects that I encounter in my day-to-day experiences.	8	10	2	1	0	4.19
3	<i>User control and freedom</i>		SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	3.1	I am comfortable with the level of control that I have over the system.	4	10	6	0	0	3.90
	3.2	<i>Alice</i> allows me the flexibility to use the environment to perform a task.	4	13	4	0	0	4.00
	3.3	I can recover from mistakes quickly and easily.	4	12	4	0	0	4.00
4	<i>Consistency and standards</i>		SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	4.1	The <i>Alice</i> interface maintains a consistent look and feel.	7	13	1	0	0	4.29
	4.2	The startup dialog box, play button, main menu and tab controls are clearly and consistently displayed.	12	8	1	0	0	4.52
	4.3	The use of the mouse controls buttons and camera navigation controls are always available to move and arrange objects.	8	9	3	0	0	4.25
5	<i>Error prevention</i>		SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	5.1	The <i>Alice</i> software allows me to focus on the objects and their associated properties and methods, rather than dealing with complex syntax.	7	12	2	0	0	4.24
	5.2	The <i>Alice</i> interface design does not cause me to make errors.	4	11	4	1	0	3.90
6	<i>Recognition rather than recall</i>		SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	6.1	The actions to be taken and options available for selection are clear and visible at all times.	3	14	3	1	0	3.90
	6.2	I do not have to remember the information from a previous screen in order to proceed with the next one.	2	5	8	4	2	3.05
7	<i>Flexibility and efficiency of use</i>		SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	7.1	<i>Alice</i> caters for novice to expert users.	4	12	5	0	0	3.95
	7.2	I often use shortcut keys and/or combination of keyboard and mouse use to control the movement of objects.	2	4	11	3	1	3.14
	7.3	I often use both the single view and the quad view to move, align and reposition objects on the screen.	5	4	6	5	1	3.33

8	<i>Aesthetic and minimalist design</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	8.1 There is no irrelevant information in the <i>Alice</i> interface design that distracts me and slows me down.	2	12	5	2	0	3.67
9	<i>Recognition, diagnosis and recovery from errors</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	9.1 <i>Alice</i> does not crash while I'm using it.	6	9	3	3	0	3.86
	9.2 In cases where I encounter system errors, the system provides an appropriate error message in simple language.	3	11	5	1	0	3.80
10	<i>Help and documentation</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	10.1 The four tutorials in the startup dialog box are useful in helping me to learn how to use <i>Alice</i> .	12	7	2	0	0	4.48
	10.2 The example worlds in the startup dialog box are useful.	12	8	1	0	0	4.52
The teaching and learning of programming							
11	<i>Learner attrition</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	11.1 I had a high level of comfort with computer usage prior to studying programming.	7	11	0	3	0	4.05
	11.2 I am able to cope with the breadth and depth of the course work covered in the programming subjects of this course.	4	12	5	0	0	3.95
	11.3 The current teaching methods and techniques used in programming subjects helped me to program successfully.	2	13	6	0	0	3.81
	11.4 I had a high level of comfort with object-oriented programming prior to the <i>Alice</i> workshop.	2	9	7	1	2	3.38
	11.5 I like to learn object-oriented programming in a more fun and interactive way.	14	6	1	0	0	4.62
Challenges faced by learners in learning object-oriented programming							
12	<i>Lack of motivation for programming</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	12.1 I am motivated to learn programming.	11	8	1	1	0	4.38
	12.2 I spend a lot of time intensively practicing programming exercises.	2	12	4	2	1	3.57
13	<i>Fragile mechanics of program creation, particularly syntax</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	13.1 Learning the syntax and semantics of a programming language is challenging.	3	10	6	2	0	3.67
	13.2 It would be easier for me to learn how to solve a problem and learn basic concepts of object-orientation without having to deal with brackets, commas and semicolons.	4	7	8	2	0	3.62
	13.3 I have felt intimidated by direct exposure to programming syntax, such as do-while, if-then-else, switch-case, and for-next.	0	2	7	9	2	2.45

	13.4	The textual nature of the conventional programming environments I use, makes it difficult for me to learn how to program.	0	1	10	9	1	2.52
14	<i>Identifying results of computation as the program runs</i>		SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	14.1	I am able to identify errors and correct them using the feedback given by the program.	4	10	5	2	0	3.76
	14.2	I am able to work independently on a program, from coding to testing.	3	12	5	1	0	3.81
	14.3	My experience from running and debugging programs allows me to solve similar problems.	4	12	5	0	0	3.95
15	<i>Difficulty of understanding compound logic</i>		SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	15.1	I am able to apply basic problem-solving techniques to create algorithms that solve problems.	4	12	5	0	0	3.95
	15.2	I have a good understanding of pseudocode.	4	13	4	0	0	4.00
	15.3	I use pseudocode to outline and understand the logic of a program before I start coding.	1	5	8	6	1	2.95
	15.4	I am able to break down a large and complex programming task into smaller subtasks.	2	14	2	3	0	3.71
How to improve the teaching of object-oriented programming								
16	<i>Algorithmic thinking and expression</i>		SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	16.1	I am able to read and write in a formal language.	7	13	1	0	0	4.29
17	<i>Abstraction</i>		SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	17.1	I am able to use creative thinking and programming concepts to write programs.	6	10	5	0	0	4.05
	17.2	I often write a program with an understanding of every line of code.	2	8	4	6	1	3.19
18	<i>Objects-first strategy</i>		SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	18.1	I have a good level of understanding of objects, gained from my first year of study.	2	12	3	4	0	3.57
	18.2	It would be easier to learn object-oriented programming during the first year of study, and later learn other control structures such as loops, If statements etc.	5	5	4	7	0	3.38
	18.3	<i>Alice</i> helps me to see everything as an object.	8	10	3	0	0	4.24
19	<i>3D animation authoring tools and visualisation</i>		SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	19.1	A visual representation improves my understanding of programming concepts.	10	9	2	0	0	4.38
	19.2	The visual effects in <i>Alice</i> provide a meaningful context for understanding classes, objects, methods, and events.	5	12	4	0	0	4.05
	19.3	Three-dimensionality makes objects seem real.	9	11	1	0	0	4.38
	19.4	I am able to use <i>Alice</i> to write a new method to make objects perform animated tasks, such as hop, fly, swim etc.	6	14	1	0	0	4.24

Criteria from the Researcher's experience							
20	<i>Appreciation of trial and error</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	20.1 When using <i>Alice</i> , I use trial and error to 'try out' individual animation instructions as I create new methods.	4	8	8	1	0	3.71
	20.2 I can visibly see the effect that each new animation instruction has on the animation.	3	10	4	4	0	3.57
21	<i>Incremental construction approach</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	21.1 <i>Alice</i> has taught me how to program incrementally i.e. I write one method at a time, testing and running each piece.	5	11	4	0	0	4.05
22	<i>Alice has improved my understanding of these OOP concepts:</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	22.1 Inheritance	4	11	5	0	0	3.95
	22.2 Methods	5	10	5	0	0	4.00
	22.3 Properties	5	12	3	0	0	4.10
	22.4 Functions	5	7	7	1	0	3.80
23	<i>Alice has improved my understanding of these basic programming concepts:</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	23.1 Loops	9	5	6	1	0	4.05
	23.2 If..statements	5	7	7	1	1	3.67
	23.3 Data types	5	6	9	1	0	3.71
	23.4 Event-driven programming	4	6	9	2	0	3.57
24	<i>Ability to collaborate</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	24.1 The <i>Alice</i> workshop has provided the opportunity to work in pairs with other learners and I have chosen to do so.	7	8	4	1	1	3.90
	24.2 I have been able to interact and communicate well with other learners.	5	11	5	0	0	4.00
	24.3 The experience of working with other learners has helped me to learn the <i>Alice</i> programming environment.	4	11	6	0	0	3.90
25	<i>Impact of Alice on DS2 learners</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	25.1 The <i>Alice</i> workshop relates directly to the sections on object-oriented programming covered in the Development Software 2 syllabus.	6	8	6	1	0	3.90
	25.2 I am interested in learning more about computer graphics and animation.	15	5	0	0	0	4.75
	25.3 I am interested in learning and working more with the <i>Alice</i> visual programming environment.	10	9	2	0	0	4.38
	25.4 I used <i>Alice</i> during my personal time after attending the first lesson of the <i>Alice</i> workshop.	6	7	3	5	0	3.67
	25.5 I am interested in learning more about object-oriented programming.	13	7	1	0	0	4.57

D.2 Data collection questionnaire instrument with results: Case Study 2



Case Study 2: Questionnaire

Results of Case Study 2 extracted from raw data in questionnaire

**An evaluation of the *Alice* visual programming environment to be undertaken
By the Development Software 2 learners at the Durban University of Technology**

Note: All the information you provide in this questionnaire is confidential and will only be used for research purposes.

Please mark your responses with an X in the appropriate blocks.

General Details											
Student Number:			Gender:			Male		Female			
Surname:			Class Group:								
First Name(s):											
Email:				Contact Tele:							
Age:		Race:		Black		White		Asian		Coloured	

Please mark your response with an X. Options are from left to right SA=STRONGLY AGREE, A =AGREE, N =NEUTRAL, D =DISAGREE, SD =STRONGLY DISAGREE

General interface design heuristics, based on Nielsen (Dix <i>et al.</i> , 2004:325) used to assess the usability of the <i>Alice</i> visual programming environment (n=55)							
	Criteria	Rating					
1	<i>Visibility of the system status</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	1.1 I am always aware of what is going on in the system.	29	24	1	1	0	4.47
	1.2 Whenever <i>Alice</i> is opened, the system indicates that it is in the process of loading the software.	35	11	7	2	0	4.44
	1.3 When I save a world in <i>Alice</i> , the system indicates that files are being saved.	41	11	2	1	0	4.67

2	<i>Match between the system and the real world</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	2.1 The system uses words, terms and phrases that I can easily understand.	42	12	1	0	0	4.75
	2.2 The templates used for new worlds and the objects in the system gallery, relate to real-world objects that I encounter in my day-to-day experiences.	29	19	5	2	0	4.36
3	<i>User control and freedom</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	3.1 I am comfortable with the level of control that I have over the system.	23	28	4	0	0	4.35
	3.2 <i>Alice</i> allows me the flexibility to use the environment to perform a task.	29	19	6	1	0	4.38
4	<i>Consistency and standards</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	4.1 The <i>Alice</i> interface maintains a consistent look and feel.	18	32	5	0	0	4.24
	4.2 The startup dialog box, play button, main menu and tab controls are clearly and consistently displayed.	34	18	2	1	0	4.55
	4.3 The use of the mouse controls buttons and camera navigation controls are always available to move and arrange objects.	26	26	3	0	0	4.42
5	<i>Error prevention</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	5.1 The <i>Alice</i> software always gives error messages to prevent errors from occurring.	18	19	15	3	0	3.95
	5.2 The <i>Alice</i> interface design does not cause me to make errors.	11	29	13	2	0	3.89
6	<i>Recognition rather than recall</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	6.1 The actions to be taken and options available for selection are clear and visible at all times.	15	30	8	2	0	4.05
	6.2 I do not have to remember the information from a previous screen in order to proceed with the next one.	5	14	17	18	1	3.07
7	<i>Flexibility and efficiency of use</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	7.1 <i>Alice</i> caters for beginner to expert users.	24	22	8	0	1	4.24
	7.2 I often use both the single view and the quad view to move, align and reposition objects on the screen.	18	20	12	5	0	3.93
8	<i>Aesthetic and minimalist design</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	8.1 There is no irrelevant information in the <i>Alice</i> interface design that distracts me and slows me down.	9	27	9	9	1	3.62
9	<i>Recognition, diagnosis and recovery from errors</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	9.1 <i>Alice</i> does not crash while I'm using it.	22	18	11	3	1	4.04
	9.2 In cases where I encounter system errors, the system provides an appropriate error message in simple language.	14	24	12	4	1	3.84
	9.3 I can recover from mistakes quickly and easily.	21	23	7	2	1	4.13

10	<i>Help and documentation</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	10.1 The four tutorials in the startup dialog box are useful in helping me to learn how to use <i>Alice</i> .	21	27	6	1	0	4.24
	10.2 The example worlds in the startup dialog box are useful.	19	29	7	0	0	4.22
The teaching and learning of programming							
11	<i>Learner attrition</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	11.1 I had a high level of comfort with computer usage prior to studying programming.	31	21	3	0	0	4.51
	11.2 I am able to cope with the breadth and depth of the course work covered in the programming subjects of this course.	27	19	8	1	0	4.31
	11.3 The current teaching methods and techniques used in programming subjects helped me to program successfully.	22	25	7	1	0	4.24
	11.4 I had a high level of comfort with object-oriented programming prior to the <i>Alice</i> workshop.	15	25	13	2	0	3.96
	11.5 I like to learn object-oriented programming in a more fun and interactive way.	34	20	0	0	1	4.56
Challenges faced by learners in learning object-oriented programming							
12	<i>Lack of motivation for programming</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	12.1 I am motivated to learn programming.	33	20	1	0	1	4.53
	12.2 I spend a lot of time intensively practicing programming exercises.	12	21	19	3	0	3.76
	12.3 <i>Alice</i> has improved my motivation for programming	18	30	5	1	1	4.15
13	<i>Fragile mechanics of program creation, particularly syntax</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	13.1 Learning the syntax and semantics of a programming language is challenging.	10	16	19	8	2	3.44
	13.2 It would be easier for me to learn how to solve a problem and learn basic concepts of object-orientation without having to deal with brackets, commas and semicolons, as is the case with <i>Alice</i> .	22	16	8	7	2	3.89
	13.3 I have felt intimidated by direct exposure to programming syntax, such as do-while, if-then-else, switch-case, and for-next.	4	10	16	17	8	2.73
	13.4 The textual nature of the conventional programming environments I use, makes it difficult for me to learn how to program.	0	2	10	33	9	2.09
14	<i>Identifying results of computation as the program runs</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	14.1 I am able to identify errors and correct them using the feedback given by the program.	12	30	11	2	0	3.95
	14.2 I am able to work independently on a program, from coding to testing.	18	33	3	0	1	4.22
	14.3 My experience from running and debugging programs allows me to solve similar problems.	15	34	5	1	0	4.15

	14.4 Alice provides immediate feedback as the program runs.	16	28	11	0	0	4.09
15	<i>Difficulty of understanding compound logic</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	15.1 I am able to apply basic problem-solving techniques to create algorithms that solve problems.	10	33	11	1	0	3.95
	15.2 I have a good understanding of pseudocode.	11	27	11	3	3	3.73
	15.3 I use pseudocode to outline and understand the logic of a program before I start coding.	4	15	18	13	5	3.00
	15.4 I am able to break down a large and complex programming task into smaller subtasks.	10	26	13	5	1	3.71
	15.5 Alice allows me to focus on problem-solving.	15	32	7	1	0	4.11
How to improve the teaching of object-oriented programming							
16	<i>Algorithmic thinking and expression</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	16.1 I am able to read and write in a formal language.	23	30	2	0	0	4.38
17	<i>Abstraction</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	17.1 I am able to use creative thinking and programming concepts to write programs.	16	34	5	0	0	4.20
	17.2 I often write a program with an understanding of every line of code.	7	15	12	18	3	3.09
18	<i>Objects-first strategy</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	18.1 I have a good level of understanding of objects, gained from my first year of study.	15	23	13	4	0	3.89
	18.2 It would be easier to learn object-oriented programming during the first year of study, and later learn other control structures such as loops, If statements etc.	21	11	8	10	5	3.60
	18.3 Alice helps me to see everything as an object.	21	27	4	1	0	4.28
19	<i>3D animation authoring tools and visualisation</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	19.1 A visual representation improves my understanding of programming concepts.	23	24	8	0	0	4.27
	19.2 Three-dimensionality makes objects seem real.	25	25	4	1	0	4.35
	19.3 The visual effects in Alice provide a meaningful context for understanding classes, objects, methods, and events.	25	27	2	1	0	4.38
	19.4 I am able to use Alice to write a new method to make objects perform animated tasks, such as hop, fly, swim etc.	37	15	3	0	0	4.62
Criteria from the Researcher's experience							
20	<i>Appreciation of trial and error</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	20.1 When using Alice, I use trial and error to 'try out' individual animation instructions as I create new methods.	9	27	14	4	1	3.71
	20.2 I can visibly see the effect that each new animation instruction has on the animation.	2	28	17	7	1	3.42

21	<i>Incremental construction approach</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	21.1 <i>Alice</i> has taught me how to program incrementally i.e. I write one method at a time, testing and running each piece.	20	27	8	0	0	4.22
22	<i>Alice has improved my understanding of these OOP concepts:</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	22.1 Inheritance	17	23	13	2	0	4.00
	22.2 Methods	23	24	8	0	0	4.27
	22.3 Properties	19	26	10	0	0	4.16
	22.4 Functions	20	28	6	1	0	4.22
23	<i>Alice has improved my understanding of these basic programming concepts:</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	23.1 Loops	23	21	11	0	0	4.22
	23.2 If..statements	21	22	10	2	0	4.13
	23.3 Data types	18	18	15	3	1	3.89
	23.4 Event-driven programming	16	23	14	1	1	3.95
24	<i>Ability to collaborate</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	24.1 The <i>Alice</i> workshop has provided the opportunity to work in pairs with other learners and I have chosen to do so.	15	26	11	3	0	3.96
	24.2 I have been able to interact and communicate well with other learners.	13	33	6	3	0	4.02
	24.3 The experience of working with other learners has helped me to learn the <i>Alice</i> programming environment.	11	29	11	4	0	3.85
25	<i>Impact of Alice on DS2 learners</i>	SA (5)	A (4)	N (3)	D (2)	SD (1)	Avg Rate
	25.1 The <i>Alice</i> workshop relates directly to the sections on object- oriented programming covered in the Development Software 2 syllabus.	19	27	4	5	0	4.09
	25.2 I am interested in learning more about computer graphics and animation.	37	14	3	1	0	4.58
	25.3 I am interested in learning and working more with the <i>Alice</i> visual programming environment.	31	20	4	0	0	4.49
	25.4 I used <i>Alice</i> during my personal time after attending the first lesson of the <i>Alice</i> workshop.	16	23	9	6	1	3.85
	25.5 I am interested in learning more about object-oriented programming.	36	19	0	0	0	4.65

Appendix E – Qualitative Analysis Tables

E.1 Initial Codebook: Framework (adapted from Guest *et al.*, 2012:57)

Critical Domain Area	Q#	Structural Code Name	Structural Code Definition
Usability	1	Usab	<i>Brief Definition:</i> The usability of the <i>Alice</i> visual programming environment.
ChallengesOOP	2	Chal	<i>Brief Definition:</i> The challenges faced by learners in the learning of OOP.
TechniquesOOP	3	Tech	<i>Brief Definition:</i> Useful ways to improve the teaching of OOP.
<i>Alice</i> Impact	4	<i>Alice</i>	<i>Brief Definition:</i> The impact of <i>Alice</i> on the learners' understanding of OOP and addressing the above-mentioned challenges.
TL@DUT	5	TL	<i>Brief Definition:</i> The learners' personal experience with the teaching and learning of programming at DUT.

Final Codebook after ATA: Derived from qualitative data of open-ended questions and interview transcripts

Final Codebook		Frequency Counts		
Code	Description	Study 1 Open-end Qs	Study 2 Open-end Qs	Study 2 Inter Vs
Usab1	Spontaneous positive responses regarding the usability of <i>Alice</i>			
Usab1.0	General overview comments			
Usab1.0.1	I like <i>Alice</i>	21	46	3
Usab1.0.2	<i>Alice</i> is a fun/exciting/enjoyable/interesting way to learn programming	8	17	4
Usab1.0.3	Improved understanding of DS2 course work	-	4	2
Usab1.0.4	Suitable for novice programmers, e.g. high school learners or first-year learners starting OOP	1	3	7
Usab1.0.5	Promotes learner collaboration	1	1	4
Usab1.0.6	Promotes self-learning; anticipated future personal use	2	3	6
Usab1.0.7	It is easier to visualise coding practices in <i>Alice</i> when compared to conventional programs	-	-	2
Usab1.0.8	Using <i>Alice</i> was easy because of previous knowledge gained from first year studies	-	-	1
Usab1.0.9	Prior exposure to <i>Alice</i> gave me a slight advantage	-	-	1
Usab1.1	Visibility of the system status			
Usab1.1.1	Immediate feedback after program execution	1	2	2
Usab1.1.2	<i>Alice</i> is interactive	3	1	-

Usab1.2	Match between the system and the real world			
Usab1.2.1	Graphics and animation	5	13	1
Usab1.2.2	Dealing with real objects brings coding into reality	1	6	-
Usab1.2.3	Visualisation, e.g. see creation of methods, see objects move on command	4	9	3
Usab1.2.4	3-Dimensionality: visual effects reflect real-world scenarios	1	2	-
Usab1.3	User control and freedom			
Usab1.3.1	Learning how to create movies/storytelling	2	2	-
Usab1.3.2	Learning how to develop video games	2	2	2
Usab1.3.3	More control over my programs	1	-	-
Usab1.4	Consistency and standards			
-	-	-	-	-
Usab1.5	Error prevention			
Usab1.5.1	Restricting the user to a space during tutorials prevents the occurrence of errors	-	-	1
Usab1.6	Recognition rather than recall			
Usab1.6.1	Drag-and-drop feature limits typing	4	9	10
Usab1.6.2	Releases learner to focus on problem-solving	2	2	-
Usab1.6.3	No complex syntax, only English-like statements	1	2	2
Usab1.7	Flexibility and efficiency of use			
Usab1.7.1	User-defined methods are created to manipulate objects and can be tested individually	1	4	1
Usab1.7.2	Objects are available from the local gallery or can be downloaded from the Internet	1	-	-
Usab1.8	Aesthetic and minimalist design			
Usab1.8.1	<i>Alice</i> interface is simple and easy to use	2	16	17
Usab1.8.2	Pre-defined methods and variables simplify coding in <i>Alice</i>	-	-	7
Usab1.8.3	Dummy-camera helps to recover from loss of focus in the <i>Alice</i> world	-	-	1
Usab1.9	Recognition, diagnosis and recovery from errors			
Usab1.9.1	Can recover from errors quickly and easily	2	1	4
Usab1.10	Help and documentation			
Usab1.10.1	Built-in tutorials assist with learning the basic techniques of programming; help teach how to use <i>Alice</i> ; and provide an overview of <i>Alice</i>	1	-	3
Usab1.10.2	The built-in demo programs and additional Internet resources assisted in answering my queries	-	-	2

Usab2	Spontaneous negative responses regarding the usability of <i>Alice</i>			
Usab2.0	General overview comments			
Usab2.0.1	I don't like <i>Alice</i>	-	1	-
Usab2.0.2	Time consuming – takes practice getting accustomed to <i>Alice</i> at first exposure	2	5	-
Usab2.0.3	The challenges of <i>Alice</i> are that it requires active thinking, creativity and prior understanding of OOP concepts	1	-	-
Usab2.1	Visibility of the system status			
-	-	-	-	-
Usab2.2	Match between the system and the real world			
Usab2.2.1	Poor quality of graphics	1	1	-
Usab2.3	User control and freedom			
-	-	-	-	-
Usab2.4	Consistency and standards			
Usab2.4.1	Process of saving is complex in relation to other software	-	1	-
Usab2.4.2	<i>Alice</i> takes too long to load	1	2	1
Usab2.4.3	Installation of <i>Alice</i> software program on laptop – slows processor speed rapidly	-	1	-
Usab2.4.4	Transferring a program from a 32-bit OS on one computer to a 64-bit OS on another computer	-	2	1
Usab2.5	Error prevention			
-	-	-	-	-
Usab2.6	Recognition rather than recall			
Usab2.6.1	Difficulty with the logic of setting statements for step-by-step execution	-	1	-
Usab2.6.2	Difficulty with the use of looping statements	-	1	-
Usab2.6.3	Difficulty with the use of do in order and do together statements	-	1	-
Usab2.6.4	Difficulty with the use of Math functions, e.g. avoiding the collision of objects	1	4	1
Usab2.6.5	Insufficient built-in methods to manipulate objects	2	1	-
Usab2.6.6	Difficulty with the use of methods/functions	1	6	-
Usab2.6.7	Difficulty with the use of variables/parameters	-	2	-
Usab2.6.8	Difficulty with the use of inheritance	-	-	1
Usab2.7	Flexibility and efficiency of use			
Usab2.7.1	Time consuming – use of the camera to turn the point of view/zoom to a particular object, e.g. set to a new scene; following an object such as a flying bird	1	12	4
Usab2.7.2	Rotating an object or repositioning an object	-	3	2
Usab2.7.3	Use of quad-view	-	3	-
Usab2.7.4	Copying statements when duplication of code required	-	1	1
Usab2.7.5	Have to consider every object	1	-	-

Usab2.7.6	Time consuming – movement of objects, e.g. moving single limbs of bodies, so as to propel the object across the screen	2	-	2
Usab2.7.7	Time consuming – advanced functions are difficult to find and use	1	-	-
Usab2.8	Aesthetic and minimalist design			
Usab2.8.1	User interface cannot be customised, e.g. restricted world templates and objects; non-adjustable windows	2	1	-
Usab2.8.2	Objects allowed to move out of range from execution window	-	1	-
Usab2.8.3	User interface cluttered	-	2	-
Usab2.8.4	I like coding and <i>Alice</i> moves away from this	1	-	-
Usab2.8.5	Structured interface doesn't provide alternate ways to access a method	1	-	-
Usab2.8.6	Ambiguous method names	1	-	-
Usab2.8.7	The look and feel of the <i>Alice</i> GUI was juvenile	-	-	1
Usab2.9	Recognition, diagnosis and recovery from errors			
Usab2.9.1	<i>Alice</i> stalls/crashes intermittently	2	2	1
Usab2.9.2	Time consuming – execution of lengthy projects that have errors at the end	-	1	1
Usab2.9.3	Sound delay during execution of program	-	1	-
Usab2.9.4	Time consuming - no recovery from errors, e.g. 'Console error' upon deletion of code; re-adding objects that 'disappeared'	1	2	-
Usab2.10	Help and documentation			
Usab2.10.1	Insufficient literature and tutorials available on <i>Alice</i>	1	1	-
Usab2.10.2	Tutorials are time consuming to complete	-	-	1
Usab2.10.3	Tutorials are not as effective in teaching <i>Alice</i> when compared to being taught by the lecturer	-	-	2
Chal1	Challenges of learning OOP			
Chal1.0	Inheritance	1	6	3
Chal1.1	Methods (calling/overriding/abstract/virtual/no default/event-driven)	4	7	3
Chal1.2	Functions (calling)	-	2	-
Chal1.3	Properties (get/set)	-	2	2
Chal1.4	Parameter passing	1	2	1
Chal1.5	Creating classes and sub-classes (abstract)	2	3	4
Chal1.6	Creating objects and instantiating objects	1	11	6
Chal1.7	SQL Server and database connectivity	-	1	1
Chal1.8	Poor understanding of every line of code, resulting in regurgitation	-	3	-
Chal1.9	Difficulty with problem-solving and applying concepts, e.g. when to write code for objects and methods	6	14	3

Chal1.10	Logically connecting theoretical concepts of OOP with practical examples	1	2	1
Chal1.11	Lack of visual representation to aid with understanding logic	1	1	2
Chal1.12	Time consuming – practicing concepts	-	2	-
Chal1.13	Lack of motivation, for reasons such as boredom, intimidation or frustration	1	3	2
Chal1.14	Lack of resources to facilitate learning, such as time, money, computers, open labs	2	1	-
Chal1.15	Having to remember syntax	5	4	2
Chal1.16	Understanding compiler errors and debugging	-	3	1
Chal1.17	Language barrier	-	1	3
Chal1.18	Difficulty with self-learning	-	1	-
Chal1.19	OOP is a new concept that has not been dealt with during first year. The objects-first strategy had therefore not been implemented	-	-	6
Chal1.20	Difficulty understanding OOP the first time	-	-	4
Chal1.21	Adjusting from practical to theory-based testing	-	-	1
Tech1				
Tech1.0	Techniques to improve teaching of OOP			
Tech1.0	Detailed, interactive explanations of programs with practical examples that enforce the understanding of theoretical concepts, instead of just giving solutions and/or notes	6	14	5
Tech1.1	Pace of lecturing could be less rapid, i.e. avoid moving ahead with new concepts before learners have had enough time to grasp other concepts	2	10	2
Tech1.2	Guiding the learner through the program logic with step-by-step instructions	-	4	3
Tech1.3	Using visual, graphical, interactive environments as part of standard teaching, to improve learner interest and motivation to learn OOP	3	15	5
Tech1.4	Introduce <i>Alice</i> as a supplementary teaching tool	2	5	17
Tech1.4.1	Vacation period after Matric	-	-	1
Tech1.4.2	During Development Software 101	-	-	6
Tech1.4.3	During Development Software 102	-	-	8
Tech1.4.4	During second year	-	-	1
Tech1.5	Providing solutions to examples	2	2	-
Tech1.6	More examples during lecture time	2	5	1
Tech1.7	Preference to work independently on a new problem before getting solutions from lecturer	-	1	-
Tech1.8	Providing partially-coded examples would reduce typing time	-	1	-
Tech1.9	Increasing level of complexity from introductory programs to advanced, in preparation for tests that are more complex	-	2	-

Tech1.10	Student support systems and resources for IT learners, such as more open lab time, additional tutorials with examples, references and tutors to assist learners one-on-one	3	3	2
Tech1.11	Offering 'Logic' as a first-year semester 1 subject/module, before being introduced to a programming language to help learners improve their logical thinking	-	1	1
Tech1.12	Introducing OOP earlier in the ND: IT programme	2	1	3
Tech1.13	Providing learners with written exercises prior to tests will help learners to grasp the style and format of the test	-	-	1
Tech1.14	Including more fun gaming applications into the syllabus, apart from business applications such as sales, inventory etc.	-	-	1
Tech1.15	Preference for practical lectures in a lab over the theory lectures in a classroom	-	-	2
Alice1	Impact of Alice on improving understanding of OOP			
Alice1.0	Yes, working with Alice has improved my understanding of OOP	16	35	16
Alice1.0.1	I have an improved understanding of OOP concepts such as methods, functions, events, inheritance, properties, parameters, classes, objects, instantiation and polymorphism	11	19	16
Alice1.0.2	Alice helped me understand OOP visually	6	7	8
Alice1.0.3	This is better than previous teaching of programming, which has been purely textual in nature	-	1	-
Alice1.0.4	I can create methods to manipulate and animate objects to perform actions	4	6	-
Alice1.0.5	Everything in Alice is viewed as an object	1	6	2
Alice1.0.6	I have a clearer understanding that objects have different parts. It is easier to work with parts of an object	1	1	1
Alice1.0.7	Reality of objects in Alice helps to understand objects in conventional languages	1	1	3
Alice1.0.8	This environment increased my motivation and confidence to practice examples in OOP	-	1	3
Alice1.0.9	This environment improved my understanding of basic concepts such as loops and selection statements	2	-	1
Alice1.0.10	I refer to examples in Alice to reinforce the understanding of concepts in conventional languages	-	-	2
Alice1.1	Working with Alice has improved my understanding of OOP only to a certain extent	2	7	1
Alice1.1.1	Cannot relate problem solving in conventional languages with the coding behind Alice. It would be more useful if the coding in Alice was shown	-	2	2
Alice1.1.2	A two week workshop is insufficient time to see all the benefits of Alice	-	2	-
Alice1.1.3	Alice would have helped me understand OOP if introduced earlier in the semester/for a new learner	-	2	1

Alice1.1.4	Alice provided extra means of revision to reinforce concepts on OOP, in a fun way	1	4	-
Alice1.1.5	Learners followed the examples given by the instructor without applying their minds to solving new problems. Thus, a solid grasp of OOP concepts cannot be made	-	-	1
Alice1.1.6	The pre-defined objects in Alice does not assist with creating new objects	-	-	1
Alice1.2	No, working with Alice has not improved my understanding of OOP	2	9	-
Alice1.2.1	OOP is more complex and uses more semantics in conventional languages than can be represented with Alice	-	2	1
Alice1.2.2	I already have a sound understanding of OOP	1	5	-
Alice1.2.3	Alice does not relate to second year work	-	1	-
Alice1.2.4	The theory of OOP still eludes me	1	-	-
Alice2				
Alice2	Impact of Alice in addressing challenges of OOP			
Alice2.0	Yes, I agree that Alice as a VPE can help address some of these challenges	15	48	-
Alice2.0.1	It is easier to learn programming through visualisation/graphics, than having to remember the syntax for coding	3	13	2
Alice2.0.2	Ability to see the visual effects of every statement of code	4	6	2
Alice2.0.3	Use of trial-and-error to alter the desired output	1	1	1
Alice2.0.4	Drag-and-drop feature releases the learner from having to write code and complex syntax	3	3	3
Alice2.0.5	Alice is fun, engaging and cultivates an interest in programming	4	6	7
Alice2.0.6	Immediate feedback when program runs	-	1	-
Alice2.0.7	Don't have to deal with complex error messages	-	1	-
Alice2.0.8	Coding is simple and straight-forward	-	3	4
Alice2.0.9	Easy to quickly solve problems encountered	-	2	-
Alice2.0.10	A visual representation of how objects interact with each other is valuable	2	5	2
Alice2.0.11	Tutorials help with the coding	-	1	-
Alice2.0.12	An interactive environment, that represents real-life situations	-	5	-
Alice2.0.13	Promotes repeated revision of programming concepts	-	1	-
Alice2.0.14	Improves learner motivation, e.g. with boredom	-	2	1
Alice2.0.15	It makes programming concepts easy to learn and understand	6	24	10
Alice2.1	No, I do not agree that Alice helps address these challenges	1	4	-
Alice2.1.1	Preference to learn more advanced concepts such as databases, abstract methods/classes	-	2	-
Alice2.1.2	Drag-and-drop feature does not assist with the hard-coding required in programming languages	1	3	1

TL1	Teaching and learning programming at DUT			
TL1.0	General overview comments			
TL1.0.Lec.1	Teaching and learning depends on how a lecturer explains a concept in class	-	-	5
TL1.0.Lng.1	I am able to learn closer to the test dates	-	-	1
TL1.0.Syl.1	The structured approach in the first year should be replaced with an objects-first approach	-	-	1
TL1.0.Syl.2	Conventional languages are more difficult to understand than programs such as <i>Alice</i>	-	-	1
TL1.0.Res.1	The step-by-step .pdf notes provided during the <i>Alice</i> workshop proved useful for self-study	-	-	2
TL1.1	Spontaneous positive responses regarding the teaching and learning of programming at DUT			
TL1.1.Lec.1	I am satisfied with the lecturer's teaching styles	-	-	12
TL1.1.Lec.1.1	In general	-	-	9
TL1.1.Lec.1.2	Only during first year	-	-	3
TL1.1.Lec.2	The lecturers are helpful when you need assistance	-	-	6
TL1.1.Lec.3	Lecturers ensure that learners do their work and motivate them to study	-	-	2
TL1.1.Lng.1	I spend time practicing programming exercises	-	-	13
TL1.1.Lng.2	I am motivated to learn programming	-	-	13
TL1.1.Lng.3	Programming is like Maths. The more you practice, the better you know it	-	-	2
TL1.1.Syl.1	The syllabus is well structured and adequate, so learners should be able to cope if they pace themselves	-	-	3
TL1.1.Syl.2	The syllabus is relevant and current to industry requirements	-	-	1
TL1.1.Syl.3	The ND: IT course offers a high practical component, which helps me to cope	-	-	1
TL1.1.Res.1	DUT provides learners with extensive resources and information for self-study	-	-	1
TL1.1.Res.2	The tutors were able to assist me with problems	-	-	1
TL1.1.Res.3	I have my own computer facilities	-	-	8
TL1.1.Res.4	I am able to get assistance from fellow students, who I believe are good learners	-	-	5
TL1.1.Res.5	I had prior exposure to C++ Java in school, and this helped me to adapt to C# programming language at DUT	-	-	2
TL1.1.Res.6	I had prior exposure to a computer in school; and this helped me to develop basic computer skills	-	-	1
TL1.1.Res.7	The computer lab facilities are adequate for teaching	-	-	1
TL1.1.Out.1	I had prior experience with programming in school, but have an improved understanding of IT since studying at DUT	-	-	1
TL1.1.Out.2	I have an improved understanding of concepts from first year to second year	-	-	3

TL1.2	Spontaneous negative responses regarding the teaching and learning of programming at DUT			
TL1.2.Lec.1	Lecturers do not provide a clear explanation of concepts, leaving the learner to do independent research or seek assistance from third year learners	-	-	6
TL1.2.Lec.2	Lecturers want to move ahead with the syllabus before learners have an adequate grasp of the concepts	-	-	1
TL1.2.Lec.3	Lecture time is insufficient to grasp the work the lecturer is doing	-	-	1
TL1.2.Lec.4	Some lecturers come to class with coded programs, which makes it difficult to follow if you get distracted	-	-	2
TL1.2.Lec.5	I prefer to go through the program step by step with the lecturer, instead of being lectured to	-	-	2
TL1.2.Lec.6	Due to a poor understanding in class I spend more time trying to grasp a concept than time on practicing programming	-	-	1
TL1.2.Lec.7	Some learners are intimidated to ask questions in class	-	-	2
TL1.2.Lec.8	I depend on the lecturer for a clear explanation of concepts. Thereafter, I am able to self-learn with notes and example programs	-	-	4
TL1.2.Lec.9	There was a lack of interaction and consultation with the lecturer, which left learners uncertain about the correctness of their solutions in tests and exams	-	-	1
TL1.2.Lec.10	The DS2 lecturer expected us to know the DS1 syllabus well. New concepts of the syllabus were introduced quickly and students were expected to keep up	-	-	2
TL1.2.Lec.11	Preference for a higher practical component during tests and exams	-	-	1
TL1.2.Lng.1	I feel demotivated when I cannot solve a problem and there is no immediate feedback	-	-	2
TL1.2.Lng.2	I feel demotivated when I cannot understand a concept	-	-	1
TL1.2.Lng.3	I focused on syntax and semantics with no understanding of the underlying concepts	-	-	1
TL1.2.Lng.4	I had no exposure to computer programming at school and wanted to deregister during first year. I am still finding it difficult and confusing	-	-	1
TL1.2.Lng.5	I felt scared and intimidated when I started studying programming, but these fears were dispelled with time	-	-	4
TL1.2.Lng.6	I had no exposure to a computer prior to studying at DUT, and found that developing basic computer skills was challenging	-	-	1
TL1.2.Lng.7	It was difficult to learn debugging and hard to write the syntax for a calculation	-	-	1
TL1.2.Res.1	There are not enough computers in the open labs	-	-	2
TL1.2.Res.2	Labs are often closed before the scheduled time	-	-	1

Appendix F – DVD

F.1 DVD containing screen shots of the Viscovery SOMine cluster maps and component pictures for Case Study 1 and Case Study 2

Appendix G – (Final) Conference paper at IADIS IS Conference in Berlin, March 2012

G.1 Conference paper presented by Anniroot and de Villiers (2012) at the IADIS Information Systems 2012 Conference in Berlin (March) and published in the proceedings. ('Anniroot' is the maiden name of the researcher, now 'Dwarika')

A STUDY OF ALICE: A VISUAL ENVIRONMENT FOR TEACHING OBJECT-ORIENTED PROGRAMMING

Jeraline Anniroot

Faculty of Accounting and Informatics DUT

annirootj@dut.ac.za

M.R. (Ruth) de Villiers

School of Computing UNISA

dvillmr@unisa.ac.za

ABSTRACT

University students learning object-oriented programming (OOP) encounter many complexities. This paper describes a study in which the primary researcher undertook empirical research aimed at analysing learners' interactions with the visual environment, *Alice* with rapid prototyping functionality. A questionnaire survey investigated the learners' experience with the *Alice* environment and their understanding of OOP. Findings indicate that learners lack problem-solving abilities; are unable to grasp programming concepts on an abstract level; and spend insufficient time practicing programming exercises. *Alice* has proven to be an effective tool in helping to address some of these challenges and in improving learners' grasp of OOP. Furthermore, the learners' subsequent programming processes and performance were investigated. Results revealed that there was no statistically significant improvement in the performance of the learners who had been taught *Alice* in comparison to similar learners who were not exposed to the *Alice* intervention.

KEYWORDS

Alice, Object-oriented programming, Visual programming environments, Abstraction, Problem-solving, Motivation

1. INTRODUCTION

This study investigates the teaching and learning of computer programming, with the purpose of improving problem-solving skills and academic performance amongst second-year students of object-oriented programming (OOP) at the Durban University of Technology (DUT) in Kwa-Zulu Natal, South Africa. Novice programmers face various challenges and difficulties in learning OOP, in particular: demotivation; the complex syntax and semantics of an OOP language; the need for immediate feedback; difficulties in understanding compound logic and the application of algorithmic problem-solving skills.

The primary researcher is an Information Technology (IT) lecturer at the DUT, while the co-author is the supervisor of the formers' postgraduate studies. The primary researcher investigated the use of a 3D visual environment called *Alice*, with the aim of improving the understanding of fundamental programming concepts and imparting OOP skills. *Alice* is an open source teaching tool, designed to provide first-time exposure to learners on the basics of OOP. It allows learners to learn fundamental programming concepts whilst creating 3D animated movies and basic video games, thus providing an engaging interactive environment (*Alice*, 2010). The *Alice* system is a breakthrough in teaching OOP, one of its major strengths being that it can make abstract concepts concrete in the eyes of first-time programmers (Dann *et al.*, 2009).

The study commenced during the first term of 2011 with cohorts of learners registered for the subject Development

Software 2 (DS2) in the Department of Information Technology. A supplementary *Alice* workshop was held during lunch hours over a three-week period, where learners experienced hands-on interaction with the *Alice* software installed in the labs. The benefit of this participation was that learners could closely associate the *Alice* software with the corresponding course work in the DS2 syllabus. The sections on OOP were taught using Microsoft C#.Net and, for the experimental group, were supplemented with knowledge gained from also learning OOP in the *Alice* environment.

2. LITERATURE REVIEW

This section considers what other researchers have observed in relation to the teaching and learning of OOP. In addition, a brief overview is given of the *Alice* visual programming environment.

2.1 Teaching and learning object-oriented programming

Learning to program is considered hard for novices. Hence it is important to understand what makes learning how to program so difficult and how students learn (Matthews *et al.*, 2009). Computer programming involves cognitive skills more than the acquisition of a body of knowledge (Hadjerrouit, 2008). It is a complex process, including understanding the task on hand, choosing appropriate methods, coding, testing and debugging an emerging program (Brooks, 1999).

Learners without prior programming experience are likely to be overwhelmed by the breadth and depth of material, thus contributing to attrition (Moskal, 2004). Developing good programming skills typically requires learners to do considerable intensive practice on programming exercises and to gain experience in debugging, which they cannot sustain unless they are adequately motivated (Law *et al.*, 2009).

The teaching and learning of programming can be complex (Jenkins, 2001). Programming has evolved significantly from the traditional imperative (or procedural) programming languages and techniques to languages and techniques that emphasise object-oriented design and implementation (Phelps *et al.*, 2005). Learning OOP involves writing programs in a language with a high level of complexity (Carlisle, 2009). The main reason why the OOP approach is difficult for most novice learners is because it is more abstract than the procedural approach (Hadjerrouit, 1999).

According to Sajaniemi (2008), students learning OOP experience problems not only in developing the required skills for writing programs, comprehension, and debugging, but also in understanding the basic object-oriented concepts. Object-orientation involves objects, classes, abstraction, encapsulation, inheritance, polymorphism, dynamic binding, and modularity, concepts that are used to understand problem situations, to design object-oriented models, and to choose appropriate means of implementation (Hadjerrouit, 1999).

This study highlights prominent challenges faced by learners of OOP, including the following:

- (e) Lack of motivation for programming (Esteves *et al.*, 2008; García-Mateos and Fernández-Alemán, 2009; Dann *et al.*, 2009);
- (f) Complex syntax and semantics (Winslow, 1996; Gomes and Mendes, 2007; Dann *et al.*, 2009);
- (g) Immediate feedback and identifying the results of computation as the created program runs (Wright and Cockburn, 2000; Galvez *et al.*, 2009; Dann *et al.*, 2009); and
- (h) Difficulties in understanding compound logic and the application of algorithmic problem-solving skills (Gomes and Mendes, 2007; Esteves *et al.*, 2008; Dann *et al.*, 2009).

These challenges are addressed in Section 4.2.1, while Section 4.2.2 suggests possible solutions in the form of characteristics and methods that can support the teaching of OOP.

2.2 *Alice*

This study is founded on *Alice*, which is a rapid prototyping environment for 3D object behaviour, designed to help novice programmers develop interesting 3D animations (Zaccone *et al.*, 2003; Dann *et al.*, 2003; Kelleher and Pausch, 2006). Among the virtues of *Alice* as a learning tool are: concrete visualisation of objects and easy inheritance; ready motivation of interesting problems to solve; release from dealing with complex syntax mechanics; and immediate feedback, whereby logic errors become visually obvious (Johnsgard and McDonald, 2008; Dann *et al.*, 2009).

These characteristics have contributed to the emergence of *Alice* over other visual programming environments that were developed to address challenges in teaching and learning programming, examples being Seymour Papert's classic *Logo* (Folk, 1981), *Karel* the robot (Becker, 2001; Larason, 2005), *Kara* (Kiesmuller, 2009), *BlueJ* (Kölling and Rosenberg, 2001), *Greenfoot* (Montero *et al.*, 2010), *Second Life (SL)* (Esteves, 2008) and *MUPPETS* (Phelps *et al.*, 2005).

3. RESEARCH DESIGN AND METHODOLOGY

3.1 Research questions

The purpose of the investigation was to determine the effectiveness of implementing the *Alice* visual programming environment, with a view to improving the computer programming performance of second-level learners at the DUT. Guided by the questions following, the primary researcher conducted empirical research on the learners' use of *Alice* and their subsequent programming processes and performance.

4. What is the effectiveness, as perceived by learners, of using the *Alice* visual programming environment and objects-first approach in addressing the challenges facing novice programming learners within the object-oriented domain?
5. To what extent do the test and exam results of participating learners relate to those of similar learners who were not exposed to the *Alice* intervention?

3.2 Research design

Every evaluation study has a research methodology and research criteria. Regarding the methodology, the study employed mixed-methods research, which according to Creswell and Clarke (2011), is a research design with philosophical assumptions as well as methods. As a method, it focuses on collecting, analysing and combining both quantitative and qualitative data in a single study or series of studies. This form of design helps to broaden understanding by incorporating both qualitative and quantitative research, or to use one approach to better understand, explain, or build on the results from the other (Creswell, 2009).

During the last week of the *Alice* workshop, learners were asked to complete a questionnaire regarding their experiences with, and opinions of, *Alice*. With regard to the evaluation criteria for this study, they were formulated from literature reviews of previous research. The data was processed by conducting quantitative analysis of the responses to closed-ended questions and qualitative analysis of responses to the open-ended questions. Concurrent collection of both quantitative and qualitative data, with the intention of comparing datasets to determine whether convergence occurs or differences emerge or if there is a combination of convergence and divergence, is known as the concurrent triangulation approach (Creswell, 2009).

Furthermore, a quantitative analysis was performed on the learners' test and exam marks to compare the performances of learners who participated in the *Alice* workshop and similar learners who had been taught by conventional means.

3.3 Participants

The participants were second-year learners registered for DS2 at the DUT. The sampling strategy employed was non-probability sampling (Steyn *et al.*, 1994), whereby there is a specific choice in whom or what is selected. The learners chosen to participate were selected from those who had passed all four first-year subjects at first attempt, which contributed to similarity between participants and ensured that they all had an average-to-good understanding of basic programming skills.

The experimental group initially comprised 50 learners. As the study progressed, attrition occurred in the lunch hour *Alice* workshop, and 21 respondents remained in the experimental group. Therefore, in the selection of learners for the comparison group, 21 with a similar success rate at first-year level were selected from the remaining learners, who had been taught OOP by conventional teaching methods only. Furthermore, the process implemented stratified sampling where each stratum was homogeneous with respect to certain characteristics (Steyn *et al.*, 1994). For example, there was identical gender composition in each group.

3.4 Data collection and analysis

The questionnaire contained 25 closed-ended items, based on a 5-point Likert scale from 1 (Strongly Disagree) to 5 (Strongly Agree). Questions 1 to 10 investigated usability of the *Alice* visual programming environment and were therefore based on the interface design heuristics of Jakob Nielsen's ten general principles (or criteria) for evaluation (Dix *et al.*, 2004). An extensive usability study was undertaken but due to word-count constraints, it is excluded from this paper. Questions 11 to 19 emerged from the literature sources and findings of previous studies on the teaching and learning of programming, the challenges faced by OOP learners, and ways of improving the teaching of OOP. Questions 20 to 25 were based on criteria identified by the researcher that emerged from her personal 10-year involvement in teaching OOP to IT learners. These, too, are excluded from this paper due to space constraints.

The open-ended section of the questionnaire, Question 26 with six subquestions, elicited qualitative feedback regarding the learners' experiences with the *Alice* environment, their consequent understanding of OOP, and

improvements they would like to see in the teaching of OOP. According to Oates (2006), after qualitative data has been generated, the data should be analysed in a quest for relationships and themes. This concept of researchers analysing and categorising their data on their own terms is a form of grounded theory, whereby themes and patterns emerge inductively from the data. Based on the qualitative data, the researchers identified prominent ideas and themes from responses to Questions 26.1 to 26.6.

For the quantitative components, data analysis was performed using SPSS (Statistical Package for the Social Sciences) Version 19.0.

4. FINDINGS

This section presents the quantitative analysis of test marks and exam results, comparing performances of the experimental group and the comparison group. The section discusses responses to closed, quantitative questions from the questionnaire. The section also discusses responses to open, qualitative questions from the questionnaire. The questionnaire was completed by the experimental group only.

4.1 Quantitative analysis of test and exam results

The independent t-test was used to compare the mean values between the summative assessment performances of the two groups in their summative assessments. The results showed no significant differences between the mean values of the two groups. Although the results are not significantly different, mean scores in the experimental group were, in general, higher than the comparison group. Table 1 indicates that examination performance and the final mark were approximately 5% higher in the experimental group.

According to Clarke (1994), teaching methods delivered by many different media or by various mixtures of media, all produce similar learning results. Clarke's claim appears to hold relevance in the present study. Similarly, Owusu et al. (2010) conducted a study with cohorts of learners from two randomly selected schools, exposing one group to computer-assisted instruction (CAI) and the other to conventional teaching methods. Results revealed that the CAI learners did not perform better than the conventional group. The study concluded that the use of CAI is not superior to the conventional approach. However, the learners in the CAI group found their e-learning exposure to be interesting.

Table 2. A comparison of the mean scores for the experimental group and the comparison group

Group	DS1 (First-year results)		DS2 (Second-year results)			
	DS101	DS102	Test1	Test2	Exam Mark	Final Mark
Comparison	63.0476	65.1905	64.7619	59.9524	49.2857	54.8095
Experimental	63.4286	69.2381	62.9524	69.9524	54.4762	59.1905

4.2 Quantitative analysis of closed-ended questions

4.2.1 Challenges faced by learners in learning object-oriented programming

Lack of motivation: While 90.5% of the experimental learners agreed that they are motivated to learn programming, only 66.7% admitted to spending a lot of time intensively practicing programming exercises, and a further 19% were unsure. Law, Lee and Yu (2009) state that in order to develop good programming skills, learners should do a lot of intensive practice on programming exercises to gain experience in debugging. This cannot be sustained unless they are adequately motivated. Further investigation is needed to establish the reasons behind the lack of practice done on the learners' part.

Complex syntax and semantics: Winslow (1996), Kelleher and Pausch (2005), and Carlisle (2009) present a strong debate, affirming that learners frequently spend more time dealing with syntactical complexity and detailed issues of coding than on learning the underlying principles of object-orientation or solving the problem. A good percentage (61.9%) of the experimental participants agreed that learning the syntax and semantics of a programming language is challenging, while 28.6% were unsure. Only 52.4% agreed that it would be easier to learn how to solve a problem and learn basic concepts of object-orientation without having to deal with brackets, commas and semicolons, as is the case in *Alice*, while 38.1% were unsure. Further research conducted by Salim et al. (2010) indicates that rapid exposure to the terminology of programming syntax, such as do-while, if-then-else, switch-case, and for-next, may intimidate learners and result in poor performance, as well as learner attrition. Nevertheless, 55% indicated that they were not intimidated by direct exposure to programming syntax, such as do-while, if-then-else, switch-case, and for-next. Thirty-five percent (35%) were unsure and only 10% agreed with the statement. Forty-eight percent (47.6%) of participants in the experiment felt that the textual nature of the conventional programming environments used, makes it easy for them to learn how to program, while another 47.6% were unsure. Carlisle (2009) suggests that the textual nature of most programming environments works against typical learning styles, but the results of this study do not appear to support that. Although the learners enjoyed *Alice*, many of the experimental group were also comfortable learning OOP by

conventional means.

Identifying results of computation as the program runs: Psychological evidence offered by Galvez, Guzman and Conejo (2009) suggests that feedback provided immediately after an error is the most effective pedagogical action. Sixty-seven percent (66.7%) agreed that they were able to identify errors and correct them using the feedback given by the program. Seventy-one percent (71.4%) felt that they were able to work independently on a program, from coding to testing, reaffirming the findings of García-Mateos and Fernández-Alemán (2009), which posit that current educational tendencies centre on the learners' viewpoint rather than on the educators', with the intention of creating independent, reflective and life-long learners. Due to experience from running and debugging programs, 76.2% of the learners felt that they were equipped to solve similar problems. A gulf of visualisation arises when programmer have difficulty mapping the observed behaviour of the running program against their mental model (Wright and Cockburn, 2000).

Difficulty of understanding compound logic: Research conducted by Gomes and Mendes (2007), Esteves *et al.* (2008) and Yu and Yang (2010) affirms that core problems experienced by many novice learners are the basic lack of problem-solving abilities and the difficulties experienced in using basic concepts, such as control structures, to create algorithms that solve concrete problems. The confidence levels of the experiment participants seemed quite high, in that over 76% claimed they were able to apply basic problem-solving techniques to create algorithms that solve problems; that they have a good understanding of pseudocode; and that they were able to decompose a large and complex programming task into smaller subtasks. Conversely, only 28.6% acknowledged using pseudocode to outline and understand the logic of a program before they started coding. One third ($\frac{1}{3}$) disagree with efforts to use pseudocode. There is a general inconsistency between the responses, which contradicts the learners' claims that they do not experience difficulty in understanding compound logic.

4.2.2 How to improve the teaching of object-oriented programming

To address the challenges identified, this section proposes techniques to help teach OOP.

Algorithmic thinking and expression: Algorithmic thinking and expression involves the ability to read and write in a formal language (Dann *et al.*, 2009). In the present study, 95.2% of the group agreed that they were able to read and write in a formal language. This claim will have to be investigated further.

Abstraction: According to Salim *et al.* (2010), many novice programming learners lack the ability to express their creative thinking in terms of programming abstractions, because they do not grasp programming concepts on an abstract level. This may lead to learners writing code without fully understanding every line. Although 76.2% agreed that they were able to use creative thinking and programming concepts to write programs, a third of the participants in the experimental group, acknowledged that on occasions they do write programs without understanding each piece of program code. With a further 19% of experimental learners being unsure about their ability to understand every line of code when writing a program, it can be safe to assume that, more often than not, learners are unable to grasp programming concepts on an abstract level.

Objects-first strategy: While 66.6% of the experimental learners agreed that they have a good level of understanding of objects, gained from their first year of study, 19% had disagreed. According to the ACM Computing Curricula 2001 (2010), the objects-first strategy commences by immediately introducing the notions of objects and inheritance and then goes on to introduce more traditional control structures. However, only 47.6% of the learners felt confident that it would be easier to learn OOP during the first year of study, and later learn the conventional control structures such as loops, if statements etc., while a third disagreed. Phelps *et al.*, (2005) state further that a way to minimise negative concerns and maximise the positive observations associated with object-oriented and objects-first programming, is to provide learners with a rich, interactive, collaborative virtual environment that supports the programming experience. A high percentage (85.7%) of the experimental learners stated that *Alice* helps them to see everything as an object.

3D animation authoring tools and visualisation: Several authors have stated that learners understand programming concepts better when given a visual representation. Three-dimensional animation assists in providing stronger object visualisation and a flexible, meaningful context for helping learners to perceive object-oriented concepts. Three-dimensionality provides a sense of reality for objects (Dann *et al.*, 2003; Gomes and Mendes, 2007; Sajaniemi, 2008; and Carlisle, 2009). A very high percentage (90.5%) of the experimental learners agreed that a visual representation improves their understanding of programming concepts. Furthermore, 81% agreed that the visual effects in *Alice* provide a meaningful context for understanding classes, objects, methods, and events. Finally, 95.2% of the experimental learners agree that three-dimensionality makes objects seem real and that they were able to use *Alice* to write a new method to make objects perform animated tasks, such as hopping, flying, swimming etc.

4.3 Qualitative analysis of open-ended questions

Participants' personal opinions and general attitudes emerged from the qualitative responses to the open-ended questions. All participants in the experiment expressed a liking for *Alice*. A pattern observed from 10% of experimental learners was their preference of having control over the program, while 10% enjoyed being able to exercise critical thinking skills. Participants appreciated not having to concern themselves with tedious typing. Others (10%) mentioned the enjoyment of interacting with other learners. Nearly 20% of the experimental learners found that *Alice* stimulates their creativity; 3D visual environments are more realistic than conventional programming software packages; and learners were motivated to improve their programming skills with time and practice.

In response to the effects of learning with *Alice*, a theme that emerged consistently (86% of participants) was acknowledgement that their understanding of OOP had improved. Moreover, 10% indicated that *Alice* enhanced their grasp of the basic programming concepts, such as selection and iteration (see Figure 4.3.1).

A pattern emerged in that 67% of the participants felt that *Alice's* visual programming environment helped them address some of the challenges they faced in learning OOP. Conversely, 10% doubted that *Alice* was useful in addressing these challenges. Nearly 62% of the experimental learners indicated that the visual feedback improved their understanding of programming concepts, corresponding with the findings of Gomes and Mendes (2007), Sajaniemi (2008), Carlisle (2009), and Montero *et al.*, (2010), all of whom consider visualisation to be a mechanism that promotes the understanding of programming concepts. Approximately 30% of the learners expressed satisfaction with not having to concern themselves with syntax and semantics (see Figure 4.3.2).

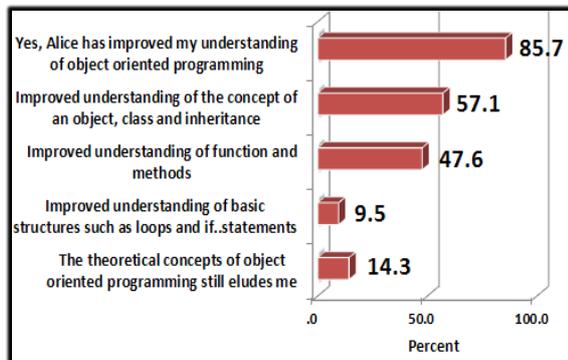


Figure 4.3.1. Bar chart representing learner response to question 26.2

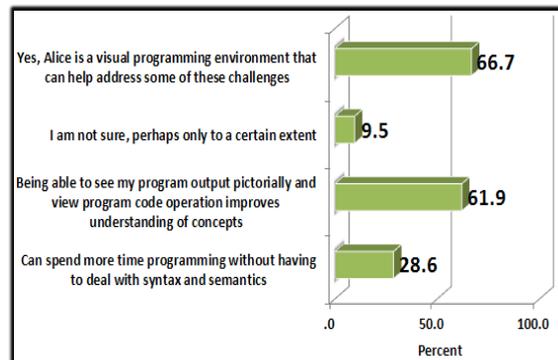


Figure 4.3.2. Bar chart representing learner response to question 26.5

5. CONCLUSION

The mixed-methods approach employed in this research involved quantitative and qualitative studies. This triangulated data collection and analysis, and contributed to confirming the findings. For example, in quantitative analysis of the closed-ended questions, 81% of experimental learners were found to agree that the visual effects in *Alice* provide meaningful contexts for understanding classes, objects, methods, and events. This corresponds with the general pattern observed from experimental learners in response to the qualitative open-ended Question 26.2 (see Figure 4.3.1), where 85.7% of learners believed that *Alice* had improved their understanding of OOP. Similar findings also emerged between the quantitative and qualitative studies, when 66.7% of experimental learners agreed in closed-ended responses that they could identify errors and correct them using the feedback given by their programs and, in the qualitative analysis of Question 26.5 (see Figure 4.3.2), 61.9% of the experimental group stated that the immediate feedback provided by *Alice* improves their understanding of programming concepts.

In response to the research questions in Section 3.1, *Alice* has, firstly, shown itself to be an effective tool that addresses challenges faced by novice programming learners within the object-oriented domain. Secondly, the findings in answer to the question on implementation of *Alice* in the classroom, do not, however, demonstrate a statistically significant improvement in learner performance when using *Alice*. The researchers recommend that a further study should be conducted in the 2012 academic year, with a greater and sustained number of participants and post-questionnaire interviews to strengthen the research.

List of References for IADIS Conference Paper

- ACM Computing Curricula 2001(online)., 2010. Available WWW: www.acm.org/education/curric_vols/cc2001.pdf (Accessed 31 October 2010).
- Alice* (online)., 2010. Available WWW: http://www.Alice.org/index.php?page=what_is_Alice/what_is_Alice (Accessed 08 August 2010).
- Becker, B.W., 2001. Teaching CS1 with Karel the robot in Java. *SIGCSE '01:Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, pp. 50–54.
- Brooks, R., 1999. Towards a theory of the cognitive processes in computer programming. *International Journal of Human Computer Studies*, Vol. 51, pp. 197–211.
- Carlisle, M.C., 2009. Raptor: a visual programming environment for teaching object-oriented programming. *Journal of Computing Sciences in Colleges*, Vol. 24, No. 4, pp. 275–281.

- Clark, R.E., 1994. Media will Never Influence Learning. *Educational Technology Research and Development*, Vol.42, No. 2, pp. 21–29.
- Creswell, J.W., 2009. *Research Design. Qualitative, quantitative and mixed methods approaches*. Third Edition. Sage Publications, United States of America.
- Creswell, J.W. and Clark, V.L., 2011. *Designing and conducting Mixed Methods Research*. Second Edition. Sage Publications, United States of America.
- Dann, W.P. et al, 2003. Teaching Objects-first in introductory Computer Science. *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*. Reno, NV. February, pp. 191–195.
- Dann, W.P. et al, 2009. *Learning to program with Alice*. Second Edition. Pearson Prentice Hall, United States of America.
- Dix, A. et al, 2004. *Human Computer Interaction*. Third Edition. Pearson Prentice Hall, United States of America.
- Esteves, M. et al, 2008. Contextualization of Programming Learning: A Virtual Environment Study. *Frontiers in Education Conference, 2008. FIE 2008.38th Annual*, pp. F2A-17–F2A-22.
- Folk, M., 1981. Review of "Mindstorms: Children, Computers, and Powerful Ideas by Seymour Papert", Basic Books: New York, 1980. *SIGCUE Outlook*. Vol. 15, No. 1, pp. 23–24.
- Galvez, J. et al, 2009. A blended E-learning experience in a course of object-oriented programming fundamentals. *Knowledge-Based Systems* 22, pp. 279–286.
- García-Mateos, G. and Fernández-Alemán, J.L., 2009. A Course on Algorithms and Data Structures Using On-line Judging. *ITICSE09*. July 6-9, pp. 45–49.
- Gomes, A. and Mendes, A.J., 2007. An environment to improve programming education. *Proceedings of the 2007 international conference on Computer systems and technologies*. Vol. 285, No. 88, pp. IV.19-1–IV.19-6.
- Hadjerrouit, S., 1999. A constructivist approach to object-oriented design and programming. *ITiCSE '99 Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education*, Vol. 31, No. 3, pp. 171–174.
- Hadjerrouit, S., 2008. Towards a blended learning model for teaching and learning computer programming: A case study. *Informatics in Education*, Vol. 7, No. 2, pp. 181–210.
- Jenkins, T., 2001. The motivation of students of programming. In *Proceedings of ITiCSE 2001: The 6th annual conference on innovation and technology in computer science Education*, pp. 53–56.
- Johnsgard, K. and McDonald, J., 2008. Using Alice in Overview Courses to Improve Success Rates in Programming I. *Software Engineering Education and Training, 2008. CSEET '08. IEEE21st Conference*, pp.129–136.
- Kelleher, C. and Pausch, R., 2006. Lessons Learned from Designing a Programming System to Support Middle School Girls Creating Animated Stories. *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium*, pp.165 – 172.
- Kiesmuller, U., 2009. Diagnosing Learners' Problem-Solving Strategies Using Learning Environments with Algorithmic Problems in Secondary Education. *Transactions on Computing Education (TOCE)*. Vol. 9, No. 3, pp. 17:1–17:26.
- Kölling, M. and Rosenberg, J., 2001. Guidelines for teaching object orientation with Java. In *Proceedings of the 6th annual conference on Innovation and Technology in Computer Science Education*. Vol. 33, No. 3, pp. 33–36.
- Larason, K., 1995. Using Karel the robot as a classroom motivator. *3C Online*. Vol. 2, No. 4, pp. 6.
- Law, K.M.Y. et al, 2009. Learning motivation in e-learning facilitated computer programming courses. *Computers and Education*. Vol. 55(2010), pp. 218–228.
- Matthews, R. et al, 2009. Multimedia learning object to build cognitive understanding in learning introductory programming. *Proceedings of the 7th International Conference on Advances in Mobile Computing and Multimedia*, pp. 396–400.
- Montero, S. et al, 2010. Dual Instructional Support Materials for introductory object-oriented programming: classes vs. objects. *Education Engineering(EDUCON), 2010 IEEE*, pp. 1929–1934.
- Moskal, B. et al, 2004. Evaluating the effectiveness of a new instructional approach. In *Proceedings of the Special Interest Group on Computer Science Education, Cincinnati, Ohio*, pp. 75–79.
- Owusu, K.A. et al, 2010. Effects of computer-assisted instruction on performance of senior high school biology students in Ghana. *Computers & Education*. Vol. 55, pp. 904–910.
- Phelps, A.M. et al, 2005. MUPPETS: Multi-User Programming Pedagogy for Enhancing Traditional Study: An Environment for both Upper and Lower Division Students. *Frontiers in Education, 2005. FIE '05. Proceedings 35th Annual Conference*, pp. S2H-8–S2H-15.
- Sajaniemi, J. et al, 2008. A study of the development of students' visualizations of program state during an elementary object-oriented programming course. *ACM Journal on Educational Resources in Computing*. Vol. 7, No. 4, pp. 3:1–3:31.
- Salim, A. et al, 2010. On Using 3D Animation for Teaching Computer Programming in Cairo University. *Informatics and Systems (INFOS), 2010 The 7th International Conference*, pp. 1-5.

Steyn, A.G.W. et al, 1994. *Modern Statistics in Practice*, J.L. van Schaik, Pretoria.

Winslow, L.E. 1996. Programming pedagogy - A psychological overview. *ACM SIGCSE Bulletin*. Vol. 28, No. 3, pp. 17–25.

Wright, T. and Cockburn, A. 2000. Writing, Reading, Watching: A Task-Based Analysis and Review of Learners' Programming Environments. *Advanced Learning Technologies, 2000.IWALT 2000.Proceedings. International Workshop*, pp. 167–170.

Yu, P. and Yang, L. 2010. Programming Skills Training in Programming Language Courses. *Educational and Information Technology (ICEIT), 2010 International Conference*. No. 3, pp. V3-14–V3-16.

Zaccone, R. et al, 2003. Using 3D Animation Programming in a Core Engineering Course Seminar. *Frontiers in Education, 2003. FIE 2003. 33rd Annual*. Vol. 2, pp. F4D-14–F4D-17.