

Theses

2012

Designing introductory programming courses : the role of cognitive load

Rainalee Mason
Southern Cross University

Publication details

Mason, R 2012, 'Designing introductory programming courses : the role of cognitive load', PhD thesis, Southern Cross University, Lismore, NSW.
Copyright R Mason 2012

SOUTHERN CROSS UNIVERSITY

Designing Introductory Programming Courses

– The Role of Cognitive Load

by Rainalee Mason
B.Multimedia (Hons) (Software Engineering)

Supervisors:

Dr Graham Cooper
Dr Bruce Armstrong

A THESIS
SUBMITTED TO THE DIVISION OF RESEARCH
IN FULFILMENT OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY

SOUTHERN CROSS BUSINESS SCHOOL

COFFS HARBOUR, NEW SOUTH WALES
NOVEMBER, 2012

Declaration

I certify that the work presented in this thesis is, to the best of my knowledge and belief, original, except as acknowledged in the text, and that the material has not been submitted, either in whole or in part, for a degree at this or any other university.

I acknowledge that I have read and understood the University's rules, requirements, procedures and policy relating to my higher degree research award and to my thesis. I certify that I have complied with the rules, requirements, procedures and policy of the University (as they may be from time to time).

Print Name:.....

Signature:.....

Date:

Abstract

This thesis explores the application of Cognitive Load Theory to the design and delivery of instructional materials for introductory computer programming.

Cognitive Load Theory (CLT) provides an explanation based on human cognitive architecture, and working memory limitations, as to why some content is difficult to learn. CLT also provides guidelines for effective instructional design for novices in complex areas to facilitate schema acquisition, development and automation.

Computer programming is an area that is both complex and difficult.

There are high levels of *element interactivity* between fundamental concepts essential for the design and development of programs (*intrinsic* cognitive load). There is also a need for attention to the aspects of language and development environment that are used (*extraneous* cognitive load). Combined, these reduce the cognitive resources available to be used in the learning process.

As part of an outreach program, introductory programming workshops using two learning environments – Mindstorms NXT and Alice – were designed for high school students using cognitive load principles. These were demonstrated to be effective in facilitating learning of programming skills and in raising interest and self-efficacy in programming within a cohort of female-only students. The students reported mid-range levels of cognitive load while completing the workshop activities, and a perception of programming as moderately difficult.

This appeared to be in conflict with widely reported difficulties in teaching programming at university level. To explore this apparent discrepancy, a survey of

university academics and a survey of novice university programming students were conducted. The results of these studies indicated that some profiles of students were experiencing high to extreme levels of cognitive load.

In order to explore the reasons for the success of the workshops, a controlled experiment was designed, using both males and females. The study was designed to investigate the effect upon learning of the nature of the programming environment interface. One group was given a truncated ‘subset’ version, while the other group was given a more ‘complete’ interface. There were significant differences between groups, favouring the Subset Group, in performance score and performance time. There was also a significant transfer effect, with participants in the Subset Group finding programming within a second environment significantly easier. Implications of these results for the choice of environments for introductory programming are discussed.

Acknowledgements

Thank you to my supervisors – Dr Bruce Armstrong and Dr Graham Cooper – who have ensured that I did actually finish this work, despite my doubts about it ever happening! Graham, you went above and beyond, and I thank you for your patience and good humour with my interruptions and mood swings. Thanks also go to his wife Kerry who handled late night phone calls with good humour and encouragement.

Thank you to my family, especially my children Arianna, Laurey, Bethany and Josiah, who told me I would regret it forever if I didn't follow the dream of completing this study. Thank you to Marty and Josiah, who held down the fort (with mostly good humour at my absent-mindedness, coupled with home-cooked meals and lots of hugs) while I retreated into my cave for the last weeks of this thesis.

Thank you to my wonderful family and friends (too many to mention) who gave their encouragement and support.

Thank you to the many high school girls who participated in the IT Girls programs over the last few years, and to Victory College in Gympie who allowed me to borrow their students for a week to run programming workshops. It's always exciting to watch young faces light up when their program works for the first time.

Dedication

This thesis is dedicated to my grandmother, May Shearim, who taught me I could do anything I wanted to do, if I just put in the right amount of effort, in the right way, at the right time. Nanna, you are sorely missed.

Preface

It has been observed by Sara Hayes (Hayes, 2012 p 11) that “there are 10 types of people in this world: those who understand binary and those that don’t”. This thesis provides useful instructional designs to facilitate and nurture the development of those that do, for the modern computerised technologically advanced world.

List of publications

The following are peer-reviewed publications that were published in the process of this thesis. See Appendix F.1 to F.5 for full copies of the papers.

Mason, R., Cooper, G. & Comber, T., 2011. Girls get IT. *ACM Inroads*, 2(3), pp.71–77.

Mason, R., Cooper, G. & Comber, T., 2011. It's (no longer) a remote chance for girls in IT. In *Proceedings of the 1st International Australasian Conference on Enabling Access to Higher Education 2011*. 1st International Australasian Conference on Enabling Access to Higher Education 2011. Adelaide, Australia: University of South Australia, pp. 310–321.

Mason, R., Cooper, G. & de Raadt, M., 2012. Trends in Introductory Programming Courses in Australian Universities – Languages, Environments and Pedagogy. In M. de Raadt & A. Carbone, eds. *Proceedings of the Fourteenth Australasian Computing Education Conference (ACE2012)*. Melbourne, Australia: Australian Computer Society, Inc., pp. 33–42. Available at: <http://crpit.com/confpapers/CRPITV123Mason.pdf>.

Mason, R. & Cooper, G., 2012. Why the bottom 10% just can't do it - Mental Effort Measures and Implication for Introductory Programming Courses. In *Proceedings of the Fourteenth Australasian Computing Education Conference (ACE2012)*. Fourteenth Australasian Computing Education Conference (ACE2012). Melbourne, Australia: Australian Computer Society, Inc., pp. 187–196. Available at: <http://crpit.com/confpapers/CRPITV123Mason%20.pdf>.

Mason, R. & Cooper, G., (2013). Distractions in Programming Environments. In *Proceedings of the Fifteenth Australasian Computing Education Conference (ACE2013)*. Fifteenth Australasian Computing Education Conference (ACE2013). Adelaide Australia: Australian Computer Society, Inc.

Table of Contents

Declaration.....	iii
Abstract	iv
Acknowledgements.....	vi
Dedication	vii
Preface.....	viii
List of publications	ix
List of Tables	xviii
List of Figures	xxi
CHAPTER ONE: INTRODUCTION.....	1
1.1 Overview of the Research.....	1
1.2 Originality of the Research	5
1.3 Summary of Data Gathering	5
1.4 Background to the Research	6
1.4.1 Information Technology and Skills Shortages	6
1.4.1.1 Importance of Information Technology.....	6
1.4.1.2 IT skills shortages	6
1.4.2 Importance of Programming to Information Technology	7
1.4.3 Declining numbers.....	8
1.4.3.1 Declining number of IT students	8
1.4.3.2 Lower Entrance Scores	9
1.4.3.3 Declining number of programming students	10
1.4.4 Lack of success.....	10
1.4.4.1 Attrition in IT Courses.....	10
1.4.4.2 Lack of progression	11
1.4.4.3 Failure rates in introductory programming	11
1.4.4.4 Lack of skills and understanding	12
1.5 Root causes for Low Student Numbers Completing Programming Courses.....	14
1.6 Organisation of this thesis.....	16
CHAPTER TWO: STUDENT-CENTRED FACTORS	19
2.1 Introduction.....	19
2.2 Programming Aptitude	19
2.3 Computer Literacy/Prior Programming Experience	22
2.4 Student First Language	22
2.5 Age and Cognitive Development.....	24
2.6 Gender.....	25
2.6.1 Low numbers	25
2.6.2 Females and Programming	27
2.6.3 Performance Differences	28
2.6.4 Progression Rates	28
2.6.5 Possible reasons for low numbers	28
2.6.6 Encouraging females	30
2.6.7 Programs to encourage females into programming.....	31
2.7 Student Attitudes.....	33

2.7.1 Motivation	33
2.7.2 Self-efficacy	35
2.7.3 Fear	36
2.8 Conclusion	37
 CHAPTER THREE: COURSE-CENTRED FACTORS.....	39
3.1 Introduction.....	39
3.2 Complexity of programming	40
3.3 Teaching tools.....	45
3.4 Methods	47
3.4.1 Design Patterns.....	47
3.4.2 Roles of Variables	48
3.4.3 Problem Solving	48
3.5 Paradigm Choice.....	50
3.6 Language.....	52
3.6.1 Language and Paradigm	52
3.6.2 Which language?	53
3.6.3 How many languages?.....	54
3.6.4 Mini languages and languages for children.....	55
3.7 Environments	56
3.7.1 No environment?	56
3.7.2 Professional or Teaching?	56
3.7.3 Environments for specific languages.....	57
3.7.3.1 BlueJ	57
3.7.3.2 Greenfoot	59
3.7.3.3 DrScheme.....	62
3.7.3.4 Jeroo.....	63
3.7.4 Learning Environments	64
3.7.4.1 Squeak and Scratch.....	65
3.7.4.2 App Inventor	67
3.7.4.3 Alice	68
3.7.4.4 Mindstorms	71
3.7.5 Which learning environment?	73
 CHAPTER FOUR: INSTRUCTIONAL DESIGN FACTORS.....	75
4.1 Introduction.....	75
4.2 How much instruction?	75
4.2.1 Constructivism.....	75
4.2.2 More instruction - problem solving and direct instruction.....	76
4.3 Cognitive Load Theory	77
4.3.1 Human Cognitive Architecture.....	77
4.3.1.1 Long Term Memory.....	78
4.3.1.2 Working Memory	79
4.3.1.3 Schemas	80
4.3.1.4 Automation	82
4.3.1.5 Thresholds.....	83
4.3.1.6 Expertise	84

4.3.2 Cognitive Load	85
4.3.2.1 Definition	85
4.3.2.2 CLT and Learning.....	85
4.3.2.3 Intrinsic Cognitive Load	86
4.3.2.4 Extraneous Cognitive Load	89
4.3.2.5 Germane Cognitive Load.....	90
4.3.2.6 Cognitive Load and Mental Effort.....	91
4.3.3 CLT Applications	92
4.3.3.1 Segmentation Effect.....	93
4.3.3.2 Goal-free Problems Effect	94
4.3.3.3 Worked Examples Effect	94
4.3.3.4 Faded Worked Examples Effect	96
4.3.3.5 Sub-goals Effect.....	97
4.3.3.6 Split Attention Effect.....	97
4.3.3.7 Modality Effect	99
4.3.3.8 Redundancy Effect.....	101
4.3.3.9 Expertise Reversal Effect.....	102
4.3.4 1Application to complex content.....	104
4.4 Application to programming.....	106
4.5 Studies in novice programming	108
 CHAPTER FIVE: IT FOR SELF-SELECTING GIRLS.....	109
5.1 Introduction.....	109
5.2 Background	109
5.2.1 Motivation	109
5.2.2 The WIT Outreach Program.....	111
5.2.3 School Visits.....	111
5.3 Hypotheses.....	112
5.4 Methodology	113
5.4.1 Careers Visits.....	113
5.4.2 IT Girls Days	114
5.4.2.1 Overview.....	114
5.4.2.2 Questionnaires	114
5.4.2.3 Workshops	116
5.4.3 Instructional Design.....	117
5.4.3.1 Mindstorms Robots Workshop	117
5.4.3.2 Alice workshops	124
5.5 Results and Discussion	128
5.5.1 Age, Computer Literacy and Programming Experience	129
5.5.2 Hypothesis 1 (Gender bias)	129
5.5.3 Hypothesis 2 – Programming Difficulty	132
5.5.3.1 Alice and Mindstorms.....	133
5.5.4 Hypothesis 3 – Self Efficacy	134
5.5.5 Mental Effort	135
5.6 Further Discussion	139
 CHAPTER SIX: IT FOR ALL GIRLS.....	142

6.1 Introduction.....	142
6.2 Background.....	142
6.2.1 Justification for location choice.....	142
6.2.2 Lightning Ridge.....	143
6.2.3 Framework.....	146
6.2.3.1 Information:	146
6.2.3.2 Positive Role Models:	146
6.2.3.3 Hands-on Activities:	147
6.2.4 Other programs	147
6.3 Methodology	149
6.3.1 Logistics	149
6.3.2 Program	151
6.3.3 Alice Workshops	154
6.3.4 Mindstorms Workshops.....	154
6.4 Hypotheses.....	156
6.5 Day 1 Results and Discussion.....	156
6.5.1 Age, Computer Literacy and Programming Experience	156
6.5.2 Hypothesis 1 (Career Aspirations)	157
6.5.3 Hypothesis 2 (Information)	158
6.5.4 Hypothesis 3 (Programming Difficulty).....	159
6.5.4.1 Alice Programming.....	159
6.5.4.2 Mindstorms Programming	160
6.5.5 Hypothesis 4 (Self-Efficacy)	160
6.5.6 Other Benefits.....	161
6.5.6.1 Enjoyment.....	161
6.5.6.2 Interest	161
6.5.6.3 Participant Comments.....	162
6.5.7 Mental Effort Measures	162
6.6 Day 2 Results and Discussion.....	164
6.6.1 Age, Computer Literacy and Programming Experience	164
6.6.2 Hypothesis 1 (Career Aspirations)	165
6.6.3 Hypothesis 2 (Information)	165
6.6.4 Hypothesis 3 (Programming Difficulty).....	166
6.6.4.1 Alice Programming.....	167
6.6.4.2 Mindstorms Programming	167
6.6.5 Hypothesis 4 (Self-Efficacy)	167
6.6.6 Other Benefits.....	168
6.6.6.1 Enjoyment.....	168
6.6.6.2 Interest	168
6.6.6.3 Participant Comments.....	169
6.6.7 Mental Effort Measures	169
6.7 Day 3 Results and Discussion.....	170
6.7.1 Age, Computer Literacy and Programming Experience	170
6.7.2 Hypothesis 1 (Career Aspirations)	171
6.7.3 Hypothesis 2 (Information)	172
6.7.4 Hypothesis 3 (Programming Difficulty).....	172

6.7.4.1 Alice Programming.....	173
6.7.4.2 Mindstorms Programming	173
6.7.5 Hypothesis 4 (Self-Efficacy)	174
6.7.6 Other Benefits.....	175
6.7.6.1 Enjoyment.....	175
6.7.6.2 Interest	175
6.7.6.3 Participant Comments.....	175
6.7.7 Mental Effort	176
6.8 Comparison between IT Girls Days.....	177
6.8.1 Career Aspirations	178
6.8.2 Knowledge of IT Careers	179
6.8.3 Confidence in Programming.....	179
6.8.4 Difficulty of Programming	179
6.8.5 Mental effort measures	180
6.9 Gender bias results (Hypothesis 5)	181
6.10 Further Discussion	182
6.10.1 Success of the program.....	182
6.10.2 Differences with previous program	183
6.10.3 Why did this work?	184
CHAPTER SEVEN: AUSTRALIAN UNIVERSITIES SURVEY.....	187
7.1 Introduction.....	187
7.2 Background	188
7.2.1 Previous Censuses	188
7.2.2 The 2010 Study.....	189
7.3 Methodology	189
7.3.1 Identifying participants.....	189
7.3.2 Terminology	190
7.3.3 2001 and 2003 Census Questions.....	190
7.3.4 Changes in the 2010 Study	191
7.4 Results - Participation rate.....	193
7.5 Results – Student numbers.....	194
7.6 Results – Languages	195
7.6.1 How many languages?.....	195
7.6.2 Primary languages	195
7.6.3 Top languages.....	196
7.6.4 Reasons for language choice	200
7.7 Results – Environments	203
7.7.1 Choice of IDE/tools	203
7.7.2 Reasons for choice of environment	204
7.8 Results - Other teaching aspects	207
7.8.1 Paradigm taught.....	207
7.8.2 On-campus hours.....	209
7.8.3 Instructor Experience.....	210
7.8.4 Texts used.....	210
7.8.5 Problem Solving Strategies	211
7.9 Results - Mental Effort Measures – Instructors and Students	212

7.9.1 Novice programming, mental effort and cognitive load	212
7.9.2 Measures of mental effort.....	213
7.9.3 Participants that did not give a score for all or any of the cognitive load aspects	217
7.10 Results - Mental Effort Measures – the bottom 10%.....	218
7.10.1 Participants' Comment Analysis.....	218
7.10.2 Methodology of further analysis	218
7.10.3 Student Profiles - Ghosts, Idlers and Strivers.....	219
7.10.4 Ghosts - students who are not there.....	220
7.10.5 Idlers – students who do not apply themselves	221
7.10.6 Strivers – Students who are less capable but are trying	223
7.11 Discussion.....	229
 CHAPTER EIGHT: STUDENT SURVEY	232
8.1 Introduction.....	232
8.2 Background.....	232
8.3 Methodology	233
8.3.1 First part (Weeks 3 to 4).....	233
8.3.2 Second part (weeks 12 to 14)	234
8.3.3 Matching.....	234
8.3.4 Hypotheses	235
8.4 Results and Discussion	237
8.4.1 Participants	237
8.4.2 Representative Sample	237
8.4.3 Students who failed/withdrew from the course	238
8.4.4 Hypothesis 1 (Performance versus Mental Effort)	239
8.4.5 Hypothesis 2 (Programming Experience versus Mental Effort)	241
8.4.6 Hypothesis 3 (General Computing Literacy versus Mental Effort)	244
8.4.7 Hypothesis 4 (First Language verses Mental Effort)	247
8.4.8 Hypothesis 5 (Gender versus Mental Effort).....	249
8.4.9 Hypothesis 6 (Age versus Mental Effort).....	252
8.4.10 Hypothesis 7 (Study Mode versus Mental Effort).....	253
8.4.11 Hypothesis 8 (Year of Study versus Mental Effort).....	255
8.4.12 Changes in Mental Effort	255
8.4.13 Student comments	258
8.5 Further Discussion	259
8.5.1 Limitations of study.....	259
8.5.2 Summary.....	261
 CHAPTER NINE: CONTROLLED ENVIRONMENT INTERFACE EXPERIMENT	263
9.1 Introduction.....	263
9.2 School Background.....	264
9.2.1 Introduction to the School	264
9.2.2 Demographics.....	264
9.2.3 Technology	265
9.3 IT Careers Day	265
9.3.1 Overview	265

9.3.2 Timetable for IT Careers Day:	266
9.3.3 Mindstorms Workshops.....	267
9.3.4 Touch and Voice Research.....	269
9.3.5 Careers Sessions	269
9.3.6 Alice Workshops	269
9.4 Methodology	270
9.4.1 Structure	270
9.4.2 Mindstorms Workshop Instructional Design.....	271
9.4.3 Treatment Groups.....	272
9.4.4 Instruments	275
9.4.4.1 Pre-IT Careers Day questionnaire.....	275
9.4.4.2 Post-IT Careers Day questionnaire	276
9.4.4.3 Mindstorms Test	276
9.5 Hypotheses.....	283
9.6 Results and Discussion	285
9.6.1 Age, Computer Literacy and Programming Experience	285
9.6.2 Homogeneity of Groups	285
9.6.3 Hypothesis 1 (Career Aspirations)	285
9.6.4 Hypothesis 2 (IT knowledge)	287
9.6.5 Hypothesis 3 (Programming Difficulty).....	288
9.6.6 Hypothesis 4 (Self-efficacy).....	290
9.6.7 Hypothesis 5 (Test Completion Time)	291
9.6.8 Hypothesis 6 (Test Score)	292
9.6.9 Hypothesis 7 (Interface Schema Acquisition).....	293
9.6.10 Hypothesis 8 (Knowledge Acquisition)	294
9.6.11 Hypothesis 9 (Difficulty of Mindstorms Programming)	295
9.6.12 Hypothesis 10 (Difficulty of Alice Programming).....	297
9.6.13 Hypothesis 11 (Cognitive Load)	298
9.6.14 Hypothesis 12 (Gender Difference - Performance).....	302
9.6.15 Hypothesis 13 (Gender Difference – Attitude)	304
9.7 General Discussion	306
9.8 Conclusion	309
CHAPTER TEN: GENERAL DISCUSSION	311
10.1 The Difficulty of Programming	311
10.2 Cognitive Load Theory.....	311
10.3 Females and Programming	312
10.4 The State of Play in Universities	314
10.5 Environment Simplicity	316
10.6 Limitations and Further Work	317
10.7 A Final Word	319
REFERENCES	320
APPENDIX A: SELF-SELECTING GIRLS.....	343
A.1. Pre-questionnaire – first IT Girls Day	344
A.2. Post-questionnaire – first IT Girls Day	345

A.3. Pre-questionnaire – Second IT Girls Day	347
A.4. Post-questionnaire – Second IT Girls Day.....	348
A.5. Pre-defined Mindstorms script.....	350
A.6. Comments from Participants.....	352
APPENDIX B: IT FOR ALL GIRLS	355
B.1. Pre-workshop questionnaire 1	356
B.2. Pre-workshop questionnaire 2.....	357
B.3. Questionnaire given after Alice/Photoshop session.....	358
B.4. Questionnaire given after Mindstorms session	359
B.5. Post-workshop questionnaire 1	360
B.6. Post-workshop questionnaire 2	361
APPENDIX C: AUSTRALIAN UNIVERSITIES SURVEY	362
C.1. Census document 2001	362
C.2. Census Document 2003.....	366
C.3. Census Questions 2010	372
APPENDIX D: STUDENT SURVEYS	382
D.1. Introductory email to students	382
D.2. Information sheet for survey	383
D.3. First part of survey	384
D.4. Second part of survey.....	386
D.5. Student Comments	387
APPENDIX E: INTERFACE EXPERIMENT	389
E.1. Careers Day Information	389
E.2. Mindstorms Research Information Sheet	390
E.3. Mindstorms Research Consent Form	391
E.4. IT Careers Day Permission Note.....	392
E.5. Mindstorms test – Subset Group	393
E.6. Mindstorms Test – Complete Group	402
E.7. IT Careers Day Evaluation	411
APPENDIX F: PUBLISHED PAPERS.....	412

List of Tables

Table 1-1 ICT enrolments 2006 – 2010.....	9
Table 5-1 Mental Effort during 1st Mindstorms Workshops.....	136
Table 5-2 Mental Effort during 1st Alice Workshops	137
Table 5-3 Mental Effort & Importance Ranking – 2 nd Mindstorms Workshops	138
Table 5-4 Mental Effort & Importance Ranking – 2 nd Alice Workshops.....	138
Table 6-1 Key Demographics	144
Table 6-2 Distance from Lightning Ridge to nearest Universities (details sourced from Google Maps).....	144
Table 6-3 School Characteristics 2009 (Australian Curriculum Assessment and Reporting Authority 2010).....	145
Table 6-4 Measures of mental effort on Alice and Mindstorms - Day 1	163
Table 6-5 Measures of mental effort on Alice and Mindstorms - Day 2	170
Table 6-6 Measures of mental effort on Alice and Mindstorms - Day 3	176
Table 6-7 p-values for differences between Alice & Mindstorms mental effort measures	177
Table 6-8 Medians of 9-Point Likert test questions in pre- and post- questionnaires	178
Table 7-1 University/Course Summary	193
Table 7-2 Number of languages in a course	195
Table 7-3 Primary Languages in 2010.....	196
Table 7-4 Language comparison by course numbers	197
Table 7-5 Language comparison by student numbers	198
Table 7-6 Reasons for language choice	200
Table 7-7 Top 10 reasons for environment choice	205
Table 7-8 Paradigms taught	208
Table 7-9 Hours in class on-campus.....	210

Table 7-10 Instructor experience in years.....	210
Table 7-11 Percentage of class time dedicated to problem solving.....	211
Table 7-12 Mental effort scores.....	214
Table 7-13 Wilcoxon Signed-rank test - comparisons of mental effort\.....	215
Table 7-14 Mental Effort for Strivers	223
Table 8-1 Gender Distribution	238
Table 8-2 Study Mode Distribution	238
Table 8-3 Grade versus Mental Effort	240
Table 8-4 Performance versus Mental Effort Analysis.....	240
Table 8-5 Programming Experience versus Mental Effort	242
Table 8-6 Programming Experience versus Mental Effort Analysis	243
Table 8-7 Programming Experience versus Mental Effort (2 categories)	243
Table 8-8 Programming Experience versus Mental Effort Analysis (2 categories)	243
Table 8-9 Computer Literacy versus Mental Effort.....	245
Table 8-10 Computer Literacy verses Mental Effort Analysis	245
Table 8-11 Literacy versus Mental Effort (2 categories).....	246
Table 8-12 Computer Literacy verses Mental Effort Analysis (2 categories)	246
Table 8-13 Language versus Mental Effort	248
Table 8-14 Language versus Mental Effort Analysis	248
Table 8-15 Gender versus Mental Effort	250
Table 8-16 Gender versus Mental Effort Analysis	250
Table 8-17 Age versus Mental Effort	252
Table 8-18 Age versus Mental Effort Analysis	253
Table 8-19 Study Mode versus Mental Effort	254
Table 8-20 Study Mode versus Mental Effort Analysis	254

Table 8-21 Change in Mental Effort versus Grade	255
Table 8-22 Change in Mental Effort versus Grade Analysis	255
Table 8-23 Mental Effort versus Comments.....	259
Table 8-24 Mental effort versus Comments Analysis	259
Table 8-25 Significant Differences in Groups	261
Table 9-1 Homogeneity between groups	285
Table 9-2 Wilcoxon Signed Rank - Career Intent Day 1	287
Table 9-3 Wilcoxon Signed Rank - IT Knowledge Day.....	288
Table 9-4 Wilcoxon Signed Rank - Difficulty of Programming	289
Table 9-5 Wilcoxon Signed Rank - Self-efficacy.....	291
Table 9-6 Test Score Totals	293
Table 9-7 Test Scores Interface Schema Acquisition	294
Table 9-8 Comparison between Groups for mental effort measures	299
Table 9-9 Very easy / Very hard mental effort proportions.....	302
Table 9-10 Summary of results.....	307

List of Figures

Figure 1-1 Illustration of context for the research	14
Figure 3-1 The BlueJ environment, showing classes and objects (in red).....	58
Figure 3-2 Greenfoot main window, showing classes and inheritance	60
Figure 3-3 The Greenfoot editor window	60
Figure 3-4 Automatically created Greenfoot class skeleton.....	61
Figure 3-5 DrScheme debugger	62
Figure 3-6 Jeroo screenshot showing executing Python program	63
Figure 3-7 Squeak interface	65
Figure 3-8 Scratch building blocks	66
Figure 3-9 The Scratch interface showing a car game.....	66
Figure 3-10 App Inventor Code Blocks.....	67
Figure 3-11 Alice interface	68
Figure 3-12 Mindstorms interface	72
Figure 4-1 Cognitive Architecture - from (Cooper, 1998)	78
Figure 4-2 Java code for an array	81
Figure 4-3 Alice Programming Interface with "split" text explanation	98
Figure 4-4 Alice screenshot with integrated text	99
Figure 4-5 Four components of 4C/ID model (from Van Merriënboer and Kester, 2005)	105
Figure 5-1 Mindstorms brick, sensors and motors.....	118
Figure 5-2 Example of a Mindstorms Humanoid Robot	119
Figure 5-3 Example of Mindstorms interface.....	121
Figure 5-4 Making the robot talk	122
Figure 5-5 Mindstorms Control Panel	122

Figure 5-6 Students with robots they were programming	124
Figure 5-7 Alice interface	125
Figure 5-8 Alice - Object method code blocks	126
Figure 5-9 Alice - code blocks and events.....	127
Figure 7-1 Top 4 languages of each study by number of courses	199
Figure 7-2 Top 4 languages of each study by student numbers.....	199
Figure 7-3 Trends in reasons for language choice	201
Figure 7-4 Trends in environment use	204
Figure 7-5 Trends in paradigms taught.....	209
Figure 8-1 Grade Distribution Comparison	238
Figure 8-2 Previous Programming Experience versus Grade.....	242
Figure 8-3 Computer Literacy versus Grade	244
Figure 8-4 Language versus Performance	247
Figure 8-5 Gender versus Performance	250
Figure 8-6 Age versus Final Grade	252
Figure 8-7 Study Mode versus Grade	254
Figure 9-1 IT Careers Day Schedule	266
Figure 9-2 Mindstorms Workshop Space	268
Figure 9-3 Mindstorms Test Space	268
Figure 9-4 Instructor demonstrating worked example in the Alice workshop	270
Figure 9-5 Experiment structure	271
Figure 9-6 Common tools - "Subset" interface (expanded).....	272
Figure 9-7 Palette tabs	273
Figure 9-8 Complete interface with "common" sub-palette expanded.....	274
Figure 9-9 Complete interface showing sub-palette with loop and wait blocks.....	274

Figure 9-10 Mindstorms Test - Example multiple-choice question	278
Figure 9-11 Dog trick 'program blocks'.....	279
Figure 9-12 Dog trick "Loop" construct	279
Figure 9-13 Dog trick "Wait" construct.....	279
Figure 9-14 Dog trick "timeline"	280
Figure 9-15 Desktop ready for Mindstorms test.....	281
Figure 9-16 Subset blanked interface (with sample answer marked).....	282
Figure 9-17 Complete blanked interface - expanded.....	283
Figure 9-18 Mindstorms and Alice workshop relationship	297
Figure 9-19 Mental Effort Measure distribution - Understanding the problem.....	300
Figure 9-20 Mental Effort Measure distribution - Using the environment.....	300
Figure 9-21 Mental Effort measure distribution - Learning from the task	301
Figure 9-22 Gender versus Group.....	303

Chapter One: Introduction

1.1 Overview of the Research

The research investigates how Cognitive Load Theory (CLT) principles, which may be used to analyse the interplay between working memory and long term memory in the processing of to-be-learnt information (Sweller, 1999) can be applied to the instructional design of introductory programming courses to improve both the accessibility and success of novice students. In this context “accessibility” means that students will consider themselves to be both more capable, and more desiring, of learning computer programming. “Success” means that students gain higher levels of knowledge and skills associated with basic programming concepts as demonstrated by enhanced performance in programming tasks (Bergin and Reilly, 2005).

Cognitive load refers to the mental imposition placed upon working memory (Sweller, 1994) at any given point in time due to the processing of information and execution of any associated procedures. Cognitive Load Theory has often been applied to instructional contexts to assist in analysing the information-processing tasks associated with learning (Mayer and Moreno, 2003).

There are three sources of cognitive load (Paas and Van Merriënboer, 1994; Sweller, 1994). These are

- intrinsic – due to the inherent nature of the to-be-learnt content;
- extraneous – due to the manner by which the to-be-learnt content is presented; and
- germane – due to a person’s conscious focus of mental efforts with the purpose of learning the to-be-learnt content.

The total cognitive load experienced by a learner at any instance in time is the summation of their current intrinsic, extraneous and germane cognitive loads. Importantly, indeed *crucially*, human cognitive resources are limited. Working memory, which is effectively the hub through which all cognitive activity must pass, is strictly bounded in both capacity (Miller, 1956) and duration (Peterson and Peterson, 1959). The three sources of cognitive load effectively compete with one another for the limited resources associated with working memory. Sometimes, in learning contexts, the cognitive load imposed through intrinsic and extraneous sources are so high that there are insufficient resources of working memory remaining available to attribute to the germane aspects of processing and so learning fails.

This thesis argues that in the context of learning introductory programming the intrinsic cognitive load is innately high due to the content being high in ‘element interactivity’ which refers to the way in which many aspects of programming must be considered simultaneously as a pre-requisite for “comprehension”, that is, mental representation of the to-be-learnt content (Sweller, 2010). High element interactivity critically impacts upon cognitive resources because cognitive resources need to simultaneously consider not only the set of individual elements, but also the interrelations between them (Sweller, 1999).

As a consequence of the high intrinsic cognitive load in the area of introductory programming, the role of extraneous cognitive load becomes critically important. By redesigning instructional materials and/or learning activities the extraneous cognitive load may be reduced (for summary of some principles of redesign see Sweller, 2003). This thesis argues that the extraneous cognitive load associated with learning introductory programming can be reduced by selecting

simplified programming environments together with cognitively designed learning activities based around the presentation of worked examples (Sweller and Cooper, 1985). These worked examples are presented in a dual-modality format (Mousavi, Low & Sweller, 1995) comprising visual demonstrations of procedures within programming environments accompanied by verbal instructional explanations of the actions and procedures undertaken.

It is argued that with extraneous cognitive load reduced by the use of a simplified programming environment and a high emphasis of multi-modality worked-examples, freed cognitive resources may be refocused to the germane task of learning the concepts and procedures associated with introductory programming. Furthermore, such refocusing may be guided by the use of programming environments that provide quick visual and/or physical feedback through either computer generated animations and/or the performance of activities by robots that results from the execution of developed computer programs.

Applying cognitive load principles in the instructional design of introductory programming workshops for high school students was expected to lead to students increasing their understanding and assimilation of key programming concepts and constructs, thus reducing their perceptions of the difficulty of programming, and increasing their self-efficacy with respect to learning programming.

Initial studies focussed upon female students who indicated a desire to attend a day of IT workshops. These were delivered by an all-female team of instructors who also discussed the range of career opportunities in IT. These workshops resulted in a decrease in participant's perceptions of the difficulty of computer programming, and an increase in their self-efficacy in computer programming.

In subsequent studies the workshops were presented to a wider pool of participants. Delivery of the workshop to a broad student population again returned decreases in perceptions of the difficulty of computer programming, an increase in participants' self-efficacy in computer programming and additionally, an increase in their aspirations for a career in IT.

The studies described above were compared with the current practices and perceptions within Australian universities through interviews with academic staff teaching introductory programming courses. Student attitudes and performance within a current introductory programming course at an Australian university were also gathered to provide an additional perspective upon perceptions regarding the difficulty of learning computer programming at a university level. These university-based studies provided evidence that many students, particularly those who perform poorly in introductory programming courses, find the task of learning programming to be inaccessible and unsuccessful, at least in part due to excessive cognitive load.

Finally, a controlled experiment was designed to test the effects of one aspect of the workshop intervention - the complexity of the computer programming interface - on students' reported cognitive load, test performance, and attitude towards the difficulty of programming. Results indicated, for both male and female participants, that the complexity of the programming interface is a contributing source of extraneous cognitive load. It is argued that this impacts negatively upon learning of introductory programming concepts and procedures, learning of the specific computer program interface, perceptions of the difficulty of programming, and attitudes towards learning subsequent programming languages and environments.

1.2 Originality of the Research

This research differs from other projects examining programs to improve the accessibility and success of introductory programming through its application of cognitive load principles to the design of programming workshops. While cognitive load principles have been applied to teach introductory concepts in several other complex domains such as mathematics (Cooper and Sweller, 1987; Sweller and Cooper, 1985), and science (Chandler and Sweller, 1991), little has been done in the direct application of these principles to the teaching of introductory computer programming. Movements towards mindfulness and application of cognitive load theory to introductory programming have only recently begun (Lister, 2008; Margulieux, Guzdial & Catrambone, 2012).

1.3 Summary of Data Gathering

Data for this thesis was collected through pre and post workshop surveys associated with intervention workshops delivered at secondary schools. The findings from these surveys are discussed in Chapters 5 and 6.

A survey regarding aspects of introductory programming courses in Australian universities was conducted via phone interviews with instructors of these courses. The results of this survey are discussed in Chapter 7.

Programming students in an introductory programming course within an Australian university were also surveyed in a two-part survey regarding their views on learning computer programming, the results of which are reported in Chapter 8.

A controlled experiment investigating the effects of a computer programming environment interface design was conducted in an Australian secondary school. The impact of a “subset” versus “complete” interface was compared by performance

measures on test questions after completion of a learning phase, along with pre- and post-experiment surveys regarding perceptions of the difficulty of programming, self-efficacy and measures of cognitive load. These are presented in Chapter 9.

1.4 Background to the Research

1.4.1 Information Technology and Skills Shortages

1.4.1.1 Importance of Information Technology

Hawkrige (1983) describes how information technology has penetrated almost all fields of human activity, and transformed economic and social life. The United Nations Development Program notes that Information Technology is not only the result of growth, but can be used to support growth and development (UNDP, 2001).

In Australia, approximately 555000 ICT workers produce \$82 billion in revenue - \$4.5 billion in exports, with continuing growth expected (Australian Computer Society, 2011). With the exception of the very recent resources boom, IT has contributed more to Australia's GDP than the mining sector (Morling and McDonald, 2011).

1.4.1.2 IT skills shortages

"Today, organizations of every kind are dependent on information technology. They need to have appropriate systems in place. These systems must work properly, be secure, and be upgraded, maintained, and replaced as appropriate. Employees throughout an organization require support from IT staff who understand computer systems and their software and are committed to solving whatever computer-related problems they might have. Graduates of Information Technology programs address these needs."

(The Joint Task Force for Computing Curricula, 2005)

It is therefore important that skilled IT workers are produced in sufficient numbers to service the industry. In the last 10 years, industry experts began to warn that the skills shortage in IT would impact Australia's competitiveness in technology (NICTA, 2007). Estimates of this skill shortage vary, but scarcities are impacting growth of ICT within Australia. In 2009, 48% of Victorian ICT companies designated access to skilled employees as an obstacle to their future growth (Dept of Innovation Industry and Regional Development, 2009).

Not all IT workers have tertiary qualifications, however industry projection reports from 2008 conservatively predict a shortfall of 3700 IT graduates in Victoria alone by 2022 (Shah, Burke & North, 2008). Some specific IT skills are in demand, particularly programming skills and skills associated with emerging technologies (Dept of Business and Innovation, 2012). While not all IT graduates will go on to become programmers, job prospects for software and applications programmers are projected to be 'above average' for the next 5 years (DEEWR, 2011).

The future career prospects for those who choose to study IT, and in particular programming, are encouraging. Despite this the number of students enrolling in IT courses in Australia continues to decline (see Section 1.4.3).

1.4.2 Importance of Programming to Information Technology

The Curriculum Guidelines for Undergraduate Degree Programs in Information Technology, published by the ACM and IEEE Computer Society (2008), points out that undertaking an introductory programming course is essential for IT students because foundational programming concepts are used in nearly all other core IT courses. The ACM Education Board produces curricula for computer science and information technology courses, and programming is named as one of their four

“great principles of computing”, along with systems thinking, modelling and innovating (Denning, 2003). Programming is also named as one of the five pillars of knowledge of which a solid background is needed to enable graduates to solve computing and informational problems (The Joint Task Force for Computing Curricula, 2005). The curriculum guidelines suggest that at least one introductory programming course should be included in the first year of an IT degree program, with preferably a two-course programming sequence.

In addition, Liao and Bright (1991) note that learning to program has benefits beyond learning a specific computer language, including general problem-solving skills, evaluation of solutions and top-down thinking.

1.4.3 Declining numbers

1.4.3.1 Declining number of IT students

Despite the optimistic prospects for careers in IT in Australia, the ACS (2010) figures show a clear drop in domestic ICT higher education enrolments of 57% over the period 2001 to 2008, with a small increase in the 2009 to 2010 period. ICT students comprised only 4% of total students at Australian higher education institutions in 2010, declining from 4.8% in 2006 (Australian Computer Society, 2011).

There are several reasons for this decline. Many prospective students believe that opportunities for good careers in computing were lost to cheaper-labour countries after the 2000 dot-com bust and will not return (Foster, 2005). A majority of school students – traditionally the source of the bulk of university enrolments – believe that IT careers involve boring indoor work sitting in front of a screen all day, with

minimal human interaction and variety (Dept of Innovation Industry and Regional Development, 2009; MMV, 2004).

While there has been a substantial decline in Australian IT course enrolments since 2001, the number of IT students in Australian universities have stayed relatively constant between 2006 and 2010, primarily as a result of an influx of international student enrolments (Australian Computer Society, 2011, 2010). At the time of the latest available figures (2010), around 55% of the ICT enrolments at Australian universities were international students, and the number of domestic ICT students had only slightly increased (Table 1-1), but was approximately half the 2001 level of 17436 students.

	Domestic		International		Total
2006	8198	44.8%	10087	55.2%	18285
2007	7839	43.0%	10384	57.0%	18223
2008	7470	38.6%	11896	61.4%	19366
2009	8328	40.5%	12243	59.5%	20571
2010	8704	44.6%	10822	55.4%	19526

Table 1-1 ICT enrolments 2006 – 2010

This lower number of domestic IT students has implications for the supply of skilled Australian IT workers in the longer term.

1.4.3.2 Lower Entrance Scores

As a response to the decline in demand for IT degrees, some universities have lowered entrance scores required for admittance (Ford and Venema, 2010), with tertiary entrance ranks as low as 44.5 (from a maximum 100) being reported for some universities (Head, 2012). The relationship of entry score to success has been well-documented (Schmitt, Keeney, Oswald, Pleskac, Billington, Sinha & Zorzie, 2009),

so this may have a bearing on the lack of success of some students (see Section 1.4.4 below).

1.4.3.3 Declining number of programming students

Research undertaken in 2001 and 2003 (De Raadt et al., 2004) showed that the number of students enrolling in introductory programming courses were declining despite the relatively constant number of IT students (domestic and international). The average number of students per introductory programming course (usually a one semester unit of study) in Australian universities dropped from 349 in 2001 to 229 in 2003. The research conducted for this thesis confirms that this number is still declining (see Chapter 7). That the steady decline is still underway is especially problematic for the IT industry because it exacerbates the difficulty of filling the steadily growing number of IT positions.

1.4.4 Lack of success

1.4.4.1 Attrition in IT Courses

Attrition (“drop-out”) levels in IT degree programs and programming courses are very high. Internal drop-out rates of 35% to 50% have been reported (Denning and McGettrick, 2005), and in one case, a drop to 23% attrition was seen as a successful intervention (Moura, 2009).

The success of programming courses is integral, as Ma, Ferguson, Roper & Wood (2007) suggest that students' poor performance in programming is most likely the main reason computer science courses have such high attrition rates. D'Souza, Hamilton, Harland, Muir, Thevathayan & Walker (2008) note that when students struggle with programming or have doubts about their ability to succeed, the resulting

negativity tends to permeate through all of their studies. Academics teaching programming are under pressure to limit failure rates and hence reduce attrition (Ford and Venema, 2010).

1.4.4.2 Lack of progression

Even those students who do successfully complete the first introductory course in programming often do not progress further. Very little is published on programming students' progression rates (Sheard, Simon, Hamilton & Lönnberg, 2009), however in one particular published study, 50% of students passing the first introductory course did not enrol in a second semester of programming (Bloch, 2000).

1.4.4.3 Failure rates in introductory programming

Failure rates for introductory programming courses are difficult to assess due to the reticence of educators to disclose them. One international survey of tertiary institutions in the US, Australia and Europe had a relatively low response rate of 63 institutions total, with an average reported pass rate of 67% (fail rate of 33%) for introductory computer courses (Bennedsen and Caspersen, 2007), however the researchers noted the possible reluctance of educators with high failure rates to disclose these to the survey. They also noted that anecdotal evidence suggests some courses with failure rates as high as 90%.

A failure rate of 27% may be considered to be low (Bloch, 2000) and often such failure rates do not include those students who withdraw part-way through a course.

1.4.4.4 Lack of skills and understanding

Of those students who do not withdraw from a programming course and do not fail the course, some are still not successful at introductory programming. In one study, approximately one-third of the students tested towards the end of an introductory programming course did not have an understanding of assignment and sequencing (Ma et al., 2007). Both of these are basic concepts introduced in the first weeks of a course. It should also be noted that the students tested were volunteers and the weaker students may have been less likely to have volunteered to do the test. A test involving the writing of code that would evaluate arithmetic expressions in one multinational study (McCracken, Almstrum, Diaz, Guzdial, Hagan, Kolikant, Laxer, Thomas, Utting & Wilusz, 2001) by 200 novice programming students gave an average student score of just 21%.

A study in 2004 used a twelve question multiple choice questionnaire to test the reading and tracing skills of 556 students who had completed their first semester of studying programming (Lister, Adams, Fitzgerald, Fone, Hamer, Lindholm, McCartney, Mostrom, Sanders, Seppala, Simon & Thomas, 2004). The average student score was 60%, with 48% of the students scoring 7/12 or below, and almost a quarter scoring 4 or below – a result on the same level as choosing options at random.

Those students who do continue on to a second course in programming, and advanced programming courses, may not be any more able than those withdrawing after the first course. There is evidence that a large proportion of students in advanced courses do not understand basic structured programming constructs such as loops, selection and sequence. For example, 11 out of 23 undergraduate IT/CS degree students in an advanced course at an American university using Java could not format basic output correctly, and 4 of the 23 could not correctly code a loop (Zhang, 2010).

These students had previously completed an introductory VB.NET programming course with a passing grade of C or higher.

A survey of attitudes towards programming in final year students – none of which had failed a programming unit previously – contained the following statements showing a deep lack of understanding of programming:

- “I put a break point and try to see where it’s crashing but I don’t understand why it crashed and I don’t know what to ask because I don’t understand what’s happening”
- “I don’t understand which lines do what and I won’t even lie to you”
- “because even if I look at a code sometimes I not really sure what it’s for or, you know, what its purpose is there, kind of thing”

(Rogerson and Scott, 2010)

In 2005, a multi-national, multi-institutional study of 314 students from 21 institutions showed that the majority of graduating students cannot design a software system. Only 9% of students produced “reasonable” partial or complete designs (Eckerdal, McCartney, Mostrom, Ratcliffe & Zander, 2006).

Prospective employers bemoan interviewing graduates who cannot carry out basic programming tasks such as coding loops that count from 1 to 10 (Atwood, 2007; Kegel, 2008).

1.5 Root causes for Low Student Numbers Completing Programming Courses

There is a need for introductory programming to become both

- ***more accessible*** – to increase the number of students entering into IT, and particularly programming courses, and
- ***more successful*** – to increase the number of students continuing through to completion of programming courses with higher levels of skills.

The current climate for this research can be illustrated in Figure 1-1 below, with the numbers of students entering IT being reduced through successive factors until the number of successful programming students being produced is very low.

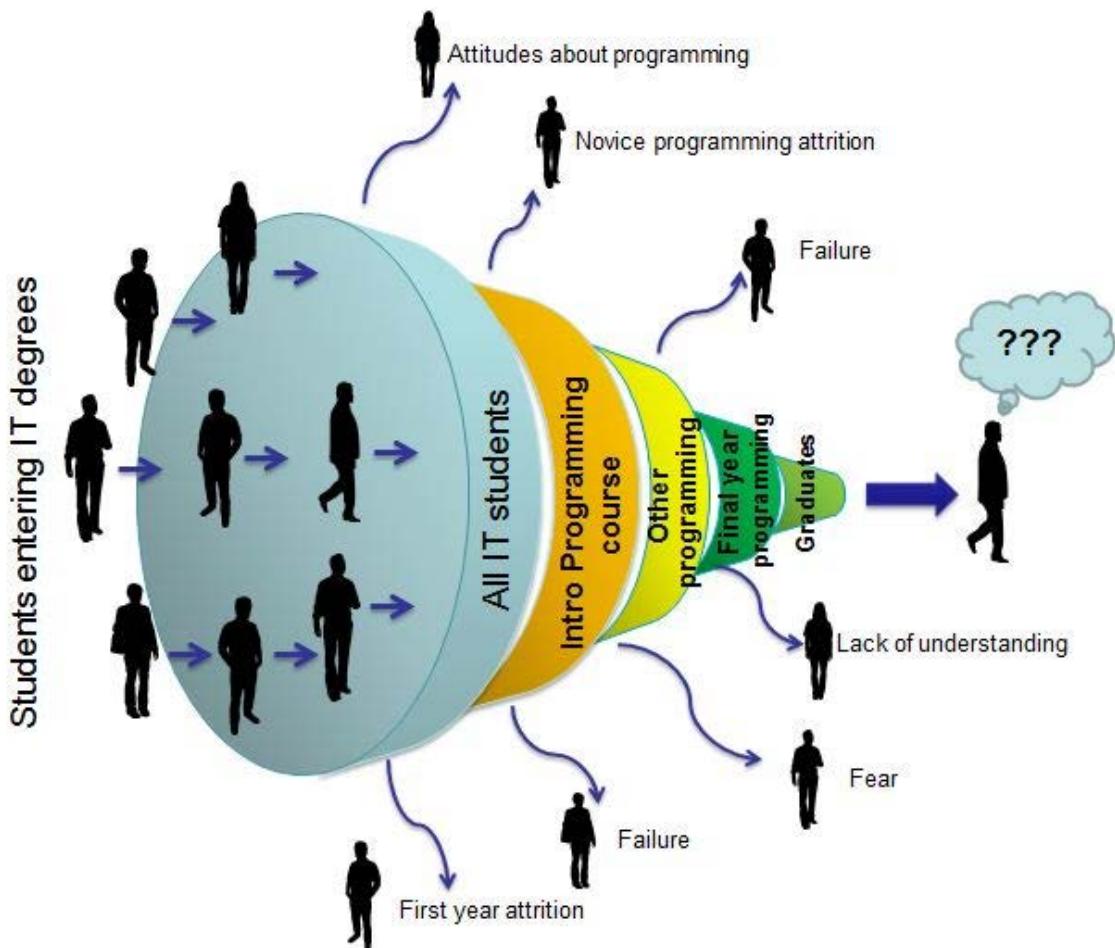


Figure 1-1 Illustration of context for the research

The next three chapters of this thesis comprise the review of the literature on possible causes and solutions to the problem of low student numbers completing programming courses.

Discussion on the possible barriers to the success of introductory programming can be divided into three main areas:

- people factors - such as programming aptitude, age, gender, student's first spoken language, computer literacy, motivation and attitudes towards programming;
- content factors - such as the innate complexity of programming, the programming paradigm chosen, and choice of first programming languages and environments, and
- instructional factors – such as the pedagogy adopted, instructional materials and student activities, access to support and provision of feedback.

The current literature on these three factors is explored in Chapters 2, 3 and 4 respectively in this thesis.

Of specific importance is the issue of the instructional design of first programming courses, which has long been a topic of debate amongst educators (Pears, Seidman, Malmi, Mannila, Adams, Bennedsen, Devlin & Paterson, 2007). The current literature on cognitive approaches to instructional design in complex areas is analysed in Chapter 4. In this context detailed consideration of Information Processing Models of human cognition (Simon, 1979) will be considered, with particular focus upon Cognitive Load Theory (Sweller, 1999). It is argued that these

have particular relevance and application to teaching complex content such as computer programming.

The main thrust of this research is how Cognitive Load Theory principles can be applied to the instructional design of introductory programming courses in order to increase performance and self-efficacy, and thereby increase accessibility to, and success in computer programming.

1.6 Organisation of this thesis

Chapter 1: Introduction

This chapter provides an overview of the thesis, a broad background and justification for the research and a summary of data gathering and outcomes. A brief outline of the thesis is included.

Chapter 2: Student Centred Factors

Chapter 2 provides a review of the current literature surrounding people factors that may contribute to the low numbers of students and poor success rates of introductory programming courses. These include student aptitude, gender, native language, computer literacy, motivation, fear and self-efficacy.

Chapter 3: Course Centred Factors

Chapter 3 provides a review of the current literature surrounding the role of course content in contributing to the lack of success of introductory programming courses. These include the complexity of programming, choice of programming languages, environments and paradigm, and the problem-solving nature of algorithms.

Chapter 4: Instructional Design Factors

Chapter 4 comprises a review of the current literature on methodologies and cognitive theories that may assist in producing introductory programming courses that are more

successful. Cognitive Load Theory (Sweller, 1994) is introduced with particular emphasis on effective ways of teaching complex content which contains a high level of element interactivity.

Chapter 5: IT for Self-Selecting Girls

Chapter 5 describes the methodology and results of an “IT Girls” program that aimed to increase self-efficacy in programming in female high school students, by designing introductory programming workshops using cognitive load principles.

Chapter 6: IT for All Girls

Chapter 6 describes the methodology and results of the above program when conducted in a remote location, with all of the female high school students in a school.

Chapter 7: Australian Universities Study

Chapter 7 describes the methodology and results of a survey of instructors of first year programming courses in Australian universities. Instructors were asked about their use of computer languages, environments and paradigms, and cognitive load measures for themselves, average students and students in the bottom 10% of performance.

Chapter 8: Programming Student Survey

This chapter reports on the methodology and results of a survey conducted with first year programming students, and compares computer literacy, gender, native language and other factors against reported cognitive load and final grade in the unit.

Chapter 9: Controlled Environment Interface Experiment

Chapter 9 reports on the methodology and results of a controlled experiment with high school students, examining the effect of a full versus truncated novice programming environment on performance and attitude towards the difficulty of learning programming.

Chapter 10: Conclusion and Suggestions for Future Research

This chapter provides a brief summary of the major findings from this research. It also restates the limitations of the study and recommends areas for further study.

Chapter Two: Student-Centred Factors

2.1 Introduction

The low success rate of introductory programming courses has led to a large body of literature on the student attributes that may impact on success in learning computer programming. Some of the factors related to academic success that have been investigated have included programming aptitude, cognitive learning styles, gender, age, prior experience, previous experience with mathematics, science and languages, literacy, self-efficacy, class size, motivation and more (Bateman, 1973; Evans and Simkin, 1989; Davy and Jenkins, 1999; Kruck and Lending, 2003; Pillay and Jugoo, 2005; Bennedsen and Caspersen, 2006; Caspersen, Larsen & Bennedsen, 2007; Dehnadi, 2007; Howles, 2007). This is by no means an exhaustive list.

This chapter reviews some of the student characteristics that are commonly regarded to be potential predictors of student success, or rather, those that are of particular interest to this thesis: programming aptitude, general computer literacy, student first language, age, gender, motivation and attitude towards programming.

2.2 Programming Aptitude

It has been noted that students who achieve success in other subjects sometimes fail to succeed in programming (Byrne and Lyons, 2001). Grade distributions of introductory programming courses are often reported to be bi-modal (Dehnadi and Bornat, 2007; McCracken et al., 2001), leading credence to the often-

criticised concept of a (creatively-named) “geek gene” (see Lister, 2010) – an innate ability in some students to succeed at programming.

It has been argued that it may be possible to predict students’ success or failure based on programming aptitude (Bennedsen and Caspersen, 2006; Dehnadi, 2007; Mazlack, 1980), which has led to the development of various tests (Davy and Jenkins, 1999; Psychometrics, 2009). The best known early test of these is the IBM Programmer Aptitude Test (PAT) which in the early 1960’s was administered by approximately 40% of all businesses when hiring programmers (Ensmenger, 2011). Although some early studies suggest that this test could be a significant predictor of success in programming (Bateman, 1973), further studies failed to find a correlation (Mazlack, 1980). Other programming aptitude tests have also been proposed with mixed results (Avancena and Nishihara, 2010).

Dehnadi and Bornat (2007) provided a draft paper with details of a test for programming aptitude which they claimed would “predict success or failure *even before students have had any contact with any programming language* with very high accuracy, and by testing with the same instrument after a few weeks of exposure, with extreme accuracy” (page 1, emphasis in original). They provocatively asserted that it was “useless” to teach programming to “those who were bound to fail” and “pointless” to teach “those who are certain to succeed” (page 1). Despite the apparently silver-bullet-like abilities of this test, the authors admitted that they had no idea how the test worked. Further experimental results by these and other researchers have not supported this test’s ability to predict programming success (Bornat et al., 2008; Lung et al., 2008; Ma et al., 2007).

Another explanation for the apparent ‘bi-modal’ grades in introductory programming has been offered by Robins (2010), who proposed the concept of

‘learning edge momentum’. He proposed that success in acquiring one concept leads to a higher probability of success in learning other closely related concepts and a greater probability of failure if the first concept is not understood. Mathematical models were used to show that this can have a flow-on effect that results in a bi-modal distribution – that is, the tightly integrated nature of concepts in introductory programming leads to student outcomes of higher-than-usual failing or high-achieving grades – the reported bi-modal results.

An observation that may support this theory is made by Byrne and Lyons (2001) in their analysis of factors that impact on success in programming. They observed that prior success in the domains of mathematics and science positively correlated to success in programming. Rather than attribute this to an innate ability, however, they observed that the concepts that students have to master in order to succeed at mathematics are similar to those that must be mastered for computer programming.

There are other models which offer explanation to the bi-modal distribution, without reference to innate genes, by considering the processes of human cognition. For example, the cognitive model of none-to-all learning discusses the role, dynamics and consequences of discrimination learning (Brainerd and Howe, 1978; Restle, 1965). The role of cognition is explored further in Chapter 4.

If students who have previously mastered concepts similar to those used in programming have an advantage over other students, then it may be beneficial to consider whether acquisition of additional basic concepts in computing, mastered as part of general computer literacy, may have an effect on success in introductory programming. Moreover, some students may even have previous programming study experience.

2.3 Computer Literacy/Prior Programming Experience

Students entering university courses, and particularly IT courses, are assumed to have at least basic computer literacy (deWit, Heerwagh & Verhoeven, 2012), and sometimes programming experience (Hardy, Heeler & Brooks, 2006). Despite this assumption by educators, of interest in the literature on this topic is how often students enter computing majors or degrees with no programming experience and limited computer use (for example see Howles, 2007).

The results of studies in this area have been mixed. The effect of prior computer literacy was found to be not associated with student performance in a tertiary business information systems course (deWit et al., 2012). In another study the level of computer literacy was found to be a predictor of ‘computer anxiety’, which was implicated in student performance (Anderson, 1996). An early study (Evans and Simkin, 1989) found some correlation of computer literacy to performance in a programming course. Other studies have shown prior programming class experience is not related to student success (Kruck and Lending, 2003), or have shown very weak correlation (Byrne and Lyons, 2001; Pillay and Jugoo, 2005). Considering the reported lack of success of programming courses in teaching basic concepts (see Chapter 1), it perhaps should not be surprising that students completing previous programming studies do not seem to necessarily have an advantage over novices to programming.

2.4 Student First Language

In Australia international students make up more than half of tertiary IT student numbers (Australian Computer Society, 2011). Programming classes therefore

have a large proportion of students whose first language may not be the same as that used for instruction in the course (English). There are relatively few studies on how this may impact on student success in these courses. Byrne and Lyons (2001) suggest that the thought processes that are used in learning a second language may actually help students learn a programming language, as the process of constructing code with correct syntax is similar to grammatical skills. One of the factors considered in their study on factors that impact on student success was whether students had previously studied a foreign language. However, there was no statistically significant correlation between foreign language points and programming exam score. According to this study, the link between second language skills and programming proficiency “remains unproven” (page 52).

Other studies have indicated that students whose first language is not the same as the instruction language perform more poorly in programming courses (Pillay and Jugoo, 2005). There may be several reasons for this lack of success. General reading comprehension skills may be poor in students studying in a second language, and students with poor reading comprehension have been previously found to perform poorly in programming (Chmura, 1998).

The difference in performance between native language speakers and those who have another first language may also be related to the connections between language, culture and reasoning. Structural differences in language systems lead to non-linguistic differences in the native speakers of the two languages. Nisbett (2003) notes that multicultural students may think differently, and therefore follow through with behaviours based on different patterns of logic and reasoning.

Even students who had studied for several years in another country and language, and therefore could be expected to have adequate reading comprehension,

showed differences in preferences for types of assignments, and modes of thinking (Ellis, 2010). The additional barriers for these international students can be demonstrated by this quote from an interview with a final year programming student whose first language was not English:

“I see a lot of syntax [nervous laugh] symbols and words put together, that are supposed to be meaningful, but to me, I grew up speaking English, and I grew up speaking my home language, and now I have to learn this whole new different language and manipulating it is a nightmare [strained laugh] a little bit of a nightmare.”

(Rogerson and Scott, 2010, page 155)

2.5 Age and Cognitive Development

The number of high school, and more recently primary school, based programs to teach programming (Guzdial, 2001; Henriksen and Kölling, 2004; Kelleher et al., 2007; Resnick et al., 2003) seems to indicate that the age of students itself is not a factor in programming success. Relatively few studies exist that have directly examined the relationship between student age and success in programming (Beise, VanBrackle, Myers & Chevli-Saroq, 2003). Those that do draw a positive correlation between age and programming success usually qualify their results by noting that the correlation “is probably more an indication of a relationship between maturity and achievement than anything else” (Evans and Simkin, 1989, page 1324).

There are, however, a number of studies that have examined the connection between cognitive developmental stages and programming success. These usually use Piaget’s Stages of Cognitive Development (Feldman, 2004; Piaget, 1971a, 1971b) - sensorimotor, preoperational, concrete operational and formal operational stages; or Bloom’s Taxonomy of Educational Objectives (Bloom, Englehart, Furst, Hill &

Krathwohl, 1969). Piaget's formal operational stage is associated with the ability to abstract and form hypotheses and solve problems systematically (Piaget, 1971a). This 'formal operational stage' has been suggested as a "required cognitive characteristic of people for learning introductory programming" (White and Sivitanides, 2002). However, in one study of 145 first year programming students, Bennedsen & Caspersen (2006) found that there was no correlation between abstraction ability and stage of cognitive development (measured by a 'pendulum test'), and programming ability, as measured by final grade in the course.

Although not used to predict performance in the first programming course, Bloom's taxonomy has also been used to classify novice programming teaching and assessment (Lister and Leaney, 2003; Lister, 2000). Neo-Piagetian perspectives have also been used to classify exam questions aimed at testing novices (Corney, Teague, Ahadi & Lister, 2012) with students struggling to answer questions testing concrete operational abilities.

2.6 Gender

2.6.1 *Low numbers*

Female students are less likely to be studying in an IT degree or in a programming class than males. Graduate Careers Australia reported that in 2011 only 15.4% of students graduating in computer science - the category that includes IT - were female (Graduate Careers Australia, 2012) compared to 62.3% of graduating students in all fields of study that were female. The declining number of females enrolling into information technology and computer science programs and progressing to careers in IT has long been recognised as a problem (Anderson, 2009), both in Australia (Australian Computer Society, 1999; Edwards and Kay, 2001;

Whitehouse and Diamond, 2005) and internationally (Barker and Aspray, 2000; Farrell, 2007).

In some countries, however, a larger proportion of females enrol in IT and computer science courses. Ng, Das, Ching & Abdullah (1998) reported that in Malaysia 50% of enrolments in IT and computer sciences were female, while in Qatar, computer science courses have a significant gender imbalance towards women, with 70% of enrolments by females (Guzdial, 2010). These differences suggest that the low enrolments in IT in western-based countries may be a cultural issue rather than the result of some intrinsic difference between men and women.

The apparent rejection of IT study and careers by females appears to start in school. Researchers note that “girls' lack of engagement in ICT is clearly evident at the senior secondary schooling level” (Monash University, 2011). A large study of Victorian students aged 14 to 19 years (Dept of Innovation Industry and Regional Development, 2009) found that only 3% of females surveyed reported a strong interest in a career in ICT.

A number of researchers have pointed out reasons why IT careers should be promoted for females:

- Kelleher et al (2007) stressed the importance of all members of society contributing to new technologies;
- Hu (2008) observed that lower numbers of students leads to a lack of gender diversity in jobs and offers the example of producing video games where employing more females may lead to games that are more attractive to women; and

- Increasing the number of women employed in IT is desirable because the IT industry needs more workers, IT is a desirable area to work in, diversity in the workplace is important, and IT is a tool for solving the big problems (Barker and Aspray, 2000).

2.6.2 Females and Programming

Not only are there a disproportionately low number of females interested in IT, but even when females do choose IT, they typically do not pursue the more ‘technical’ aspects of IT. In the study of Victorian students aged 14 to 19 (Dept of Innovation Industry and Regional Development, 2009) students were asked to indicate career areas in which they may be interested. While 22% of males indicated interest in being a software developer/programmer, only 6% of females did so. ‘Analyst programmer’ was attractive to a mere 6% of males, and only 1% of females.

The movement away from more technical paths for females is reflected in the jobs that were awarded to graduates in Australia. According to a study of ICT skills in Victoria:

“The main jobs that were awarded to females, rather than males, were web or multimedia design positions, trainers (ICT) and sales representatives/executives and ICT sales assistants. These positions typically have the lowest media salary packages...Less than 10% of females were appointed to specialist positions associated with database design and other specialisations, software and web developers and programmers, or telecommunications specialist, field engineer and network planners.” (Dept of Business and Innovation, 2012)

2.6.3 Performance Differences

Females are just as capable as males at performing well at introductory programming. No significant performance difference was shown between males and females in several studies (Beise et al., 2003; Byrne and Lyons, 2001; Mazlack, 1980; Pillay and Jugoo, 2005). One study actually showed that females were better at programming than males (Evans and Simkin, 1989), but gender was not counted as a ‘strong predictor’ of performance.

2.6.4 Progression Rates

There are few papers published that mention female student progression rates in programming. Perhaps indicative of the lack of progression is one study (Bloch, 2000) in a US university which comprised 60% female students composition overall. For the first programming unit 31% were female, which is higher than the average previously reported in other studies, and in the IT industry (Australian Computer Society, 2011). Unfortunately, this encouraging percentage reduced to 18% by the end of the second semester.

2.6.5 Possible reasons for low numbers

One possible reason for women not enrolling or continuing in IT could be the influence of educators. Barker & Aspray (2006) observed that “teachers’ beliefs and attitudes about appropriate behaviors and roles for boys and girls, combined with their attitudes and beliefs about technology, can subtly influence girls to not study computers.” (p.20). Consequently, by the time a girl is ready to decide on a career, gender biases have become fixed and authority figures have persuaded girls that IT is not a suitable career.

Another possible reason for female students to not consider studying IT or programming is lack of information about IT careers (Farrell, 2007). Fisher & Margolis (2002) found that women were more interested in the context of computing than computing as a pure discipline which suggests that when promoting IT to girls the usefulness of IT needs to be stressed.

Beyer & DeKeuster (2006) surveyed 180 computer science students (62 female, 118 male) and 98 MIS students at a small public US university in 2001 and 2002. They concluded that positive role models are important in encouraging women to enrol in computing studies. They also noted that women need to be aware that women are as capable of using computers as are men and that computer science is not just working with computers.

Chalmers and Price (2000) comment that stereotypes about gender and ICT must be challenged and critiqued if female students are to consider IT as a viable career choice for themselves. Images in popular culture can dissuade females from studying IT, in particular the stereotype of the unattractive male computer ‘nerd’ with poor social skills, isolated and working in sterile environments (State Government Victoria, 2001).

The other end of the stereotype pendulum is the concept of the ‘brogrammer’ – a male white computer professional with a fraternity party mind-set (Gross, 2012; MacMillan, 2012; Raja, 2012). Ex- editor of tech website Lifehacker and web/mobile app developer Gina Trapani (2012) observes that these stereotypes are counterproductive at a time when companies are competing for talented programmers and only those that are inclusive of both men and women will be chosen for employment.

Although the term ‘brogrammer’ is relatively tongue-in-cheek, sexism in the IT industry abounds, particularly within strongly technical areas. For example, a presentation (Aimonetti, 2009) given at a Ruby conference that used sexual innuendo with the intent of humorous effect demonstrates an attitude towards women that would make many women uncomfortable. For women in the audience this was “a pointed reminder of disturbing behaviour, and a reminder that such events can happen again at any time” (Fowler, 2009). At a conference in 2010 sponsored by Microsoft (LeMay, 2010) scantily clothed female “hosts” were presented to the several thousand strong audience which resulted in an official apology from Microsoft and much negative publicity.

2.6.6 Encouraging females

It is also useful to understand why some women enroll in IT, as this may indicate possible approaches to take when attempting to attract women. It was noted previously that negative influences from educators can discourage girls so it is not surprising that positive influences can encourage girls. A survey of 275 members of an online site devoted to women interested in the technical side of computing found that school experiences were important and that women were strongly influenced by friends and colleagues (Turner, Bernt & Pecora, 2002). The authors also observed that a degree in computer science was not a necessary prerequisite for an IT career for many women as an “interest and talent in IT emerged gradually and developed over time” (p.16). In a survey of 16 women Kahle and Schmidt (2004) found that half the respondents were motivated by an interest in the field or by the intellectual challenge, and a quarter were motivated by the salary.

2.6.7 Programs to encourage females into programming

Several programs have been initiated by Australian organisations to encourage female students to consider studying IT. These programs typically are conducted ‘on-campus’ so that participants will have access to the required technology.

For example, North Sydney Institute of Technical and Further Education (TAFE) conducts a “Digi-girls” program which is designed to promote ICT to girls and provide career information. The program is aimed at Year 10 girls but has previously also included girls from Years 9, 11 and 12. The Digi-girls program consists of two days of teaching and hands-on exercises, culminating in presentation of the work the girls have done to parents and school representatives. The program is run on the TAFE campuses (NSI TAFE, 2011).

In Queensland, the Commonwealth Scientific and Industrial Research Organisation (CSIRO) Education Program runs the Sunshine Coast Science, Engineering and Technology Expo. The expo program is aimed at Year 5 and Year 9 students and held at the University of the Sunshine Coast’s science faculty. Year 5 students attend a one-day “Labs on Legs” workshop which covers science subjects as well as robotics. Year 9 students are split into separate boys and girls groups and attend a careers panel session as well as participating in a CSIRO workshop (Walsh, 2008).

In Victoria, Swinburne, Monash and Deakin Universities have partnered to conduct a longer-term project in secondary schools. The Digital Divas program runs for a semester as a single-sex elective program for Year 8 students, and at time of writing has been expanded to 12 schools around Melbourne (Fisher, Lang, Forgasz & Craig, 2009). The program uses a mix of engaging curriculum, informal mentoring by

university students and role models of ICT career women in the classroom to encourage students to pursue career paths in ICT.

Each of these programs focuses on schools in city or inner regional areas. Another two programs have attempted to reach more rural and regional areas of Australia.

The now defunct Go Go Gidgits Online Computer Club (Learning Space, 2008) was a part of Education Queensland's Girls and ICT strategy (2003-2008) and was supported by a number of organisations and companies. Female school students in Years 3 to 12 completed computing exercises and then uploaded them to an online space, where they were awarded points and prizes. Before closing in 2008 around 600 members took part in Go Go Gidgits, including students from remote areas, although the program was not directly aimed at remote students.

A more recent program specifically designed to reach regional female students is the Robogals program (Robogals, 2011). Based in Melbourne, Robogals is a student-run organisation that visits schools to introduce girls to engineering and technology through robotics workshops. The Robogals program is restricted by the availability of female student volunteers in regional university areas who are willing to travel to Melbourne to complete the training program and then go out into their regional schools to promote IT.

Other such programs designed to encourage female students into IT have occurred in other countries (for example, see MIT, 2012a) but details have been provided here for Australia to contextualise the focus on gender to the localised Australian situation, and to briefly overview the tyranny of distance in the Australian continent.

2.7 Student Attitudes

2.7.1 Motivation

With respect to factors relating to individuals and learning, Helme and Clarke (2001) state that students need both motivation (the will to learn) and skills (capability) to be successful learners. Motivation in this case can be divided into three types: intrinsic, extrinsic and achieving motivation (Entwistle, 1998).

- Intrinsic motivation is present when there is interest and curiosity about the activity being performed (Deci, 1975; Ryan and Deci, 2000a);
- Extrinsic motivation is present when there is anticipation of a reward for successful completion of the activity; and
- Achieving motivation is centred around doing well or performing better than peers (Ryan and Deci, 2000b)

Carbone, Hurst, Mitchell & Gunstone (2009) found that students who were intrinsically motivated generally displayed higher capabilities in programming. In contrast, those students who were externally motivated (by the idea of passing the unit, for example) or motivated by achievement (getting higher marks) did not necessarily cognitively engage with the content. This research also found that intrinsic motivation can be a result of success in programming, and as one student commented:

“once I got the hang of coding it became very addictive
... hence I gradually developed an interest that I never
thought I would.” (page 30)

Introductory programming is a core (compulsory) course in Computer Science and Information Technology degrees (Association for Computing Machinery, 2008).

Students do not choose to take these units, and may have no intrinsic motivation to learn programming. As observed by Thomas, Ratcliffe and Robertson (2003) many

students major in computer science but have no desire to obtain a job that requires highly developed programming skills, and enter programming courses believing that programming is difficult and un-enjoyable.

Students who do try sometimes feel like they are getting no results from their effort, which is de-motivating, as this example feedback shows:

“This course nearly drove me to insanity. I put so much time into this subject – time that was so desperately needed in my other subjects – with almost always fruitless results.” (Carter, Shi, White, Bouvier, Cardell-Oliver, Hamilton, Kurkovsky, Markham, McClung, McDermott & Riedesel, 2011 page 9).

While motivation is necessary for success for students, conversely, lack of success results in lack of motivation for students. Gomes, Santos & Mendes (2012) conducted a study involving 166 undergraduate students in a first programming course, to determine students’ attitudes towards learning to program. Some of these students were attempting the course for the second time, after failing their first attempt. The study found differences in both motivation and self-efficacy between students who were attempting the course for the first time, and those who were repeating the course. Lack of success led to lower motivation and lower self-efficacy. The same study found a strong correlation between students’ learning performance and their personal prior perceptions of their competence in programming, as well as a correlation between grades and student motivation (page 136).

Another study of students who had failed the first programming course and were repeating it found that these students often had little desire to learn programming (despite being enrolled in a computing degree) and this lack of motivation could have led to their failure in the first course (Sheard and Hagan, 1998).

2.7.2 Self-efficacy

Self-efficacy is defined as the extent to which a person believes they can accomplish goals and tasks (Bandura, 1977), derived from previous performance, observed experience of others, verbal persuasion by others and physiological states. Self-efficacy has been found to affect positively the power someone has to face challenges competently (Bandura, 1982). People usually avoid tasks where self-efficacy is low, and undertake those where self-efficacy is high (Csikszentmihalyi, 1997). As previously noted, in the study by Gomes et al. (2012), there was a positive correlation between students' learning performance and self-efficacy in programming. In another study of introductory programming students, Hauser, Paul & Bradley (2012) found that computing self-efficacy positively correlated to higher performance.

There is a problem when computing self-efficacy is low, perhaps related to previous failure, observation of other students that are struggling or lack of verbal persuasion - by peers and teachers - that programming is achievable. For example, in one study of an introductory programming class, after six weeks of Java classes the *majority* of students felt that they were "way behind the class, with no chance to catch up" (Bloch, 2000). This is an example of the perception of the difficulty of programming classes for most students.

Closely related to this concept of self-efficacy in computing is that of 'comfort level' offered by Wilson and Schrock (2001). Their study of twelve factors that may impact on the success of 105 introductory programming course students found that the 'comfort level' experienced by students had the greatest effect on their success. Comfort level included 'perceived understanding of concepts in the course as compared with classmates' and low 'anxiety level when working on assignments' (page 185), and was therefore closely related to self-efficacy. Similarly, another study

of 380 introductory students who had not completed previous introductory programming (non-repeating students) found that comfort level was most important for success in first programming courses (Ventura, 2005).

A 2010 Working Group report on motivating top students (Carter, White, Fraser, Kurkovsky, McCreesh & Wieck, 2010) describes a different (related) scenario, where able students enter introductory programming classes, find that material which they have already learnt is being covered, disengage with the content, and find that when they re-engage that new content, which they have not previously learnt, has been covered. These students then suffer a drop in confidence and self-efficacy and often fail the unit (page 30). It should be noted that this scenario was based on observations by instructors, rather than on evidence. There is some evidence that teachers may be inaccurate observers of students (Nestler et al., 2012).

2.7.3 Fear

Some researchers have described the anxiety that some students feel when preparing to program as “fear”. Rogerson and Scott (2010) have defined this fear as “experiencing a lack of confidence or apprehension regarding their ability to code or program” (p 148). Their qualitative study examined students who were in their final year of their undergraduate degree. These students had never failed a programming course while registered for their current degree, and all had expressed a desire to continue on to an honours year. Yet most of these ‘successful’ students were experiencing considerable fear when trying to program.

“The net result of these problems is that many students leave the discipline because they perceive programming as a battle against an arbitrary and hostile machine; they may never experience it as the creative activity that it is.” (Felleisen, Fisler & Krishnamurthi, 1999)

It appears from this that in order to succeed in this first course students need to:

- feel interested and motivated towards studying programming,
- believe that they can succeed at programming (self-efficacy),
- feel comfortable in the classroom, and need to
- experience success in programming exercises.

2.8 Conclusion

While there appears to be no ‘programming gene’ that causes students to become good programmers, some individual factors do appear to have some impact on ease of learning programming. Prior programming or computer experience may lower anxiety or fear surrounding programming, and hence raise self-efficacy, which is tied positively to performance in programming courses. Students whose first language is not the language of instruction in programming will most probably find the complexities of learning programming more difficult than those who are native speakers in the language used for instruction. In the literature, the age of students in itself does not appear to be related to programming success, however maturity, through cognitive development, does play a part.

Female students are less likely to study programming in the first instance, and to progress to further studies in programming. Those that do learn programming appear to have similar success as male students, making it important to encourage female students to consider studying programming. Some of the reasons given for low female numbers include the influence of educators, lack of female role models, lack of information about possible careers, and stereotypes of programmers as either unattractive ‘nerds’ with poor social skills, or ‘brogrammers’ with sexist attitudes. It

is important that female students are accessed with timely information, female role models and given the opportunity to experience success in programming, if females are to be encouraged into the field.

Both male and female students need to be internally motivated – that is, motivated by interest and curiosity – to be successful at programming. Success in small programming exercises leads to higher self-efficacy in programming, which in turn leads to greater probability of success and higher performance in programming courses. Self-efficacy can be built through previous successful experiences, observation of others' achievement or verbal persuasion of the achievability of programming. Fear of programming has the opposite effect, and many students experience low self-efficacy and high anxiety while learning to program. Encouraging success in smaller (and easier) exercises initially may help build confidence, self-efficacy and reduce fear in these students.

This chapter has examined internal, *student-centred* factors, that impact on the success of students learning computer programming. External *content-centered* factors, such as course design and programming language and environment choice may also impact on student success. These language and environment factors are explored in Chapter 3 “Course Centred Factors”.

Chapter Three: Course-Centred Factors

3.1 Introduction

Quite apart from student-centred concerns such as motivation, attitude, programming aptitude and demographics, there is much discussion amongst educators on *what* should be taught in introductory programming classes and *how* it should be taught. Even though comprehensive curricula for introductory programming courses have been published by international professional bodies such as the Association for Computing Machinery (ACM) (2008), studies have shown that often curriculum change is driven by vocal individuals in academic departments, student viewpoints and academic fashion, rather than education theory or research (Gruba, Moffat, Sondergaard & Zobel, 2007).

There is obviously much interest in improving course curricula. One large international study of computing education publications (1306 papers) published between 2000 and 2005 found that of a representative sample of 352 papers, around half fell into the “new way to organise a course” category. It could be expected that the driver for this curriculum change would be empirical research, however the same international study found that almost 40% of articles including human participants presented only anecdotal evidence of their position (Randolph, 2007). A similar study of Australasian computing education publications in the years 2004 to 2007 found that 69% were ‘reports’ – a category that included descriptive papers of activities or course changes that had anecdotally resulted in positive outcomes, or those that described interventions combined with simple surveys of student satisfaction (Simon, 2007), rather than controlled studies. Another study of research-only papers (that is,

papers that were not ‘reports’ but were experiments, studies or analysis) in the years 2005 – 2008 in the area of programming education found that papers that investigated student learning in terms of established teaching theories or models of learning were not common, and called for more attention to be given to this area (Sheard et al., 2009b).

It has been observed that educators tend to teach that content with which they themselves feel most comfortable (Koffman, in Bruce, 2004; Decker and Hirshfield, 1994), and that often this includes teaching techniques which they have learned from observation and experience as former undergraduate students (Lister, 2008).

With this in mind, this chapter reviews the literature regarding the complexity of teaching introductory computer programming, some of the teaching tools and methodologies that have been used in attempts to make learning more successful, and the role that paradigm, programming language and environment choice may play in the success of teaching and learning introductory programming.

3.2 Complexity of programming

Why is learning to program so difficult? A primary reason may be that novices to programming need to learn *many* interrelated aspects to programming, topics and skills in an introductory course.

The ACM Curriculum Guidelines for Undergraduate Programs in Information Technology (2008) specifies the “Programming Fundamentals” that need to be covered as core topics in any Information Technology degree. At the minimum, this Curricula recommends that “*most of the topics in the Fundamental Programming Constructs (PF2) unit along with some of the topics from Fundamental Data*

Structures (PF1) and Algorithms and Problem Solving (PF4)” should be covered in a first course in programming (page 32).

Their unit “Fundamental Programming Constructs” (page 103) includes the topics:

- Basic syntax and semantics of a higher-level language
- Variables, types, expressions, and assignment
- Conditional and iterative control structures
- Simple I/O
- Functions and parameter passing
- Structured decomposition
- Recursion

“*Fundamental Data Structures*” (page 103) includes the topics:

- Primitive types
- Arrays
- Records
- Strings and string processing
- Data representation in memory
- Pointers and references
- Linked structures
- Knowledge of hashing function
- Use of stacks, queues
- Use of graphs and trees
- Strategies for choosing the right data structure

“*Algorithms and Problem Solving*” (page 105) includes:

- Problem solving strategies
- The role of algorithms in the problem-solving process
- Implementation strategies for algorithms
- Debugging strategies
- The concept and properties of algorithms

As an example, perusal of the two introductory programming courses offered by Southern Cross University shows that these courses follow the ACM Curriculum Guidelines. Course 1 covers an introduction to applications development, variables and constants, decisions and conditions, modularisation (functions), multiform projects (GUI), repetition, arrays, database files, other files, user interface design, and including multimedia in projects. Course 2 (also an introductory course) covers writing a program, debugging, stepwise refinement and problem solving, selection and iteration control structures, arrays, string handling, file handling, passing parameters, library functions and user defined data types (Southern Cross University, 2012, 2010).

This is an extensive set of material for novices to cover, however, as the Curricula states: “Programming courses require a student to master a set of concepts and also to develop sophisticated skills” and yet advises that “limiting the demands for concept learning in this course will allow time for substantial work by students on developing programming skills.” (page 32).

The above concepts comprise the prescribed curriculum, but what is actually taught? An international cross-institutional survey of teachers of introductory

programming courses (Schulte and Bennedsen, 2006) found that in addition to the basic topics such as algorithm development, selection and repetition and variables advised by ACM, teachers also routinely covered information hiding/encapsulation, code reuse, unit testing and choosing test data, and debugging (also see Ahmadzadeh, Elliman & Higgins, 2005; Murphy, Lewandowski, McCauley, Thomas & Zander, 2008).

The same survey showed that 85% of respondents covered some object-oriented concepts in the introductory course, even if they were not teaching in the objects-first paradigm (Schulte and Bennedsen, 2006). Added to this, many educators agree that design of programs needs to be taught in this first programming course (Dale, 2005), not merely language syntax.

The lack of success of many students in programming courses indicates that students have trouble learning these interrelated topics. One reason for this difficulty could be that the traditional instructional design of programming courses requires two different learning styles to be used at once (Jenkins, 2002) : surface learning - tacit acceptance of facts and memorisation, often automated - and deeper learning - critical analysis of new ideas, linking to previously understood concepts and understanding and application to novel situations (Marton and Säljö, 1976). Novice programmers often need to interpret the problem statement, use the computer effectively (which may include using an integrated development environment), create the program in a file with the correct syntax of the chosen programming language, using the correct semantics, structure and style, compile their program, find the output, test it and remove bugs. *While* doing this, they also need to learn algorithms and problem-solving!

Du Boulay (1986) defined five areas of understanding for students that needed to be mastered to be successful in programming:

- “General orientation” – the idea of a program, what it is and what it can be used to accomplish;
- “The notional machine” – an abstract model of the computer as it executes programs. This is dependent on the programming language being used;
- “Notation” – The syntax and semantics of the programming language that has been chosen;
- “Structures” – a structured set of solutions to standard problems – related knowledge;
- “Pragmatics” – the skills that must be mastered to accomplish programming such as planning, developing, testing, debugging etc.

With a typical first course in programming, these multiple areas of understanding must be gained *at the same time*, and then applied *at the same time* to successfully program a solution to a problem.

The necessity of concurrently teaching the principles and concepts of programming *and* teaching the details of a programming language and environment is bemoaned by educators such as in the following:

“every time I wanted to introduce an interesting new principle of programming, I had to spend precious lecture time on language syntax, deciphering cryptic error messages, and using the development environment.” (Bloch, 2000)

Buck and Stucki (2000) have remarked that most introductory programming courses present too much too soon and overwhelm novices. Faced with the difficulty of presenting design and problem-solving activities at the same time as teaching

notation and pragmatics, many instructors default to focusing on the actual programming language and the coding of a program, rather than more difficult deeper structural and abstract design-oriented activities (Schulte and Bennedsen, 2006).

The primary difficulty of teaching programming may well reside in the necessity of teaching several interrelated non-trivial concepts all at the same time, all of which are new to a novice. A novice needs to learn and apply all of these concepts, all of which interact, and often which must be performed all at the same time. This concept of “element interactivity” (Sweller, 1999) and its retarding impact upon learning is discussed in detail in Chapter 4 of this work.

Various tools and methodologies have been developed in an attempt to help students’ understanding and learning of each of these areas that are required for computer programming. These tools and methodologies are explored in the next sections of this chapter.

3.3 Teaching tools

Gomez-Albaran (2005), in a review of software tools that are used to support the teaching of programming, classified these into four categories:

- reduced development environments;
- example-based environments;
- visualisation or animation-based tools; and
- simulation environments.

Some examples of *reduced development environments* and *simulation environments* will be further examined in Sections 3.7.3 and 3.7.4 of this chapter so will not be discussed further here.

Example-based environments are based on the tendency of humans to use problems that they have already solved as a “reference-point” or template to solve new tasks. Example-based environments consist of a set of programming examples, combined with tools to extract relevant examples to the task at hand (Guibert and Girard, 2003; Guibert et al., 2004; Neal, 1989; Repenning and Perrone, 2000). A criticism of these tools has been that they generally require novices to recognise similar programming problems (deep structures) – a task that may not be possible for students with lack of experience and knowledge in programming (Brusilovsky and Weber, 1996).

Many *visualisation and animation tools* have been developed to try to help students understand data and algorithm execution. Visualisations provide a window into the program execution, and these are usually shown in an animation with the intent of helping the student to construct mental models of the process and understand how a required result is achieved (Boisvert, 2006). Visualisations used in this way are not new (for examples, see Boisvert, 1995; Byrne et al., 1999; Doube, 1998), however the effects on student learning have had mixed empirical results.

Some studies have shown positive results on student learning (Blank et al., 2003; Crosby and Stelovsky, 1995; Kasmarik and Thurbon, 2003; Rößling and Freisleben, 2000), while others have discovered no apparent advantage (Balci et al., 2001; Stasko and Kraemer, 1993; Thomas et al., 2004). One older review of experimental studies using animations and visualisations came to the conclusion that the way that students use visualisation tools has a greater impact on student learning than what the actual technology shows to the student (Hundhausen, Douglas & Stasko, 2002). A more recent review of algorithm visualisation tools found over 500 tools of varying quality were available to educators (Shaffer, Cooper, Alon, Akbar,

Stewart, Ponce & Edwards, 2010). The same paper found that the experimental literature suggests that the most important factor in success is engagement of the students' attention, although the design of the animations, and whether they were coupled with text also had bearing on success (Tudoreanu and Kraemer, 2008).

3.4 Methods

Literature on teaching introductory programming is extensive, and to complete a review of all the differing methods that could be used to teach an introductory programming course is beyond the scope of this chapter. For the purposes of this thesis, only three relevant methods have been examined: the use of design patterns, roles of variables, and emphasis on problem-solving as a path to learning content.

3.4.1 Design Patterns

Related to the example-based environments discussed in Section 3.3 is the idea of *design patterns*. A design pattern is a reusable description or template that will help solve a commonly occurring program within a given context (Beck and Cunningham, 1987). It has been suggested that as the complexity of programming languages grows, students will need to recognise standard design solutions in order to program effectively – in effect a ‘cognitive map’ that will help with problem solving (Porter, 2006).

Although design patterns are often used by experienced programmers (Gamma, Helm, Johnson & Vlissides, 1995) for larger programs, Proulx (2000) created a design pattern framework that could be used in introductory programming teaching with small problems. This framework was based on fundamental programming and design patterns, including patterns for reading and writing data,

counting, and selecting from options. In her method, novice programmers were presented with an introduction to the problem, an explanation of the idea and related concerns, and then a presentation of the solution in the context of a chosen programming language. It was argued that this would help novices to build mental models of the common structures used in programming.

3.4.2 Roles of Variables

Another method that is somewhat related to design patterns, and intended to help students build abstractions of common concrete processes, is an approach called “Roles of Variables” (Sajaniemi, 2002). Sajaniemi examined novice programs that were written by experts in an effort to identify generic roles that describe variables and their successive values, and identified nine common ‘roles’ that these variables can play. For example, a variable role called ‘STEPPER’ is assigned values in a predictable order (eg: 1, 2, 3, 4 ... or 0, 2, 4, 6... and so on). Experts understand these roles as part of their tacit knowledge of variables, but making these roles explicit means that they may be formally taught to novice programmers to improve mental models and hence program comprehension (Kuittinen and Sajaniemi, 2004). There has been reported success using this approach across institutions and using various programming languages including Python and Java (Nikula, Sajaniemi, Tedra & Wray, 2007; Sorva, Karavirta & Korhonen, 2007).

3.4.3 Problem Solving

ACM defined curricula includes ‘problem solving strategies’ as one of the basic topics in a first programming course (Association for Computing Machinery, 2008). Problem solving is implicitly or explicitly included in most (if not all)

introductory programming courses (Jarusek and Pelánek, 2012; Middleton, 2010; O’Grady, 2012; for some examples see Rajaravivarma, 2005; Tasneem, 2012).

In a post on a computing educators forum, in answer to a question “May you please suggest ways to develop the programming skills of my students?” (Raouf, 2011), the overwhelming number of responses referenced problem-solving as the best way forward. Representative answers included “... [give] a lot of problem solving in your programming of choice to your students. The more time they spend on solving problems will enhance their programming logic skills.” (Bogalin, 2011) and “... I think the way to teach logical thinking is to require logical thinking. That is often accomplished using problem-based learning methods.” (Selig, 2011).

Problem solving in the context of introductory programming consists of providing students with a problem description, asking the students to break this down into smaller sub-problems, implementing the subprograms using code and programming environment of choice, testing and then re-assembling these into a solution for the problem (McCracken et al., 2001). Students are given an example, and then given a number of similar and related problems to complete, from which they are to learn.

Learning based on problem solving has its roots in discovery learning, and constructivist theory (Bruner, 1961; Piaget, 1973), which posits that students will learn more if they discover or construct knowledge for themselves by exploring learning materials, rather than being explicitly instructed. Much of the literature on teaching introductory programming focuses on problem discovery and analysis (for examples, see Hovis, 2005; Lomako, 2007) and encouraging exploratory development (Braught, 2005). The effectiveness (or otherwise) of learning by problem-solving for novices in domains with high element interactivity is examined in Chapter 4.

3.5 Paradigm Choice

Various programming paradigms can be used while teaching introductory programming, including procedural/imperative, object-oriented ("OO") and functional. The functional paradigm is used in a very small percentage of courses (de Raadt, Watson & Toleman, 2004) and will not be further examined here. Although object-oriented programming needs to be included in the programming curriculum (Association for Computing Machinery, 2008), hot debate exists amongst programming educators about when to introduce the paradigm (Bruce, 2004). A traditional teaching approach is to teach the majority of the first programming course using the procedural/imperative paradigm, thereby covering the basics of program control - sequence, selection and iteration (Dijkstra, 1972), as well as other basic concepts. This is followed by the introduction of object-oriented concepts in the last few weeks of the course. This approach is often entitled "objects-later" (Ehlert and Schulte, 2009).

Advocates of the 'objects-later' approach argue that introducing object-oriented concepts early in the introductory sequence adds another layer of abstraction (and difficulty) to an already difficult learning experience (Koffman, E in Bruce, 2004). Students are "exposed to very complex and often subtle concepts before they have any adequate contextual foundation upon which to base comprehension" (Buck and Stucki, 2000). Other educators point out that basic structured design and programming constructs are used within methods in OO, and so gaining structured programming skills is a prerequisite for learners who wish to master object-oriented programming (Burton and Bruhn, 2003; Zhang, 2010).

This traditional objects-later approach, however, has been criticised because it requires students to learn in the procedural/imperative paradigm and then requires them to "paradigm shift" to understand concepts such as inheritance and polymorphism that exist within the OO context (Decker, 2003; Decker and Hirshfield, 1994; White and Sivitanides, 2005).

An alternative to this approach is to teach "objects-first", which involves teaching about classes and objects early and emphasising object-oriented concepts (Cooper, Dann & Pausch, 2003; Kölling, 2006a). Although this avoids the paradigm shift required as part of "objects-later", some educators believe that teaching objects-first is excessively difficult and mostly unsuccessful (Ramalingam and Wiedenbeck, 1997; Reges, 2006; Thomas et al., 2004). To counter this perceived difficulty, instructors have worked on ways of organising content so that learning objects-first is less complex, for example, by separating out content that involves 'client' techniques - where classes are *used* - for the first course, and following this by 'implementer' techniques - where classes are *created* - in a second course (Roumani, 2006).

The increased difficulty of teaching with objects-first compared to objects-later is considered to be true on anecdotal evidence, however there is not a consensus in the research literature about which of these two approaches is actually most difficult for students (Lister, Berglund, Clear, Bergin, Garvin-Doxas, Hanks, Hitchner, Luxton-Reilly, Sanders, Schulte & Whalley, 2006). Perhaps the perceived difficulty of teaching objects-first is why the objects-later approach has been preferred even when teaching an OO programming course. A survey of 52 educators in the UK (Chalk and Fraser, 2006) showed that most used objects-later for this purpose. The prevalence of objects-later was also linked to language choice, with 80% of courses

that used C# or C++ using the objects-later approach, compared to 55% of courses that used Java, which is a completely object-oriented language.

Empirical studies with secondary school students comparing the two approaches show that retained general programming knowledge is the same for the objects-first and objects-later approach (Ehlert and Schulte, 2009). The same study showed that objects-first has some advantages for students in understanding the object-oriented paradigm, although the objects-first course was perceived to be more difficult by these students.

Proponents of the objects-first approach agree that it is challenging, and requires pedagogical tools and instructional material that will support the approach (Bruce, 2004). It has also been argued that part of the difficulty of teaching objects-first has been the result of instructor choice of languages and programming environments (Kölling, 2006b, 1999a), and that this difficulty and complexity could be reduced with the use of specially-designed supportive environments (Cooper et al., 2000; Kelleher and Pausch, 2005; Kölling, 1999b; Zhu and Zhou, 2003).

3.6 Language

3.6.1 *Language and Paradigm*

Regardless of what paradigm is chosen for a first introduction to programming, often there is a mismatch between the paradigm used and the choice of language. In the study referenced above (Chalk and Fraser, 2006), 55% of courses using Java (an object-oriented programming language) were using a procedural approach. In a census of introductory programming courses at Australian universities, de Raadt, Watson & Toleman (2002) found a similar mismatch, where 81% of students were taught using OOP languages, and yet over half of all students were

initially taught using a procedural approach.

3.6.2 Which language?

A large survey of introductory programming instructors was conducted in 2004, based on the ACM Special Interest Group in Computer Science Education (SIGCSE) email list and a list of educators gleaned from text publishers contact lists, to determine the content and topic emphasis of introductory courses in computing (Dale, 2005). Of the 351 responses (10% of the invitees) 156 (44%) used Java, followed by 32% using C++. The SIGCSE respondents used Java (65% of courses) more than twice as often as C++. Many (76%) reported teaching Java as a procedural language for at least part of the course.

The prevalence of Java in introductory courses is also seen in other surveys. In 2006, in an international survey of introductory programming teachers, Java was used as the first language by 58% of respondents, followed by C++ at 18% and Pascal at 9%. (Schulte and Bennedsen, 2006). In 2003, a re-run of the 2001 census of Australian university introductory courses was expanded to include universities from New Zealand (De Raadt et al., 2004). Java was again the most popular language in both countries, with 29 courses in Australia (44.4% of students), and 5 courses in New Zealand (60.4% of students), followed by C++ (18.7%) and VB (17.3%) respectively. Why Java is chosen as the first language is a matter of conjecture – de Raadt, Watson & Toleman (2003) found that choice of language was made for a variety of reasons including pedagogical reasons and industry relevance.

This reliance on Java may be a stumbling block for novice programmers, as Java has many syntax rules and requires a considerable amount of overhead code (eg: a main method) to produce a simple beginning program up to a few hundred lines

(Bloch, 2000; Pears et al., 2007). It also requires high precision as one small syntax error can produce unexpected results, or at worst, make a program non-functional (Jenkins, 2002). The necessity to learn a large number of syntax rules is part of the overload experienced by students, as indicated by the following student quotes:

“I have no problems with the concepts. I grasp them quite easily. I think it's [the] transferring - trying to picture it in my head, the way it works with the theory, and then trying to translate that to code and make it work the way it's working in my head.” (Rogerson and Scott, 2010 p 156)

“I find when I think about it I've got to take my thoughts and then translate them into code, and that's an active process.” (p 163)

Bloch (2000) notes that while objects-first may not be too difficult for beginning programmers, Java may well be, which begs the question: what are the reasons for choosing Java, and other languages? Jenkins (2002) suggests that languages for a first programming course should be chosen for pedagogical suitability rather than industry acceptance. Yet in the 2001 census of Australian courses, 33 out of the 57 courses indicated that their current language was chosen for the reason of industry relevance, marketability and student demand, compared to only 19 which indicated 'pedagogical benefits of the language' (de Raadt, Watson & Toleman, 2002) as the reason for their choice.

3.6.3 How many languages?

There is some evidence to suggest that learning multiple languages increases students' ability to learn new languages by emphasising the similarities between language constructs (Maheshwari, 1997). However, this study was conducted with second-year students, not introductory students. For novices to computer

programming the addition of additional syntax rules and other factors involved with learning multiple professional languages may likely challenge their cognitive resources, already overwhelmed by the complexities of learning introductory programming.

3.6.4 Mini languages and languages for children

Some small, simplified languages - sometimes called "mini-languages" (Brusilovsky, Calabrese, Hvorecky, Kouchnirenko & Miller, 1997) - have been developed especially to support the initial steps in programming education. These may have limited functionality, but aim to provide a foundation for learning a more general purpose language, such as Java, C# or C++. One of the earliest of these is Logo (Papert, 1980), with its 'turtle graphics'. An 'actor' - in the case of Logo a 'turtle' represented by a triangle - is controlled by a small set of commands, and a program can be as short as one line of code. Papert believed that Logo would allow children to construct knowledge about programming by discovery learning (see Section 4.2.1).

Many other mini-languages exist, including several variants of Karel the Robot (Borge et al., 2004; for examples, see Buck and Stucki, 2001; McDowell, 2002; Xinogalos et al., 2007). Often these are dependent on the creation of an environment in which to use the language, or the language is integrated with the development environment and cannot easily be used without it - for example, KPL (Schwartz, Stagner & Morrison, 2006), Scratch (Maloney, Burd, Kafai, Rusk, Silverman & Resnick, 2004), Squeak (Guzdial, 2001; Ingalls, Kay, Kaehler & Wallace, 2007) and XNA's Kodu (Caron, 2009; MacLaurin, 2009), many of which are designed to be used by younger (school aged) students as an introduction to programming.

3.7 Environments

3.7.1 No environment?

Many instructors choose to use simple editors and command line compilers, and do not use a development environment or other tools unless the language demands it (De Raadt et al., 2004). In 2003¹, the Australian and New Zealand programming course census indicated that 45.4% of Australian courses and 55.7% of New Zealand courses fell into the 'no-environment' category. Instructors who choose to not use an environment cited several reasons: additional cost for students, the additional (wasted) time required for students to learn about the environment, and the view that using an environment obscures the steps in the programming process (De Raadt et al., 2002). Another possible reason may be that more senior educators have themselves initially experienced programming in a command-line environment before more complex IDEs were available, and may be more comfortable using this approach.

3.7.2 Professional or Teaching?

Using professional development environments in the introductory teaching of programming appears to solve the second of these difficulties (time wasting), as students will be required at some stage to learn how to use a professional environment before they graduate as a professional. From this viewpoint, teaching using a professional development environment from the outset will 'save time' as the student will only need to learn the details of one environment.

¹ A more current percentage of Australian programming courses that use no development environment is given in Chapter 7.

A criticism of this approach is that professional environments expose students to all of the chosen language from the beginning and may have features and/or tools, such as error messages for example, that are incomprehensible to novices (Felleisen, Findler, Flatt & Krishnamurthi, 1999), because they are designed and directed at experts. Further, professional environments may require students to spend considerable time learning how to use the environment including setting many options to create even a simple program (Bloch, 2000). The availability of the whole language, with no protection, or "quarantining", from syntax errors, allows students to accidentally invoke advanced features.

For example, a misplaced curly brace in Java can create an "inner class", which CS1 students don't need, and produce a remarkably misleading series of error messages. (Bloch, 2000 p 160).

As previously noted, Java (the most popular language for first year programming courses) has a large overhead of code that is needed to create the simplest programs, including a static "main" method, and professional environments generally do not protect novices from this overhead.

In response to this problem, several learning environments have been created, with the purpose of providing a simplified introduction to computer programming in a particular language and paradigm.

3.7.3 Environments for specific languages

3.7.3.1 BlueJ

BlueJ is one of the most popular environments for teaching introductory programming, and in 2003 was used by 11.2% of introductory courses in Australia and 12.7% of courses in New Zealand (De Raadt et al., 2004). An introductory survey of introductory programming teachers in 2006 showed that BlueJ was used by 27% of

respondents - compared to 14% who used the professional programming environment Eclipse (Schulte and Bennedsen, 2006). The BlueJ website lists approximately 1000 universities which use the software worldwide (Kölling, 2012), and at the date of writing BlueJ has been downloaded over 10 million times (Utting, Brown, Kölling, McCall & Stevens, 2012).

BlueJ is built on top of a standard Java Software Development Kit (SDK), but presents a front-end that facilitates discussion of object-oriented design (Nourie, 2002), through displaying classes, inheritance and objects visually (see Figure 3-1 below). Visualisation is one of the three cornerstones of the BlueJ architecture, along with interaction directly with objects, and simplification of the environment (Kölling, Quig, Patterson & Rosenberg, 2003). Objects can be manipulated directly to help students understand methods and properties.

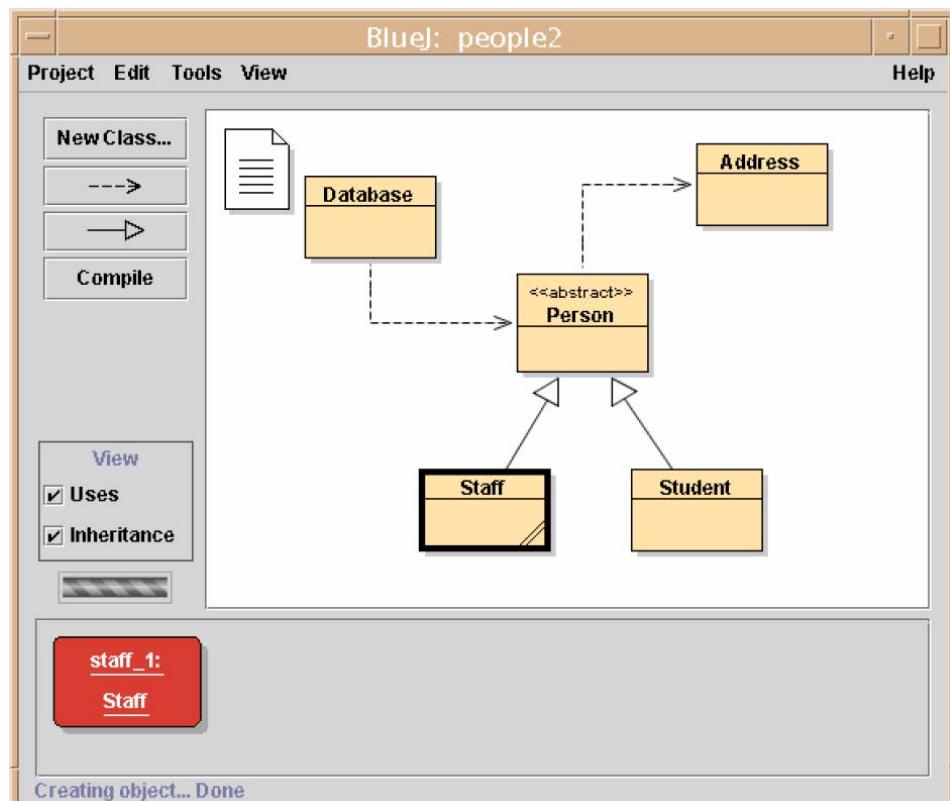


Figure 3-1 The BlueJ environment, showing classes and objects (in red)

Students do eventually need to write Java code, however they are encouraged to first learn to execute programs, then read code, then modify code and finally extend projects (Kölling et al., 2003). By the end of the first course (usually a semester-long unit) students are expected to be able to design and write small programs, debug and test. As BlueJ uses a standard version of Java, standard notation and terminology, it is argued that the transition to a professional environment will be simplified by removing the necessity of learning new syntax (Van Haaster and Hagan, 2004). It is expected that students will transition to a professional development environment after the first year of programming within the BlueJ environment (Kölling, Quig, Patterson & Rosenberg, 2003).

Another teaching environment from the author of BlueJ, and which also uses Java as its language, is Greenfoot (Kölling, Utting, McCall, Brown, Stevens & Berry, 2012).

3.7.3.2 Greenfoot

Greenfoot uses standard Java syntax in a graphical environment that aims to help students visualise classes and their relationships (Henriksen and Kölling, 2004), while removing some of the complexity, such as the static main method. The main window of Greenfoot, in a demonstration 'Asteroids' game, can be seen in Figure 3-2 below. Greenfoot, like BlueJ, shows the class structure visually, but objects are instantiated on screen in the simulation window on the left.

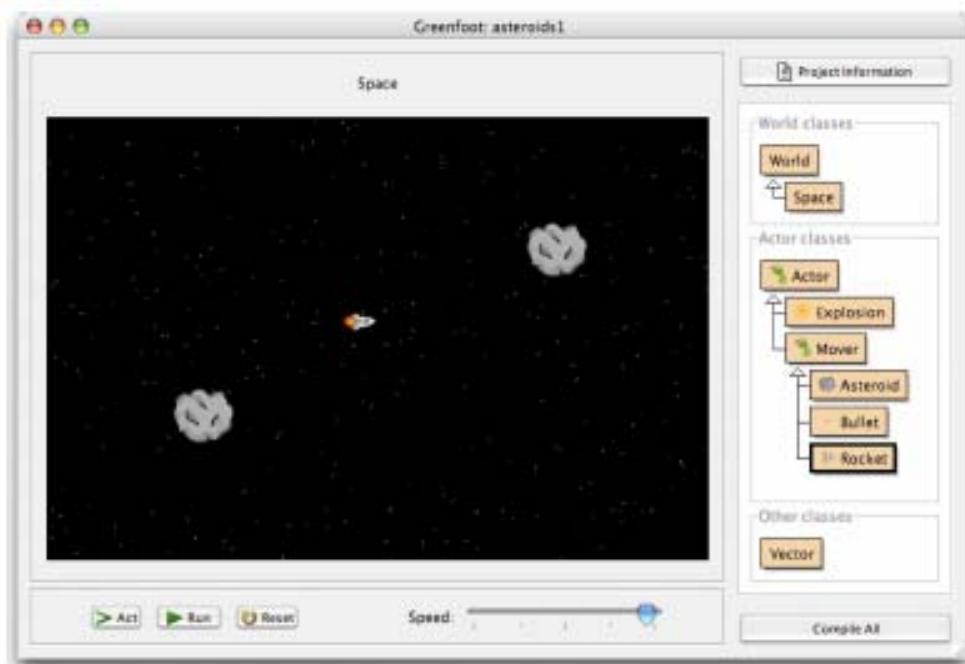


Figure 3-2 Greenfoot main window, showing classes and inheritance

Each class can be edited in a Greenfoot editor window and students are required to manually type Java code, however skeleton code for each class is automatically created (see Figure 3-3 and Figure 3-4 below) so the beginning input from the student can be as simple as a one-line method call (Kölling, 2008).

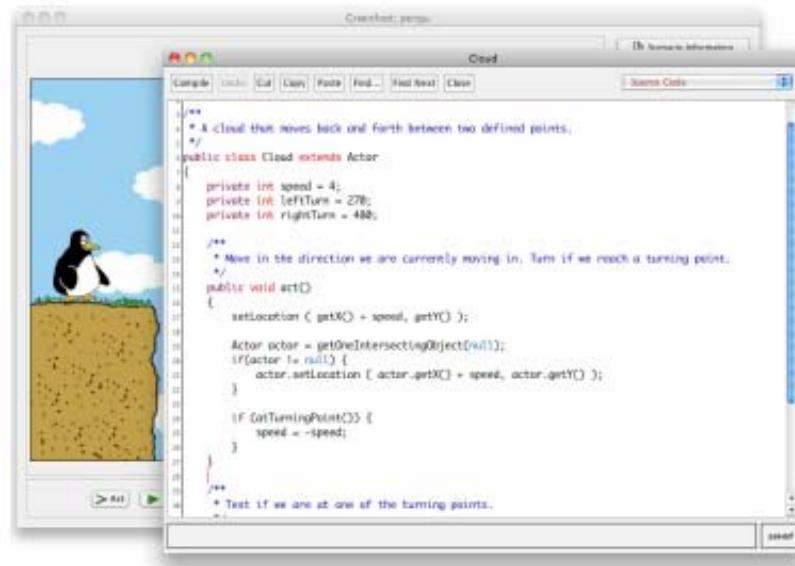


Figure 3-3 The Greenfoot editor window

```

import greenfoot.*;
/**
 * Write a description of class MyActor here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */

public class MyActor extends Actor
{
    /**
     * Act - do whatever the actor wants to do.
     */
    public void act()
    {
        // Add your action code here.
    }
}

```

Figure 3-4 Automatically created Greenfoot class skeleton

The teaching-support goals of Greenfoot are (Kölling, 2010):

- visualisation of important concepts of object-orientation;
- interactions that demonstrate programming concepts;
- representations in the environment that reflect the underlying programming model, leading to a consistent mental model;
- the ability to create a 'first success' without knowing much syntax;
- reducing complexity to avoid cognitive overload;
- teaching support infrastructure to be available.

Greenfoot does not include many of the tools found in professional environments, such as version control, unit testing and refactoring support, as it is purposefully kept as simple as possible for students (Kölling, 2008).

While BlueJ was created to be used in a first programming course at university, Greenfoot is aimed at students from 14 years onwards - high school introductory courses, as well as university first programming courses (Utting, Cooper, Kölling, Maloney & Resnick, 2010). As Greenfoot uses the full Java language, the environment can be used for advanced courses as well.

3.7.3.3 DrScheme

An earlier environment developed to help university students learn programming in Scheme is DrScheme (Felleisen, Fisler & Krishnamurthi, 1999), now renamed to "Racket" (Felleisen, Flatt & Findler, 2008). The interface for this environment is designed as a set of layers that increase in complexity as novice programmers learn concepts. Beginning layers contain a subset of the language and error messages that are not critical are suppressed (Krishnamurthi, Felleisen & Duba, 2000). Students type code in a simplified user interface that only contains two panes - coding and execution (Findler, Clements, Flanagan, Flatt, Krishnamurthi, Steckler & Felleisen, 2002).

One of the main features of this environment is that debugging is fully integrated - on execution if there is an error, program execution halts and the offending code is highlighted (see Figure 3-5 below).

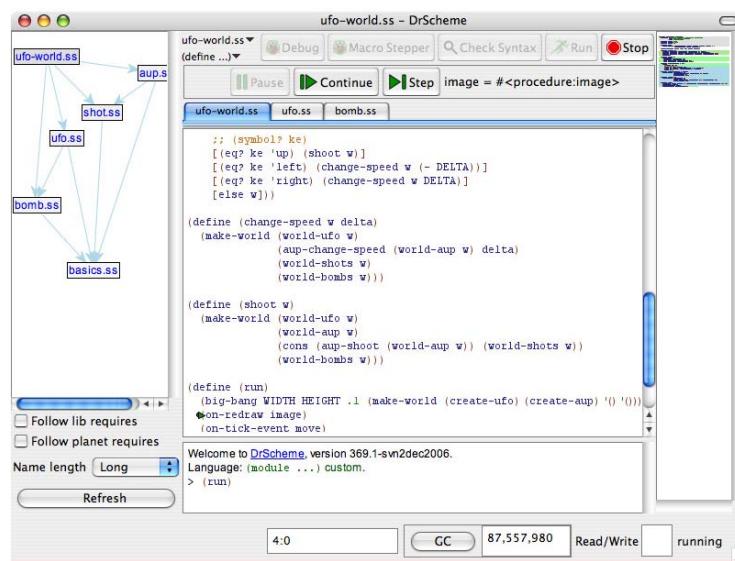


Figure 3-5 DrScheme debugger

Error messages are also tailored to student knowledge level - starting out as more detailed and becoming more expert-oriented as novice knowledge increases.

DrScheme has been successfully used as an introduction to programming, followed in successive courses by Java (Bloch, 2000).

3.7.3.4 Jeroo

Another environment designed as a first introduction to object-oriented programming while using simplified standard languages is Jeroo (Sanders and Dorn, 2011). Jeroo allows students to create Java-like, Python or VB.NET programs that create and animate up to four kangaroo-like animals that have the ability to interact with their surroundings (Sanders and Dorn, 2008). Jeroo supports "objects-first" and is thought to be effective because of the visualisation of program execution, fundamental control structures and objects, and because it has storytelling aspects that particularly are attractive to girls (Sanders and Dorn, 2003a).

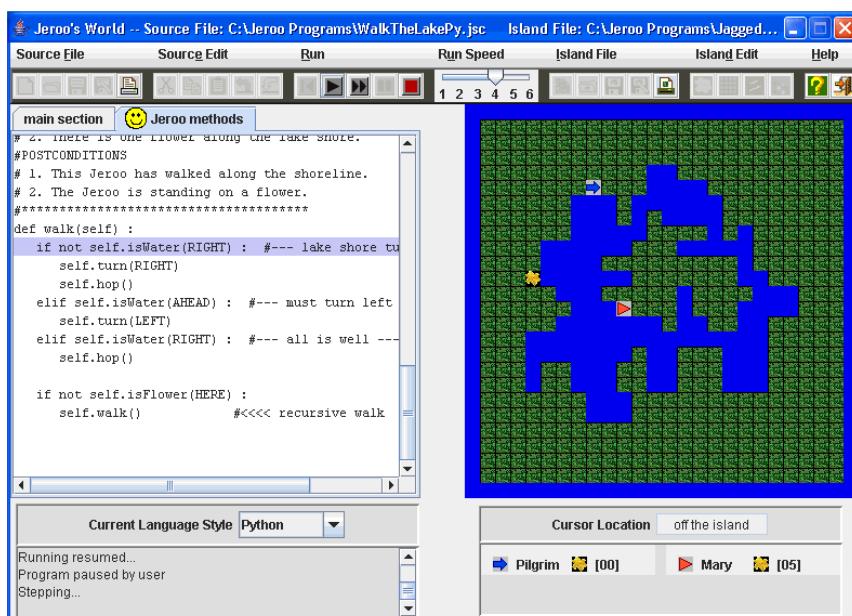


Figure 3-6 Jeroo screenshot showing executing Python program

Jeroo has a simplified interface that concentrates on showing all aspects of the interface at all times (see Figure 3-6 above), and highlights lines as programs are executed to help students build their mental models (Dorn and Sanders, 2003).

Jeroo is often used with a reduced language which is very similar to Java, C++ and C#, but has very limited features. There is only one class in this language - a "Jeroo" class - and no data types or variables other than references to Jeroo objects (Sanders and Dorn, 2003b). More recent developments have seen the introduction of a subset of Python and a subset of VB.NET (Sanders and Dorn, 2008). Jeroo is recommended to be used for a short time (approximately 4 weeks) at the beginning of the first programming course, then to have students transition to another environment and a more professional language (Dorn, 2008). Transition to Java has been 'seamless' for university students who are novices to programming, according to the creators of the environment (Sanders and Dorn, 2003b).

Jeroo is an example of a *reduced development environment*, as well as a *simulation environment* (see Section 3.3). While it has been included in this section on environments for specific languages because it can use a subset of Python or VB.NET, it could be included in another class of environments - those that are specifically created for learning, and do not necessarily use a particular professional language.

3.7.4 Learning Environments

Other learning environments exist that are not based on a subset of a particular language, or that do not require students to actually type code. These environments are often simulation environments that help students to visualise program execution, and build mental models of algorithms (c.f. Section 3.3).

3.7.4.1 Squeak and Scratch

Squeak EToys is a learning environment that is based on a constructivist learning model (Ingalls, Kay, Kaehler & Wallace, 2007). It allows children to draw objects and then write scripts to control them (Kay, 2008). Squeak is based on Smalltalk (Ingalls, Kaehler, Maloney, Wallace & Kay, 1997) and code is created by using 'tiles' - which means that children do not need to type code (see Figure 3-7).

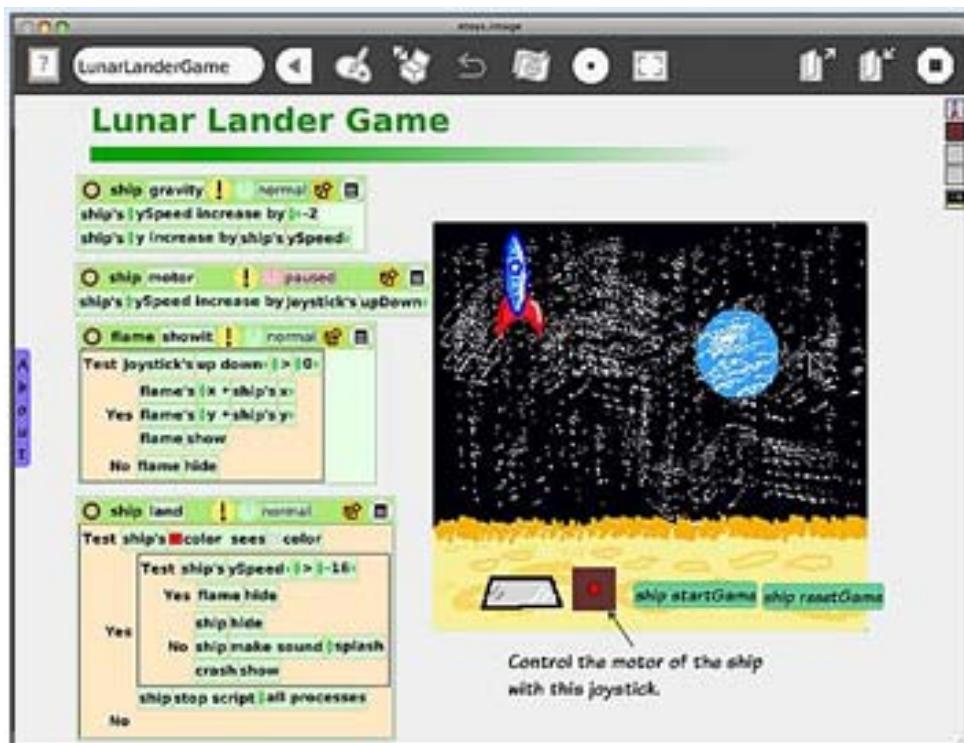


Figure 3-7 Squeak interface

Squeak has been successfully used in a university course to teach user interfaces (Guzdial, 2001), although this was not an introductory programming course.

Based on previous work with Logo and Squeak EToys, Scratch is an open-source environment written in Squeak (Maloney, Burd, Kafai, Rusk, Silverman & Resnick, 2004). It is intended to increase technological literacy in economically disadvantaged communities and was originally introduced at after-school centres in the USA (Resnick, Kafai & Maeda, 2003). Aimed at 8 - 16 year olds (Utting et al.,

2010), Scratch allows students to combine media to produce interactive creations (Lifelong Kindergarten Group, 2007a), by using 'snap-together' programming building blocks (see Figure 3-8 below). These blocks exclude syntax errors by only fitting together in ways that make syntactic sense (Lifelong Kindergarten Group, 2007b).



Figure 3-8 Scratch building blocks

Each script can be run separately or together and several scripts can be visible in the interface at one time (see Figure 3-9).

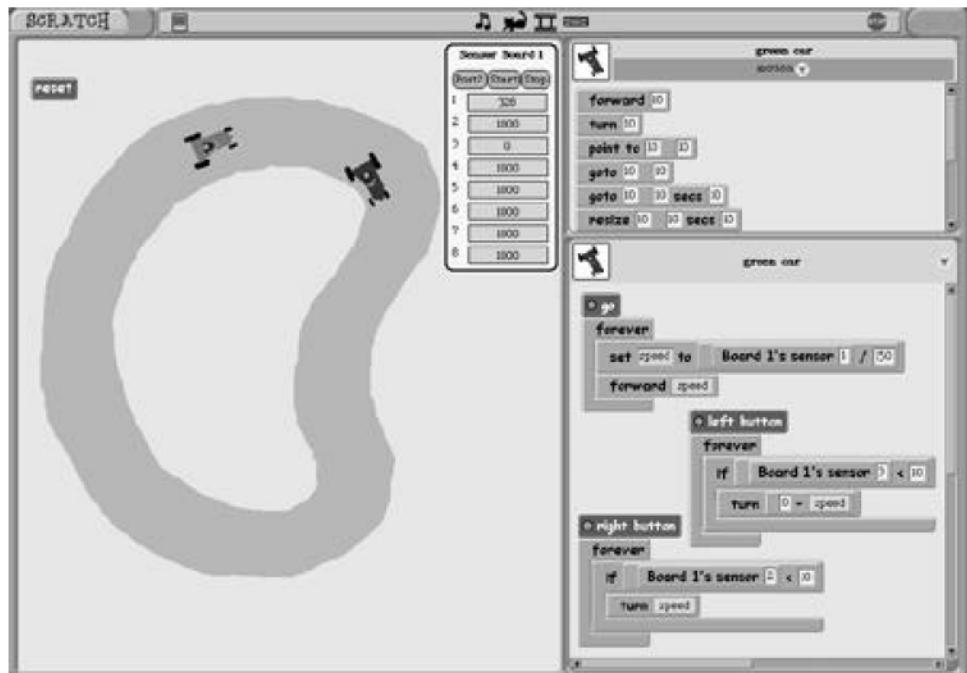


Figure 3-9 The Scratch interface showing a car game

Scratch does not support defining classes, inheritance, exception handling, procedures/functions, parameter passing and return values or some other concepts

such as recursion, but does teach the basic programming concepts of sequence, iteration, conditionals, variables, arrays, event handling and more (MIT, 2010). The Scratch interface allows the results of the student's programming to be immediately available within the environment, and projects may also be shared via the internet (MIT, 2012b). At time of writing, over 2,800,000 projects had been uploaded internationally.

3.7.4.2 App Inventor

A more recent programming environment that uses programming blocks for creating mobile Android applications is Google's App Inventor (MIT, 2012c). This new environment has been successfully used to teach programming in an introductory programming university course (Wolber, 2011). App Inventor uses the same code library as Scratch for its code blocks (see Figure 3-10 below), and also has a simplified interface with two main windows.



Figure 3-10 App Inventor Code Blocks

The App Inventor Designer allows users to visually design how the Android mobile app will appear, then programming blocks are used to build the program using the App Inventor Blocks Editor (MIT, 2012c). App Inventor has a web-based, drag and drop interface and uses an event-driven programming framework (Saha, 2012)

3.7.4.3 Alice

Another very popular environment that also uses code blocks to avoid the need for students to learn syntax, is Alice (Carnegie Mellon University, 2006). Alice was initially created to inspire middle-school girls to be more motivated towards programming through the creation of 3D animated movies (Kelleher and Pausch, 2007).

Alice displays the program state by showing objects and their interactions in a 3D virtual world, and animating state changes (see Figure 3-11).

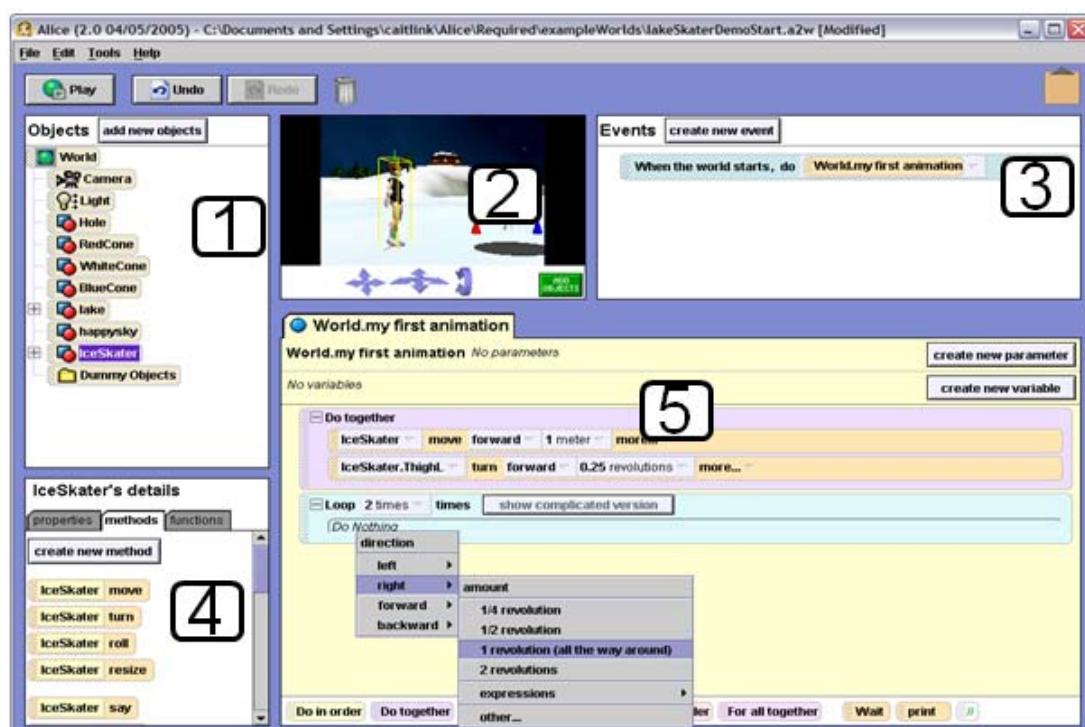


Figure 3-11 Alice interface

The Alice interface includes an object tree (1) which lists all the objects in the current Alice world, the scene editor (2) allowing placing of objects in the 3D world, the event window (3) allowing association of methods with events such as mouse clicks, details area (4) which shows methods, functions and properties for selected

objects, and the programming area (5) where programs are built by dragging programming blocks from the details area (Dann, Cooper & Slater, 2006).

Alice reduces the complexity of learning programming by removing the necessity of remembering syntax (Kelleher, Cosgrove, Culyba, Forlines, Pratt & Pausch, 2002), and by helping students gain mental models of parameter passing concepts and objects (Cooper, 2010).

Many studies have been performed on the efficacy of using Alice in introductory courses. In universities, the following studies have shown that using Alice as an introduction to a more standard Java programming course is beneficial for students. Grant, Hsaio & Turich (2010) found that using Alice as a 3 week introduction to a Java course resulted in the withdrawal rate reducing from 20% to 5% of students. Students who had had the Alice introduction also performed significantly better on a programming test than those who had not.

Moskal, Lurie & Cooper (2004) also found that students exposed to Alice had approximately one full letter grade improvement in their final score, compared to those who did not, and 'at risk' student retention into a second programming course increased from 47% to 88%. These results were replicated in another larger study of approximately 1000 students where those who had an Alice (in the first course) to Java (in the second course) programming sequence were compared with those who had a C++ to Java sequence. Those who experienced Alice in the first course (compared to C++) had a 4% increase in women enrolled, a 4% increase in pass-rate and 4% decrease in withdrawals from the course. More tellingly, in the subsequent course in Java, those who had encountered Alice in their first course had a 5% increase in pass-rate and 11% decrease in withdrawal rates in the second course (Mullins, Whitfield & Conlon, 2009). A comparison of confidence in programming in

an online programming course between students who were exposed to Alice then Java, compared to Java alone showed that the students experiencing Alice had higher confidence overall and were more likely to continue programming studies (Daly, 2011).

Other positive experiences have been reported using Alice in various settings with students of all levels of ability (Bishop-Clark, Courte, Evans & Howard, 2007; Harriger, 2009) - for example, for game programming in a first programming course (Seidman, 2009). Alice has been used successfully in mathematics, science, language, literature, social studies, history and art classes as well as technology and programming studies in middle school (Peluso and Mauch, 2009; Rodger, Slater, Hayes, Lezin, Qin, Nelson, Tucker, Lopez, Cooper & Dann, 2009). The use of Alice is reported to increase motivation, self-efficacy and have advantages in teaching abstract concepts such as 'objects' and parameters (Brown, 2008).

Most studies find that using Alice followed by Java is better than using Java alone (Dann, Cosgrove, Slater, Culyba & Cooper, 2012), although there have been some detractors. One study (Garlick and Cankaya, 2010) found that students who were taught basic programming concepts using pseudocode for two weeks followed by Java performed better on a programming test than those students who were taught using Alice for two weeks followed by Java. The test involved a 'make correct change' program similar to those required in traditional Java courses, which may have advantaged the students in the pseudocode group. The same study reported that students also had difficulty transitioning from Alice to Java. However this study has serious flaws, in that it does not report how students were recruited, how they were placed into groups, and evidence of the quality of the test.

Other researchers have remarked that Alice needs to implement inheritance more fully - something that is being addressed in Alice 3 (Cooper, 2010; Dann, Cosgrove, Slater, Culyba & Cooper, 2012). A weakness of Alice is that all abstractions are visual, which means that abstractions that do not interact physically as objects in the 3D world are more difficult to understand (Brown, 2008).

3.7.4.4 Mindstorms

A programming environment that (at least initially) removes the need to see syntax at all, is the Lego Mindstorms NXT programming environment (The LEGO Group, 2009). Programs are built to control the display, sounds, motors and sensors of physical Lego-based robots. The Mindstorms kits (including a control processing 'brick', motors, sensors and various wheels, Lego building blocks and connectors) and associated programming environment have been used in introductory programming courses, and also in advanced courses in operating systems, compilers, networks and artificial intelligence (Klassner and Anderson, 2003; Klassner, 2002). The Mindstorms environment comes with its own block-based programming language (see Figure 3-12), however Mindstorms robots can also be programmed in various other programming languages including Java (Barnes, 2002).

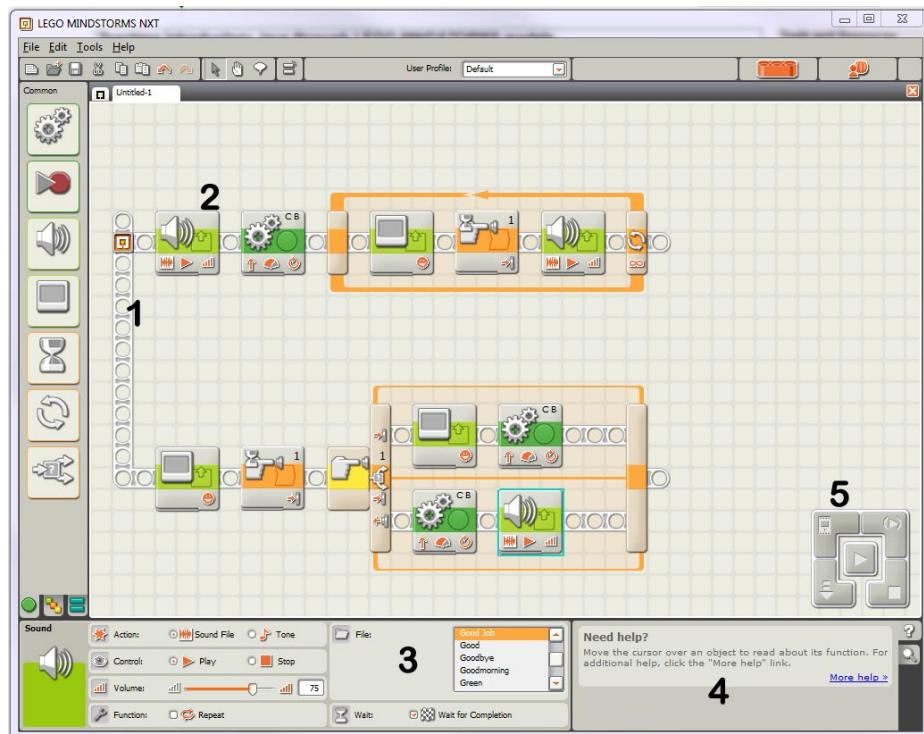


Figure 3-12 Mindstorms interface

The Mindstorms language consists of programming blocks that are dragged and dropped onto a "sequence beam" (timeline - indicated by **1** in Figure 3-12 above) which controls the flow of execution of the program. The properties of each programming block (**2**) are set by an inspector (**3**) in the main window of the program. Help (**4**) is always available by mouse-over of various components, and programs are compiled and downloaded into the robots by using an on-screen control panel (**5**).

Results using Mindstorms to teach introductory programming have been mixed. Some studies have found increased motivation and satisfaction in students (Barnes, 2002), while others have found no significant differences in motivation, confidence and self-efficacy between students who were exposed to Mindstorms for three weeks and then continued to C++ compared to a control group that used C++ throughout the course. (McGill, 2012; McWhorter and O'Connor, 2009).

Mindstorms have been used as part of curriculum change to encourage engagement, along with reduction of the set of Java learned in an introductory course

and removal of the final exam (Ford and Venema, 2010). The result of these changes was a large reduction in the fail-rate for the course from 30-40% to 18.2%, and reported increase in satisfaction. However, when 111 students who had passed this course were tested on variable assignment in a subsequent course, two-thirds of the students in this course scored 0 out of 4.

3.7.5 Which learning environment?

Which of these environments (or others) should be chosen for introductory programming at university level? All have strengths and weaknesses. One study that compared teaching introductory programming using two teaching environments - BlueJ and Programming Microworld of objectKarel - showed that students had strengths in differing areas based on the environment which they had experienced. Combinations of more than one environment were recommended to give the best results (Xinogalos, Satratzemi & Dagdilelis, 2007). Some of the environments above have common features that may impact on the students' focus of attention, cognitive load and motivation to understand the environment and language while forming mental models of basic algorithms and problem-solving. These include:

- reduction of the number of windows used - many learning environments use only one or two windows - with all relevant information available;
- programming 'blocks' or code skeletons that are used to remove the necessity of learning significant amounts of syntax;
- visual environments that are often drag-and-drop;
- connection with physical objects (in the case of Mindstorms) or physical representations of familiar objects to demonstrate abstract ideas.

Each of these features are broadly designed to reduce the number of things that a student needs to recall, reduce the number of things that a student needs to do, and

reduce the level of abstraction of concepts that a student needs to understand. As it happens, there are positive benefits to learning that flow from reducing the collective load of things that students need to recall, do and conceptualise.

This chapter has examined *content-centered* factors that impact on the success of students learning computer programming. The focus of discussion has been primarily upon the paradigm(s), the language(s) and the development environment(s). In the next chapter of this thesis, the *instructional design* of instructional materials and instructional activities for novices in complex, technical areas is explored, with particular focus on Cognitive Load Theory (Chandler and Sweller, 1991). Some of the above features are analysed in terms of this theory. Of primary interest is the question of *how* such paradigm(s), language(s) and development environment(s) may best be used to facilitate student success in learning introductory computer programming. These cognitive load based approaches to instructional design are explored in Chapter 4 “Instructional Design Factors”.

Chapter Four: Instructional Design Factors

4.1 Introduction

"Learning results from what the student does and thinks, and only from what the student does and thinks. The teacher can advance learning only by influencing what the student does to learn"

Herbert A Simon, quoted in (Ambrose, 2010 page 1)

4.2 How much instruction?

The heart of the debate about effective teaching revolves around how much instruction should be given to learners - should learners discover knowledge for themselves, or should they be told what they need to know? Koedinger & Aleven (2007) referred to this as the 'assistance dilemma'. Approaches exist along a continuum with "discovery learning" at one end - where learners are encouraged to discover knowledge for themselves - and "provision of direct instruction" at the other.

4.2.1 Constructivism

Discovery Learning is based in *constructivism* which derived from the works of Piaget (1973, 1971a) and Bruner (1961). Piaget (1973) argued that learners needed to construct their knowledge by discovery (freely exploring learning materials) otherwise they would be devoid of creativity and could only repeat what they were told. Bruner (1961) argued that discovery learning produced benefits for retention and transfer, as well as increased motivation, and this has been confirmed in several studies (Guthrie, 1967; Lepper, 1988; McDaniel and Schlager, 1990).

Constructivism, or discovery learning, also has disadvantages in that learners may not remember how they solved a problem, if they struggled for long enough while solving it (Lewis and Anderson, 1985). In addition, if learners struggle while

attempting to find the solution to a problem by discovery, they increase the time taken, and may never actually discover the important principles they were meant to learn (Ausubel, 1968).

4.2.2 More instruction - problem solving and direct instruction

Also on the continuum of approaches between discovery learning and direct instruction is "problem solving". Problem solving, as mentioned in Chapter 3 of this thesis, involves some form of guidance, followed by learners solving a series of problems. Gutherie (1967) compared the performance of learners given explicit rules followed by problem practice ("problem solving") with performance by learners who had to discover the rules for themselves. Although problem-solving learners performed better on recall-type problems, discovery learners performed better on transfer problems requiring their newly learnt rules to be applied in new contexts.

Both discovery and problem-solving approaches require extended time and practice to receive the benefits of these approaches (Brunstein, Betts & Anderson, 2009). In the study by Brunstein et al., 50% of learners using discovery and given less practice than an earlier study (which favoured discovery learning) felt lost and wanted to quit the study, while the remainder performed more poorly than those learners who had been given direct instruction.

A comparison between three experimental conditions - tutorials (direct instruction), problem solving and learner exploration - was performed by Charney, Reder & Kusbit (1990). The tutorial condition received the highest, and the discovery (learner exploration) condition the least, level of instruction respectively, with problem-solving in between. Performance results indicated that the problem-solving condition was the most effective. However, this study was criticised by Tuovinen and

Sweller (1999) because time-on-task was not controlled, and so presented as a confounding variable.

There are many studies showing superiority of direct instruction in different domains such as mathematics (Cooper and Sweller, 1987), science (Klahr and Nigram, 2004; Matlen and Klahr, 2010), procedure learning (Rittle-Johnson, Siegler & Alibali, 2001) and even LOGO programming (Fay and Mayer, 1994; Lee and Thompson, 1997). Klahr and Nigram (2004) and Strand-Cary and Klahr (2008) found that direct instruction was superior to discovery learning in more complex domains, and that this held for “far” transfer as well, where learnt concepts and procedures were applied to novel projects by children in a science fair.

The bottom line appears to be that the level of direct instruction needed depends on the nature of the instruction, the prior knowledge of the learner, and how much practice time is available. A theory that provides explanation for these effects, in terms of human cognitive architecture, is Cognitive Load Theory.

4.3 Cognitive Load Theory

Cognitive Load Theory (CLT) is an instructional theory based on knowledge of human cognitive architecture and in particular its limitations (Sweller, 1999). There are many instructional design consequences that result from the relations between parts of the human memory system and the way in which information is processed through these. The following section outlines human cognitive architecture, its limitations and the nature of expertise.

4.3.1 Human Cognitive Architecture

Human Cognitive Architecture is often described as an information processing model with operational relations between the separate conceptual functions of Long

Term Memory and Working Memory (Sweller, 1999). There are added complexities to the model with the inclusion of Sensory Memory to handle, very briefly, the input from our senses, and modality specific slave-systems for management and integration of distinct visual and auditory information components within Working Memory (Mayer, 2005). Figure 4-1 shows the basic relations between sensory memory, working memory and long term memory.

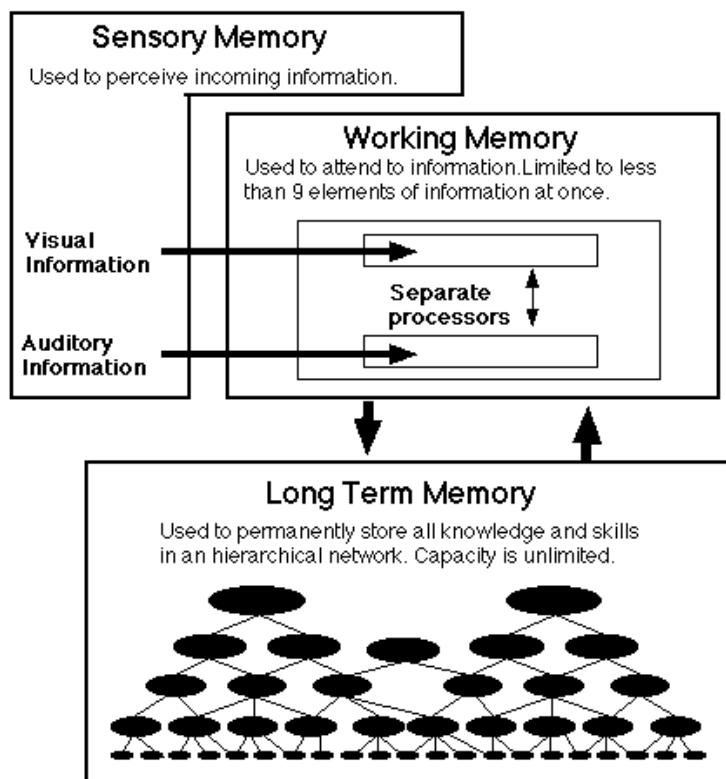


Figure 4-1 Cognitive Architecture - from (Cooper, 1998)

4.3.1.1 Long Term Memory

Long term memory is a virtually permanent store of knowledge. It is effectively unlimited in its capacity, with knowledge stored in the form of “schemas” which are hierarchically structured information networks that gather and interrelate throughout a person’s lifetime (Sweller 1999). Evidence for this comes from De Groot (1965), who showed that given a sighting of a chess board configuration for a few seconds, chess experts (“grand masters”) could reproduce the position of most of

the pieces, something that could not be accomplished by novice chess players. A replication of this experiment by Chase & Simon (1973) found that this was not merely a better recall of positions of pieces per se, as only 'real' game configurations could be more accurately reproduced in this way by the masters. When shown random configurations of pieces, the masters performed no better than novices. Unlikely as it seemed, the chess grand masters had stored board configurations in long-term memory, which allowed them to recognise familiar configurations of pieces. Simon and Gilmartin (1973) estimated that up to 100000 board configurations were memorised by chess grand masters.

Information in long term memory is held in the form of *schemas* which are discussed more in Section 4.3.1.3. Before that discussion, however, some comments are required about Working Memory.

4.3.1.2 Working Memory

Working Memory is dedicated to the conscious processing of information. It is where activities such as thinking and problem solving take place. Working memory is strictly bounded in its capacity with 7 elements (+/-2) representing typical performance by adults on random memory tasks (Miller, 1956). Working memory also has limited duration, with all contents of working memory lost within about 20 seconds (Peterson and Peterson, 1959). Humans are able to work-around their limitations of working memory through selecting 'larger' size elements to process (Miller, 1956), by grouping together items or 'chunking'. For example, the phone number 02-6659-3190 is easier to remember than the sequence of digits 0 2 6 6 5 9 3 1 9 0.

The number of elements that can be handled by working memory are further reduced when considering elements that are not random, but hold well-defined relations. For example, the formula “ $F=ma$ ” does not only require holding of the concepts of force, mass and acceleration, which *some* people will recognise as being represented by the letters “F”, “m” and “a”, but also their relation as defined by the formula. To overcome the limitations of the capacity of working memory, humans use organised elements of knowledge, which are held in long term memory as schemas, within working memory tasks (Sweller, 1999). People who have sufficiently well-developed schemas in physics would identify, store, and process “ $F=ma$ ” as “Newton’s Second Law of Motion”.

4.3.1.3 Schemas

Schemas are well organised hierarchical networks of conceptual and procedural information. As expertise in a content domain expands, organises and intercepts, so too do schemas become larger, more well organised and more interrelated (Chi et al., 1982). A "novice" in a content domain only holds small disconnected schemas, and so needs to hold a larger number of elements in working memory, than does an "expert" in the area, when consciously considering the same body of content.

An example from programming (Figure 4-2 below) demonstrates the difference between an expert's and novice's schemas.

```

1 class ArrayDemo {
2     public static void main(String[] args) {
3         // declare an array of integers
4         int[] anArray;
5
6         // allocates memory for 4 integers
7         anArray = new int[4];
8
9         // initialise elements
10        anArray[0] = 100;
11        anArray[1] = 200;
12        anArray[2] = 300;
13        anArray[3] = 400;
14
15        System.println("Element at index 0: " + anArray[0]);
16        System.println("Element at index 1: " + anArray[1]);
17        System.println("Element at index 2: " + anArray[2]);
18        System.println("Element at index 3: " + anArray[3]);
19    }
20 }
```

Figure 4-2 Java code for an array

This Java example deliberately does not use any loop construct to initialise the array, so that only the array (initialising and retrieval of values) may be considered in our example. An expert (even without the helpful comments) will recognise the code for initialising the array, and from their long term memory and well-developed schemas of encountering, creating, and manipulating arrays will understand the first 13 lines of code as resulting in "an array of integers that looks like [100, 200, 300, 400]". Their schema for arrays already includes other information about arrays - that they are represented in square brackets in Java ("[]"), that array indexes start with 0, and much more related information about arrays, including the fact that this code could more easily (at least for the expert!) have been accomplished with a loop. The print statements (lines 15-18) then become a trivial exercise in retrieving parts of the array, which is already well understood.

In contrast, a novice may be stymied by the class declaration, or the static main method, and may be overloaded with information before even getting to the array. Even disregarding these, the novice will have to hold the parts of information about the array as discrete elements of information in their short-term memory:

- that it is an array of integers,
- that it has four elements,
- that array indexes begin with 0 in Java,
- that the first element is 100,
- that the second element (with index 1!) is 200,
- that the third element is 300; and
- that the fourth element is 400.

With less-developed schemas than an expert, a novice may not even notice the relations between the individual integers in the array, because their working memory is already holding the maximum number of possible cognitive elements.

If the number of integers in the array is increased to 20, say, then this will likely not change the cognitive load at all for an expert, who will be treating the array as one element. For a novice, however, expanding the array to 20 integers will impose additional complexity and thus additional load on cognitive resources.

Complex sets of information can be manipulated in working memory by more knowledgeable individuals because they hold more well developed (larger and better organised) schemas in their long term memory. As an individual's level of expertise in a specified content domain increases, there is another critical aspect of their schemas that also develops to further reduce their working memory load, and that is the *automation* of their schemas.

4.3.1.4 Automation

Automation is the second primary performance feature considered within human cognitive processing models. This refers to a person's ability to proceduralise

and apply their schemas while needing only very low levels of conscious attention (Shiffrin and Schneider, 1977). Once a schema has been created, further practice over a long period of time means that it can be processed automatically without the need for high levels of conscious control (Kotovsky, Hayes & Simon, 1985) - that is, without using working memory, or rather, with only minimal levels to “trigger” the execution.

As an example, consider the act of reading, which you are doing right now. Once recognition of the letters that form words has been mastered, we read without consciously considering the letters that make up the words. As our reading skills further develop (through acquisition and automation of schemas for words) we may develop the capacity to read without even consciously considering words that make up phrases. This “automating” of reading skills frees working memory to concentrate on the *meaning* of a written passage, rather than its components (Sweller, 2005).

4.3.1.5 Thresholds

As schemas develop, broadening their orbit of influence, and in-filling “gaps” that may exist between various nodes and branches, they may sometimes intersect with areas of influence from other related, though distinct schemas. This may result in a re-constituted mindset that provides the learner with a repositioned world view, or capacity to conceptualise ideas in a manner that was not previously possible for them. This newly acquired re-constituted schema will itself also become automated with practice or experience.

From the perspective of cognitive architecture, “understanding” can only occur if and when a learner has acquired sufficiently developed and refined schemas which enable him or her to hold in their consciousness, that is, in their working

memory, all of the necessary information associated with a particular concept simultaneously.

The re-constitution, or “reconfiguration” of several related schemas into a single more unified all-encompassing higher order schema may sometimes provide moments of relatively spontaneous insight which a learner may experience as “ah-ha phenomena”. This may explain why some challenging concepts that are domain specific are particularly difficult for learners, but when mastered provide a “threshold” to a whole new area of learning. Such shifts in the relative bedrock of a knowledge base may lead to a theoretical basis for accounting for particular change-points that may be experienced by learners, something akin to crossing of a “threshold concept” of learning (Meyer and Land, 2003). This may particularly occur in complex conceptual domains, because the reconfigured schema base provides a relatively sudden, but stable, change in the learners’ capacity to reflect on certain concepts that are discipline specific. These changes in capacity to utilise a new perspective on concepts may act as a “gateway” to higher levels of understanding and expertise in the specific discipline.

4.3.1.6 Expertise

Well-defined and large stores of information in connected schemas allow experts to recognise familiar situations and the actions that are required in those situations. Models of expertise typically distill the primary traits of expertise down to the possession of these relatively large well-structured networks of schemas, and the capacity to access and apply the knowledge and skills held in these schemas, with low levels of conscious attention. These two attributes are sufficient to explain virtually all aspects of expert performance, including transfer and problem solving (Cooper and

Sweller, 1987). The primary job of educators should be to help students build schemas and then automate them so that they are able to free their cognitive resources to attend to, and process, new material.

4.3.2 Cognitive Load

4.3.2.1 Definition

Cognitive Load is a “count” of the number of elements that must be held within working memory for any particular processing task (Sweller, 1999). Each element will be in the form of a schema (Chi, Glaser & Rees, 1982) as it exists for an individual based upon their current personalised knowledge base.

4.3.2.2 CLT and Learning

Cognitive Load Theory explores the application of information processing models to the processes of learning, and more importantly, to the design of instructional materials and activities, with the purpose of facilitating learning. Central to this approach of instructional design is the limitations of working memory. If the mental processing capacity of these limited resources is exceeded at any point during a learning transaction (“overloaded”), then learning will falter due to the “dropping” of information that was currently being attended to and processed.

There are many apparent similarities between this model of human cognitive architecture and its manner of information processing, with those associated with computer architectures and data processing. This is not accidental. There have long been identified relations between these two areas that have effectively evolved in parallel (Hunt, 1982). “Cognitive overload” is similar to a “buffer overflow”.

Overloading means that some data is dropped. Once dropped, it is lost - irretrievably - and so while some form of processing may still occur, the nature of the information now being handled is incomplete, or worse still, incorrect, and can result in an effective program crash, or memory access errors.

As argued in the previous section, the primary goal of instruction is to facilitate students acquisition of automated schemas, and hence expertise. For this reason there needs to be careful consideration and monitoring of the working memory consequences of the instruction. Cognitive Load Theory considers load on working memory in three dimensions - *intrinsic*, *extraneous* and *germane* cognitive load.

4.3.2.3 Intrinsic Cognitive Load

Intrinsic cognitive load refers to the innate difficulty of a body of to-be-learnt content (Chandler and Sweller, 1991). Some content, by its *intrinsic* nature, is considered to be “difficult”, and thus requires a high level of mental effort to comprehend and learn. This is primarily due to the inherent complexities, interrelations and subtleties of the information. Both mathematics and computer programming are such intrinsically difficult areas (Sweller & Chandler 1994).

Element Interactivity

Information that contains many internal relationships between knowledge elements (element interactivity) imposes a higher level of load on working memory than information that can be considered to be a series of simple, unrelated facts (Sweller & Chandler 1994). This is because it is not only the separate discrete elements that need to be consciously attended to, but also the relations between those

elements (Tindall-Ford, Chandler & Sweller, 1997). The example below illustrates element interactivity in the context of languages:

"A popular example to consider is the learning of a second language -- not a second programming language, but a second spoken language. Assume that you, the reader, are relatively expert at English, and learn Indonesian, French or Spanish. Each is written in a similar script to English and has a large level of overlap in phonemes.

In learning a second language (or a third, fourth and so on), there at least two distinct aspects that need to be learned. These are the vocabulary of the language and the syntax (or grammar) of the language. The vocabulary is relatively simple, being in general terms a one-to-one word-swap for "me", "you", "good", "bad", "morning", "evening" and so on. There are several hundred, perhaps thousand, such word-swaps required in order to be able to communicate meaningfully at a conversational level.

The second aspect, the syntax, is far more difficult to learn, because to construct a sentence such as "How are you this morning?" is generally not just a process of swapping words. All of the words may need to be considered simultaneously, along with their inter-relations. Note, too, that this is a simple, common sentence! If there are more subtle aspects being included such as personal relationship, respect associated with maturity (age of the persons involved), time of day being said, and the gender of the items (not the people, but the items such as table, chair, food) in question (of which there is no meaningful equivalent in English) then the translation can become much more difficult if there is a requirement to "be grammatically correct".

A question such as "Have you eaten dinner this evening?" needs much more than a simple word-swap process to be translated into many languages because the order of the words may need to be re-sequenced, there are likely issues of verb conjugation, there is a need to be mindful of the "respect-relationship" between the players (akin to using "sir", "Mr" or "mate" in the Australian context), and we have not even yet considered aspects of enunciation and emphasis.

The purpose of this discussion of language is to indicate that learning a second language contains a mix of intrinsically relatively “easy” content - the one-to one word swaps - plus the relatively “difficult” content of syntax, which is difficult due to the element interactivity between the separate words. On top of this there may be physical performance difficulties such as recognising and producing specific sounds (phonemes).” (Mason and Cooper, 2012 pp 189-190)

Learning a programming language is not the same as learning a second spoken language. However, some of the difficulties of doing so are similar. As discussed in Chapter 3, in the context of learning computer programming there are several distinct and interacting domains that must be mastered at the same time - syntax, structures, algorithms, the concept of the notional machine, and pragmatics such as developing and debugging (Du Boulay, 1986). This multi-domain aspect of programming is likely to be a primary contributor to its reputation as being difficult to learn.

Moreover, when speaking a foreign language, *even if* some of the pronunciations are poor, and the phrasing or verb endings are not technically correct, there is a high likelihood that the person receiving this mispronounced, poorly structured, grammatically incorrect message will *still* understand the broad intent of the message. In the realm of computer programing no such luxury of “lee-way” or “making-allowance” exists. An error as small and apparently insignificant as using a comma (,) instead of a full stop (.) is generally sufficient to cause a program to return an error that prevents execution. This pedantic, unfriendly response to code that is not perfectly correct may be a contributor to not just the difficulty of learning programming, but also to the “fear” that has been expressed by some learners of computer programming.

This section has described how element interactivity is a primary source for imposing high levels of intrinsic cognitive load. As identified earlier, there are other

sources of cognitive load that are crucially important to analysing the effectiveness of instructional materials. The next section addresses the role of cognitive load that is *extraneous* to acquiring automated schemas of the to-be-learnt content.

4.3.2.4 Extraneous Cognitive Load

Extraneous cognitive load refers to the load generated by the format of instructional materials and/or to the performance of learning activities. Some formats and activities hinder learning by loading the learner with unnecessary information processing (unnecessary to the task of acquiring schemas and automating them). Extraneous cognitive load is, in-principle, *reducible*, as it is open to being manipulated by the instructional designer. Extraneous cognitive load is *extraneous* to the conceptual understanding of content, yet needs to be processed by a learner's cognitive resources because it is the means by which information is presented to a learner. On this basis it may be the "materials", or "media", or "activities" involved in the teaching-learning transaction (Ayres & Sweller 2005).

For example, text based instructions specifying how to travel from one location to another are usually more difficult to understand than the use of visual maps, even though they are equivalent content sources. Maps have an obvious spatial relation to the two-dimensional landscape which they represent, while the textual form does not. Note, however, that for a person who has never seen or used a map before, this will not be the case, because such a person will lack the schemas associated with map use in general. This highlights a key factor that must be considered when analysing a user's (or learner's) cognitive load. It is dependent upon the individuals current set of schemas, and their level of automation.

Similarly to the use of maps for directional instructions, different programming languages and environments may present differing levels of visual representations of core programming concepts, such as loops (iterations). The choice of computer programming language and/or environment may have a direct impact upon the ease of comprehending, learning and applying underlying programming concepts. The language itself is extraneous to understanding these core programming concepts, yet it is the vehicle by which these are usually presented to a student. A student who does not hold schemas for core programming concepts and who falters on aspects of language or syntax, will likely not be able to distinguish between such core concepts versus aspects of the language, as the reason for why he or she has faltered.

4.3.2.5 Germane Cognitive Load

The third factor - *germane* cognitive load - that contributes to total cognitive load is from the conscious focus of attention that a learner brings to a task to deliberately “remember” and/or “understand” to-be learnt content (Paas and Van Merriënboer, 1994). This cognitive load is germane to the tasks of schema acquisition and automation. The activities involved in germane cognitive load generally consist of making mental comparisons and contrasts between already-existing schemas and newly presented information, along with some form of practice (Cooper, Tindall-Ford, Chandler & Sweller, 2001) to proceduralise the application of the developing schemas. The broad instructional strategy that benefits learning is to maximise the level of cognitive resources dedicated to germane activities, while reducing extraneous cognitive load (Paas et al., 2003a).

Germane activities may range from shallow processing tasks such as rote-learning to deep processing elaboration strategies. For example, consider a student studying chemistry. The elements of the periodic table may be rote-learnt as a simple sequence, or more deeply processed and linked to other facts that may already be held, or developing, such as atomic weight, valency and chemical reactions.

Similarly, programming may be viewed as a set of isolated base concepts to be rote-learnt, or analysed, abstracted and re-organised into a wide range of algorithms that are utilised within programming structures. In general, the deeper the level of processing, the more able will the learner be in recalling and applying the information due to the level of "understanding" that has taken place, rather than only memorisation.

In situations where both intrinsic and extraneous cognitive loads are high it is likely that these will effectively drain the capacity of working memory for germane load, and thus impede learning.

4.3.2.6 Cognitive Load and Mental Effort

Mental effort is a cognitive construct that has its derivation in Cognitive Load Theory (Sweller, 1999) and refers to the focus of attention and level of cognitive resources that are allocated to a task (Kirschner, 2002). "Mental effort" is a term considered to be more accessible to lay persons due to its association with "how hard one is thinking" rather than a more load-oriented association to "how many things one is thinking about". Mental effort has historically been used as a measure to evaluate cognitive load (Paas and Van Merriënboer, 1993).

The bottom line, from a Cognitive Load Theory perspective (Paas, Renkl & Sweller, 2004), is that these three demands of mental effort (intrinsic, extraneous and

germane) *compete* for cognitive resources. An issue arises because human cognitive resources are strictly limited (Miller, 1956) and so the combined demands of these three targets of mental effort cannot always be met. The reality is that students, in attempting to deal with the broad complexity of programming, in conjunction with a specific programming language and/or environment, may have insufficient cognitive resources left available to dedicate sufficient resources to the primary purpose of the teaching and learning transaction - that of actually engaging in germane tasks such as comprehending, learning and proceduralising the newly presented to-be-learnt content regarding core programming concepts.

It is important to note that these potential difficulties described in comprehending and learning may be greatly amplified for students at the lower end of the performance spectrum.

4.3.3 CLT Applications

Cognitive Load Theory posits that intrinsic cognitive load is difficult to manipulate because it is so bound-into the actual nature of the to-be-learnt content. Recall that extraneous cognitive load derives from the “quality” of the instructional materials. Cognitive Load Theory argues that extraneous cognitive load should be engineered to be kept within manageable limits, presenting materials, activities or problems that are not too taxing for the novice's existing schemas. Finally, the third foundation of Cognitive Load Theory is that the cognitive resources dedicated to germane load should be maximised. This is generally only achievable by first removing as much extraneous load as possible (Sweller, 2005). Cognitive Load Theory has generated several instructional principles based on human cognitive

architecture and the management of these three categories of cognitive load. Nine of these principles, which are relevant to this thesis, are discussed below. These are the:

- Segmentation effect
- Goal-free problems effect
- Worked example effect
- Faded Worked Examples effect
- Sub-goals effect
- Split-attention effect
- Modality effect
- Redundancy effect
- Expertise reversal effect

4.3.3.1 Segmentation Effect

As working memory is limited both in capacity and duration, and humans require time to process essential content, presenting information too quickly will result in a lack of opportunity to integrate the information with existing schemas, or to form new schemas (Mayer, 2001). This has given rise to the *segmentation principle*, which suggests that dividing to-be-learned content into segmented blocks will aid schema acquisition (Mayer and Moreno, 2003) by giving learners the time they need to carry out essential cognitive processing. Studies (Mayer and Chandler, 2001; Mayer et al., 2003, 2002) have shown that information presented in learner-paced segments results in significantly better test performance in both recall and transfer outcomes than information presented as a continuous unit.

4.3.3.2 Goal-free Problems Effect

Novices solve problems by noting the ‘end’ or goal of a problem, and then search for a ‘means’ or process/theorem to get to that goal (Sweller, 1999, 1988). If no such theorem is known, then the novice searches for a subgoal that may help them move closer to the goal. In practical terms this means iteratively seeking ways to reduce the difference between the ever-changing current state and the goal-state. This problem-solving strategy is called ‘means-ends analysis’ and is used when novices do not have schemas for the relevant problem (Sweller, 1988). Means-end analysis imposes high levels of cognitive load due to the strategy being driven by seeking to reduce differences between the current state and the goal state. The focus is on getting the answer, rather than *how* the answer was obtained.

One way to reduce cognitive load is to withhold the identification of a goal in a problem, which stops the novice working backwards from the goal. *Goal free problems* promote forward exploration through the problem space, by asking learners to generate rules by past procedures, moves or actions. In this way, schema acquisition is aided, as schema help movements forward towards a goal (Sweller and Levine, 1982).

Such explorations do work well (and are akin to discovery learning), but can only be applied in situations where the problem space is very, very limited, otherwise the learner will have too many choices and may get ‘lost’.

4.3.3.3 Worked Examples Effect

Recall from Chapter 3 that many introductory programming courses place a relatively high emphasis upon problem solving. A core, central activity within the learning materials, is to require students to be actively engaged in producing a

solution to a computer programming problem solving task, as part of their learning about core concepts of programming. Such a focus on problem solving may hinder learning, as inquiry-based instructional design places heavy load on working memory (Sweller, 2005).

Many studies have indicated that a relatively high weighting upon problem solving tasks is less effective at facilitating learning than an emphasis upon studying a set of similar worked examples (Cooper and Sweller, 1987; Moreno et al., 2006; Sweller and Cooper, 1985; Zhu and Simon, 1987). There is also evidence that for novices, a worked example approach is superior to tutored problem solving as well as a combination of problem solving and worked examples (McLaren and Isotani, 2011). The underlying theoretical analysis is based upon the (previously referenced) argument that novices use means-ends-analysis to solve problems, that this process imposes a relatively high level of cognitive load, directs attention to the answer, rather than the process of obtaining the answer, and consequently impedes learning (Sweller 1988).

Worked examples function, primarily, by preventing use of means-end analysis. Learners are encouraged to work forwards through a solution, as demonstrated by the worked examples, focussing attention on the patterns and rules applied to move from one problem state to the next.

When studying worked examples, learners are freed from the performance demands of problem solving (using means-ends analysis) and can use freed cognitive resources to focus on understanding the content, and building schemas.

As learners increase their expertise, worked examples become less effective, in part due to reductions in motivation and attention (see Section 4.3.3.9) and so *faded worked examples*, where examples are progressively left more incomplete, can be

used to scaffold increasing skills (Renkl, 2005) while keeping cognitive load manageable.

4.3.3.4 Faded Worked Examples Effect

Faded worked examples function by maintaining a level of *germane* load. As the learner builds schemas, the scaffolding is reducing, resulting in wider gaps that the learner must fill in.

A fading worked example approach has been found to be superior to well-supported or tutored problem-solving (Schwonke, Renkl, Krieg, Wittwer, Aleven and Salden, 2009), even in the domain of ill-defined problems, such as visual pattern recognition (Anastasiade, 2009).

The faded worked examples approach has been considered in the domain of introductory computer programming. Gray, Clair, James & Mead (2007) have suggested that fading worked examples would be useful for teaching individual constructs and discrete blocks of code. They argue that this approach does not address the larger problem of program design, however "a deeper understanding of design, semantics and testing of program parts will facilitate development of higher level program skills" (page 109).

Buck & Stucki (2000) suggest that introductory programming students should be given a more complete work, with a missing fragment, which is similar to the worked-examples approach. They suggest that this leads to students solving smaller problems first, while retaining context – an “inside-out” approach, however they do not base this on cognition theory. Note from Chapter 3 of this work that this approach is also recommended by Kölling (2010; 2003) in the process of learning programming using the BlueJ and Greenfoot environments.

4.3.3.5 Sub-goals Effect

Sub-goals have similarities to faded worked examples in that they are designed to assist a learner to step their way through a problem space to a solution (Catrambone, 1998). Initially, many sub-goals are used, so that each step is small, and progressively the number of sub-goals is decreased so that the “step” between each goal increases.

Progressively reducing the number of sub-goals has similarities to fading worked examples in that initially the level of support is high, and then decreases over time as the learner builds schemas. Sub-goals have been used successfully to improve performance and transfer in novices learning to program mobile applications, using the AppInventor environment (see Chapter 3) and multimedia instructional materials (Margulieux, Guzdial & Catrambone, 2012).

4.3.3.6 Split Attention Effect

Recall from the discussion in Chapter 3 about simplified learning environments, such as App Inventor, that these environments often have common features such as a reduction of the number of windows used with all relevant information being made visually available - BlueJ has two main windows, as does Greenfoot. The implications of this environment design decision are discussed in terms of cognitive load in the context of the Split Attention Effect.

The Split Attention Effect (Chandler & Sweller 1992) argues that split formats of instruction, whereby two or more mutually referring sources of instruction are presented in isolation, and need to be mentally integrated to be understood, will be

less effective at assisting learning than an integrated format where all mutually referring sources of information have been combined into a single source.

For example, consider a split attention format such as was presented in Figure 3-11 (reproduced below in Figure 4-3) of the Alice programming environment interface. In Chapter 3, each of the components of the interface is described in a separate paragraph, which may reside on a separate page in this completed work. The instructional text is physically "split" from the location of the text-referenced diagram which sits above it.

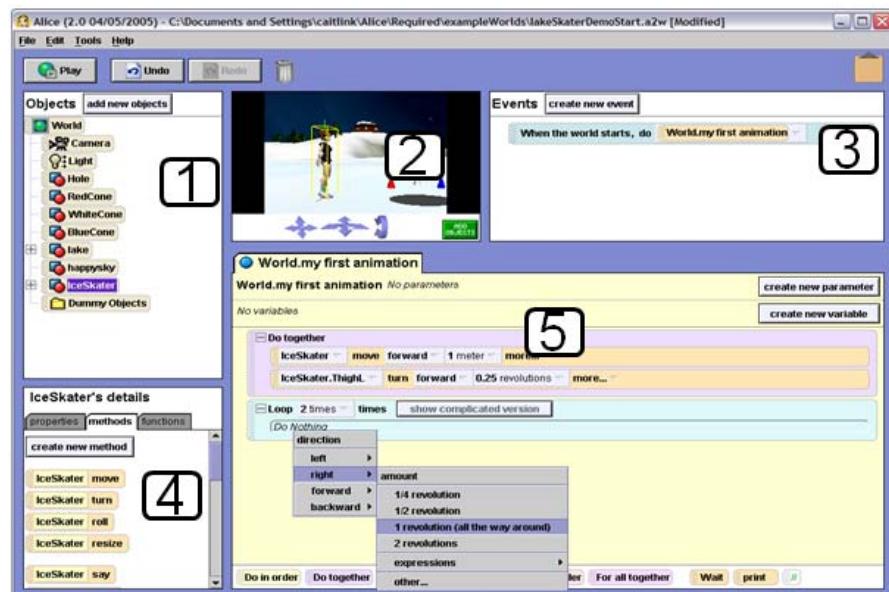


Figure 4-3 Alice Programming Interface with "split" text explanation

This imposes a level of visual search and requires a mental integration of meaning between the textual and graphical components of instruction to enable comprehension. This search and mental integration is not required for an equivalent-content instructional design as presented in Figure 4-4 where the textual information has been integrated (embedded) within the diagram.

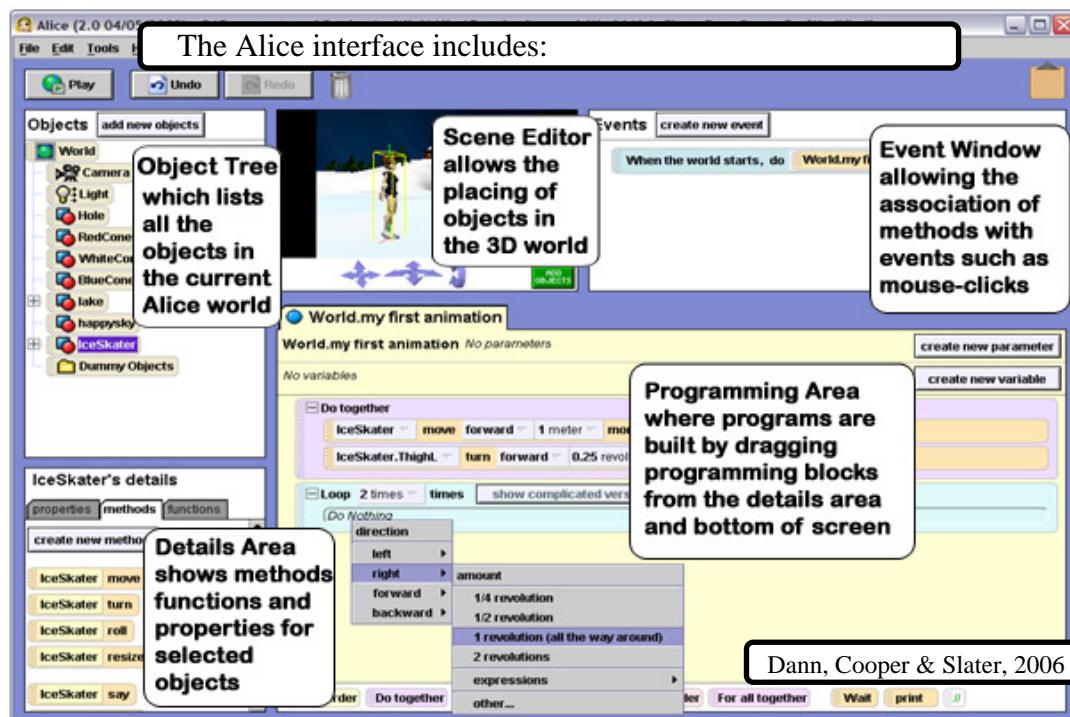


Figure 4-4 Alice screenshot with integrated text

The added extraneous cognitive load due to the visual search and mental integration of the two mutually referring sources of information in a split format causes additional taxing of cognitive resources, and so reduces the cognitive resources available to be applied to the germane activities associated with schema acquisition and automation. As a direct result of such an instructional design, learning is reduced. This effect has been demonstrated to be present in many instructional settings and is highly robust (Sweller 1999).

Related to the split-attention effect, but making use of the multi-sensory build of humans, is the *modality effect*, which concerns the *modes* of presenting two mutually-referring sources of information.

4.3.3.7 Modality Effect

As outlined in the previous section, the *split-attention effect* occurs when two sources of logically-related visual information (such as a diagram and associated

statements) are presented in isolation, and have to be mentally integrated to be understood. If one of these sources of required information is presented in auditory form, and the other in visual form, learning can be enhanced rather than impeded (Tindall-Ford, Chandler & Sweller, 1997).

The explanation for this phenomenon is found in the composition and organisation of working memory, which has separate processors for auditory information (the "phonological loop") and visual information (the "visual-spatial sketch pad") (Baddeley, 1992). When a task has two sources of information in the same mode (for example, both visual) that interact and must be integrated to learn from the content, the relevant working memory processor may become overloaded. Spreading the load between the two processors for visual and auditory modes effectively expands the working memory capacity available, and hence the resources available for germane activities, such as schema acquisition or automation.

Note that novice programming instruction often consists of attending to a written problem statement and attempting to write or understand code that is also presented in some form of textual environment or text editor, both physically discrete from the problem statement (a source of split attention) and both requiring processing by the visual-spatial sketch pad in working memory (highly loading the visual-spatial sketch-pad). These features of programming environments and activities are "intrinsically" likely to impose high levels of cognitive load, especially for novices in programming.

A key trait for effective integrated instruction and also effective dual-modality instruction is that the multiple sources of information which are being presented to the learner are *all* required to enable comprehension of the to-be-learnt content. It should be noted that this is not simply "repeating" what is essentially the same content via

multiple sources. Indeed, simply “repeating” information may actually impede learning, as described by the redundancy effect.

4.3.3.8 Redundancy Effect

Chandler and Sweller (1991) compared learning and instructional time needed to study three visual-only versions of an anatomy lesson on blood flow through a heart. The first version consisted of a diagram of a cross-section of the heart with labelled chambers and arrows illustrating the direction of blood flow within the heart. The other two versions added textual descriptions such as "blood from the upper and lower parts of the body flows into the right atrium", either embedded in the diagram (version 2, as an integrated instructional format) or as a series of statements placed under the diagram (version 3, as a split instructional format). Performance by participants in a post-treatment test was significantly higher and significantly faster by the *diagram-alone* treatment group, with the diagram and separate text (split instructional format) condition group performing least well of the three groups.

The redundancy effect has also been demonstrated in visual plus auditory instructional formats. Studies by Mayer, Heiser & Lonn (2001) and Leahy, Chandler & Sweller (2003) demonstrated that audio explanations accompanying visual instruction aid learning only when tasks are more complex, and are only superior for visual information that is not self-explanatory.

Textual information, even when embedded in a diagram (thus avoiding the split-attention effect) can hinder learning if it is redundant to the information already being presented. Based on cognitive load principles, if learners are attending to two sources of information that involve the same content, part of their cognitive resources must be used to ensure that the two sources remain coordinated, leading to a

redundancy effect. The second source of information is not needed, but some cognitive resources are allocated to this redundant information.

4.3.3.9 Expertise Reversal Effect

Some studies on the instructional design aspects of Cognitive Load Theory have been performed in the domain of computing. Darabi, Nelson & Palanki (2007) gave final year university students a complex problem which comprised troubleshooting a chemical processing plant's system malfunctions. One treatment group was given worked examples, and the other group was given problem-solving strategies. Performance by the problem-solving group was significantly superior to the performance of the group given worked examples, which appears to contradict the worked example effect in Section 4.3.3.3 above.

Recall, however, the chess piece configuration studies referenced in Section 4.3.1.1. As chess grand masters have well-developed schemas for real board positions, it is not surprising that they were able to reconstruct board configurations that they were shown for a brief time (De Groot, 1965). The second study referenced in the above section, by Chase & Simon (1973), showed that experts actually performed *more poorly* than novices at random board configuration reconstruction. Kalyuga, Ayres, Chandler & Sweller (2003) refer to this as the *expertise reversal effect*.

This effect can be explained by the schemas that the expert chess players hold in long term memory. When confronted with a chess board, grand masters subconsciously look for meaningful patterns that will determine their next action. Failure to find these meaningful patterns means that resources are diverted to resolving the conflict between the schema and the environment.

A colleague of the author, who teaches introductory programming to university students, confided that he found the learning environment Alice "difficult to program - the blocks slow it down, and it's confusing". This is consistent with the *expertise reversal effect* as one may anticipate that his possession of elaborate schemas for writing programming code (which he has been doing continuously for 30 years) would reverse the helpful learning effects that novices would experience in the Alice environment, which is designed to eliminate the need to learn (and use) basic syntax.

The expert reversal effect has been demonstrated across many of the cognitive load effects. That is, worked examples are superior to problem solving, but only for novices (Kalyuga, Chandler, Tuovinen & Sweller, 2001); integrated formats are superior to split-formats, but only for novices (Mayer and Gallini, 1990).

The instructional designs generated by Cognitive Load Theory for novices, that are useful for novices, operate by guiding novices' cognitive processes through a supported process that makes specific allowances for a novices' lack of schemas. Experts, however, do hold such schemas. Moreover, an expert's schemas are automated. Cognitive Load Theory instructional effects designed at nursing a novice's low-schema base cognitive processes effectively disrupt the high-schema base cognitive processes that are held by an expert.

There are also germane cognitive load strategies, such as the *imagination effect* (Cooper, Tindall-Ford, Chandler & Sweller, 2001) that display a reversal effect. In this case, however, where a prompt is given to learners to use their imagination to mentally rehearse a procedure, effectiveness is only obtained for experts, and not novices, because novices do not yet hold the schemas required to enable accurate mental rehearsal of procedures (Cooper et al., 2001).

4.3.4 1Application to complex content

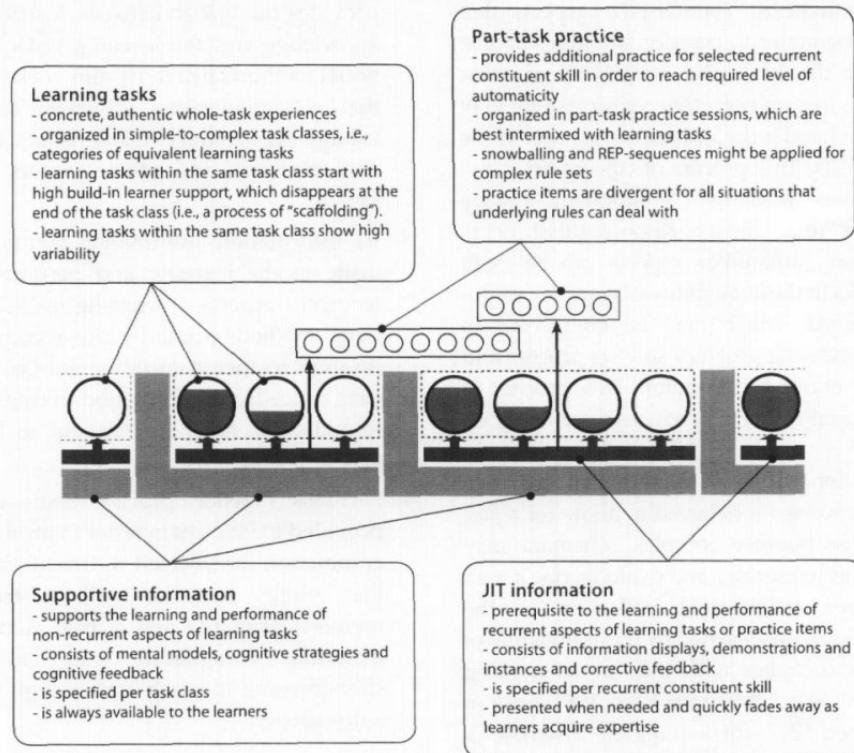
Given these instructional design principles, based on human cognitive architecture, how should courses that include complex, interrelated content (such as programming) be designed? Tuovinen (2000) recommends a set of instructional choices, based on cognitive load theory, in particular goal-free problem solving, worked examples, split attention, and redundancy effects. It is argued that a synthesis of instructional cognition research implications would be beneficial to novice programming.

Irrespective of whether an instructors' philosophy is based upon Constructivism or Direct Instruction, all of the above are seeking to reduce the total cognitive load placed upon a learner to an acceptable level – that is, reducing load to where a learner can both solve and learn from the activities.

One comprehensive model suggested by Van Merriënboer, Clark & Croock (2002) involves a combination of worked examples, segmented information, part-task practice and other components including problem solving to teach complex skills. This "*4C/ID model*" (four component instructional design) aims to focus on the integration and performance of task-specific skills, uses a combination of supportive instructional information and just-in-time information and recommends a mixture of part-task and whole-task practice (Van Merriënboer and Kester, 2005).

The 4C/ID model, as its name implies, involves four components: learning tasks, supportive information, just-in-time information, and part-task practice (see Figure 4-5 below).

Figure 2 □ A graphical view on the four components: (a) learning tasks, (b) supportive information, (c) just-in-time (JIT) information, and (d) part-task practice.



**Figure 4-5 Four components of 4C/ID model
(from Van Merriënboer and Kester, 2005)**

Learning tasks, as shown on the above diagram, consist of fading worked examples, with initially high learner-support and progressively higher learner input, for each "task class".

Supportive information is always available to learners, and consists of mental models, and feedback.

Just-in-time information (to avoid cognitive overload by too much information at the beginning of a task class) is presented when needed, and removed as learners gain expertise - to avoid the expertise reversal effect (Van Merriënboer, Kester & Paas, 2006).

Part-task practice – which is related to fading worked examples - supports the automation of skills. Making sure that practice items diverge as much as possible

within the rules of the task set allows learners to elaborate schemas as well as automate them (Van Merriënboer and Sweller, 2005).

4.4 Application to programming

Educators agree that to be successful in novice programming, it is important to form multiple mental models, deep and interlinked knowledge hierarchies and abstract, generalised problem-solving patterns (Winslow, 1996). According to Mead, Gray, Hamer, James, St Clair, Sorva & Thomas (2006) there has been much research from cognitive scientists, learning theorists and computer scientists about learning to program, however much has not transferred to texts or curriculum. Gray, Clair, James & Mead (2007) suggested a graduated exposure to programming concepts in a first course, primarily using fading worked examples, but lamented that the results of several research projects identifying cognitive based approaches associated with skill acquisition had not, as a whole, made their way into introductory or intermediate programming courses. Garner (2002) describes a software tool called CORT which uses fading worked examples to help novices program, and which is explicitly based on Cognitive Load Theory.

One paper that did describe a whole introductory programming course based on cognitive load theory and in particular worked examples (Caspersen and Bennedsen, 2007) explained how this design resulted in a pattern-based approach to programming and schema acquisition (see Chapter 3 for a discussion on pattern-based approaches). Patterns explicitly describe expert schemas in order to help novices build mental models of difficult concepts.

Segmenting content, using worked examples, designing instructional material to remove split attention and redundancy as much as possible, and taking advantage of

the modality effect to create better instructional content, are all processes that should reduce extraneous cognitive load and thus free resources for germane activities to assist in schema formation and automation in novice programmers. Consider, however, that the interface for developing programs, either for a novice or an expert, is the computer screen with some form of software (with or without a development environment). This interface, including the choice of development environment, also has the potential to be a source for extraneous load on novices, dependent on its design. Some of the features of introductory and learning programming environments outlined in Chapter 3, may - *without necessarily being designed for that reason* - reduce cognitive load and hence aid learning. In particular:

- *reduction of the number of windows used* - many learning environments use only one or two windows - with all relevant information available;
- *programming 'blocks'* or *code skeletons* that are used to remove the necessity of learning significant amounts of syntax;
- *visual environments* that are often drag-and-drop, that utilise recognition of information rather than the need to recall information;
- *connection with physical objects or physical representations of familiar objects* to demonstrate abstract ideas.

Reduction of the number of windows used may help reduce cognitive load caused by split attention, *programming 'blocks'* and *code skeletons* decrease the amount of to-be-learned content (segmentation principle) to manageable levels, *visual environments* present information that is processed more easily by the visual-spatial sketch pad, and *connection with physical objects or physical representations of familiar objects* takes advantage of already existing schemas to build new schemas of sometimes abstract ideas.

Another component of programming environments that has not been considered so far is the number of elements (buttons, icons, and so on) with which learners need to contend, in each environment. If a professional programming environment is used, it may have advanced features and menus that are not necessary for introductory novice programming. Even if the buttons and icons are not used, this may still form a source of cognitive load, as visual objects can capture attention even when they are extraneous to the goals of the observer (Theeuwes, Kramer, Hahn & Irwin, 1998). In the same way as redundant information increases mental effort because the extra information must be attended to and then ignored, the additional elements in a programming environment also may form a source of tacit distraction and hence cognitive load.

4.5 Studies in novice programming

Chapter 5 describes the first study of several studies in this thesis. Cognitive load instructional design principles were used in conjunction with the use of two learning programming environments to build introductory programming workshops as part of an "IT Girls" program that aimed to increase self-efficacy in programming in female high school students.

Chapter Five: IT for Self>Selecting Girls

5.1 Introduction

This chapter describes work undertaken by the author as the coordinator of a "Women in Technology" (WIT) program. The program aims to encourage and support female students studying information technology units. It has several facets, one of which is an outreach program to high school aged students. A component of the outreach program is an invitation to female students to visit a university campus to participate in an "IT Girls Day" which involves teaching and learning activities for introductory computer programming. Data from two of these IT Girls Days forms the first study of this work.

5.2 Background

5.2.1 Motivation

At Southern Cross University (SCU) female IT students have averaged around 19% of enrolments, in contrast to the higher percentage of female students at SCU as a whole (54%) (Southern Cross University, 2011). These low numbers of female students in IT and the potential to increase student numbers were the main drivers in support of the WIT program at SCU. The WIT program supports female students studying IT units and promotes IT programs to potential female students via social events, offering scholarships and awards, disseminating information on possible careers in IT, and promoting SCU as a women-friendly place to study IT.

As part of the general promotion efforts, the author and other staff have been visiting local schools to speak to IT students in Years 11-12 about education pathways since 2008. As very few girls attended these sessions it was decided to target younger

students, before they had chosen subjects for their final years of school, and run a combination of school visits followed by female-only workshops on the Coffs Harbour SCU campus. The targeting of this age group is supported by research by Graham and Latulipe (2003) who suggest that female students need to be introduced to computer science before cultural stereotypes have been absorbed, i.e. years 9 and 10 (14-16 years of age). Barker, Snow, Weston & Garvin-Doxas (2006) found that younger girls enjoyed such workshops more than older girls, possibly because of less exposure to such events and because they had not yet decided on possible career choices.

It was determined that the materials and activities delivered in the IT Girl Day workshops should have the objective of facilitating learning of core programming concepts. The intention was to demonstrate through well designed instructional materials and activities that:

- computer programming was not as difficult to learn as many people think it is (Jenkins, 2002), and
- attendees were able to successfully write computer programs.

The workshops were also aimed at presenting computer programming in a positive social context with female instructors used as positive role models. The design of these materials is detailed in Section 5.4.3.

It was hypothesised that using instructional materials designed to manage cognitive load in the manner described in Section 5.4.3 would enable students to learn the basics of programming, enjoy programming activities, increase self-efficacy and decrease apprehension of programming. The IT-Girls Day activities gave the author the opportunity to apply these cognitively based instructional design principles to

small workshops with programming novices and receive feedback on how they were received. It was not anticipated that such instructional materials would make programming “easy” per se, but *accessible* and *achievable* for these programming novices.

5.2.2 *The WIT Outreach Program*

A Higher Education Equity Support Project (HEESP) grant (designed to support projects that promoted equity) allowed the author to develop a program to present to younger girls the proposition that IT is a career worth considering. This program had two parts:

- a career information session delivered to Year 9 or 10 students in groups of up to 50 students at a time (male and female), followed by an invitation to female students to the second part of the program; and
- an "IT Girls Day" for girls only, at the Coffs Harbour university campus, where students could participate in hands-on programming activities, meet staff and students and ask questions in an informal environment.

Four local schools met the HEESP grant requirements (low socio-economic profile or rural) and career information sessions were scheduled at each of them. The process to arrange the school visits was revised as the organisers developed appropriate mechanisms to deal with individual school requirements and the need to work with relevant teachers in the promotion of the program.

5.2.3 *School Visits*

The first school (638 students, Years 7-12, 45% girls) visit was not a success due to an unsuitable location for the career presentation. The session was held on a

basketball court with a basketball game in progress on a neighbouring court - with resulting noise problems. All of the 150 Year 10 students - were seen at the same time, for a 10 minute session and none of the girls chose to take part in the next stages of the program. The manner in which this was organised may be indicative of the "low priority" given to the marketing of IT to girls.

At the second school (a smaller rural school with 262 students, Years K-12, 43% girls) the career visit (see Section 5.4.1) was successful and 10 girls were recruited for the first "IT Girls Day" on-campus workshop described in this chapter.

The third school (804 students, Years 7-12, 55% girls) were very enthusiastic during the initial careers class visits, but failed to attend the campus workshops on two separate occasions. This highlights a difficulty in arranging school visits. Any arrangements cannot be made with students directly but must be made through the relevant teachers. In this case, the school failed to arrange transportation for the students to the campus.

The careers visit to the fourth school (974 students, Years 7-12, 48% girls) was successful and this school accepted the invitation to an "IT Girls Day" and was the second on-campus workshop referred to in this chapter.

5.3 Hypotheses

The hypotheses that were tested in this study are listed below.

- ❖ **H_{A1} (Gender Bias):** Participants will perceive females to be more capable as computer programmers after the workshop than before the workshop.
- ❖ **H_{A2} (Programming Difficulty):** Participants will perceive computer programming to be less difficult after the workshop than before the workshop.

- ❖ **H_A3 (Self-efficacy):** Participants will perceive their self-efficacy in computer programming to be higher after the workshop than before the workshop.

5.4 Methodology

5.4.1 Careers Visits

For each school visit an all-female group of two or three persons visited the school to speak to all of the students (male and female) in a school year about the range of careers in IT and education pathways to those careers. This group comprised the author as WIT coordinator and representative of the university teaching staff, a current female IT student schools ambassador aged less than 25 years, and a mature female student who was studying at honours level who had experience in the IT industry. The schools ambassador answered any questions the students had about studying at university. All presenters talked about the wide range of careers available in IT, education pathways through TAFE and university, and their personal experience with working in or studying IT.

Lego Mindstorms NXT robots were taken to these careers information sessions to spark interest in the students, and to trigger conversations about the use of sensors, programming and other IT in everyday life.

The careers information talks typically involved speaking to two classes each time (around 50 students) for a school period (ranging from 40 to 60 minutes). Any interested female students were then invited to an on-campus "IT Girls Day" to be held approximately three weeks after the careers visit. It was decided to limit this initial day of workshops to a maximum of 12 participants. The small group size was more manageable, could be treated more informally than a larger group, and numbers were also limited by the number of Mindstorms robots that were available. Ten girls

from the second school visited accepted the offer to participate in the initial IT Girls Day. At the fourth school, ten students accepted the offer to participate in the second IT Girls day.

5.4.2 IT Girls Days

5.4.2.1 Overview

Both IT Girls Days began with a short welcome to the program by the coordinator, and introduction to the participating staff. The girls were asked to fill out a pre-workshop questionnaire. They were then divided into two equal groups with the first group completing a session on programming Alice (v2.2 - <http://www.alice.org/>) while the second group was introduced to programming Mindstorms NXT robots (see <http://mindstorms.lego.com/>). This programming session of approximately 1 hour 15 minutes was followed by morning tea and an informal chat session and then the two groups swapped activities for another 1 hour 15 minutes programming session.

At the conclusion of these sessions a post-workshop questionnaire was administered. The students were also provided with a sample bag that included the ‘Alice’ software they had used in the workshop. It was hoped that by providing this they may be encouraged to continue developing their programming skills after the workshops.

5.4.2.2 Questionnaires

The pre- and post- questionnaires had similar questions, with the same format and question sequence (see Appendix A.1 and A.2 for questionnaires on the first IT Girls Day, Appendix A.3 and A.4 for questionnaires on the second IT Girls Day). They were constructed with the intent of determining if there were any changes in the

girls' attitudes towards programming after a half day exposure to programming in Alice and programming the Mindstorms robots.

The first questionnaire asked for the girl's first name (for matching purposes), age, and then presented a series of Likert scale questions regarding the participants':

- computer knowledge/skill;
- programming experience;
- perceived difficulty of programming;
- confidence with programming;
- agreement that girls were better than boys at programming;
- level of interest towards programming with Alice;
- anticipated difficulty of programming with Alice;
- level of interest towards programming with robots, and
- anticipated difficulty of programming robots.

All Likert scales were presented in a horizontal format with low values on the left-hand side of the scale and high values on the right hand side of the scale. For the two questions regarding computer knowledge and experience this ranged from "not knowledgeable at all" and "none" for the extreme low values to "expert" and "huge amount" for the extreme high values. For each of the remaining questions a statement was provided regarding difficulty, confidence, interest and girls abilities being better than boys, and Likert scale responses ranged from "strongly disagree" for the extreme low value to "strongly agree" for the extreme high value.

The second questionnaire repeated the name and age questions so that the responses could be matched with the first questionnaire. It also repeated the questions regarding attitudes towards programming difficulty and confidence, and gender impact on programming ability, added two questions regarding their enjoyment of

Alice and Mindstorms, repeated the questions about their attitude towards Alice and robots and added a question about difficulty of use. The second questionnaire also asked Likert scale questions about the mental effort required for Alice and Mindstorms, with respect to understanding the tasks (“exercises”); navigating and using the software, and; learning from the activities. All Likert scale questions followed the same format as used in the first (pre) questionnaire, with low values on the left hand side of the scale and high values on the right hand side of the scale. The second questionnaire concluded with three open ended questions about the workshops regarding what was enjoyed most, what could have been done better and an invitation for “any other comments”. An additional question was included on the second day “What, if anything, about your activities today, affected your perceptions of how easy or difficult programming can be?”.

5.4.2.3 Workshops

Both the Alice and the Mindstorms programming workshops' instructional materials were specifically designed to be suitable for female novice programming students. The workshops were presented in small groups, with often more than one (female-only) instructor available to help students who requested assistance. The materials were presented in the form of a number of small publicly demonstrated worked examples in accordance with Cognitive Load Theory application to instructional design (see Section 5.4.3), coupled with 'free time' where students could create their own programs. The computer programming environments used are also designed to be user-friendly for these age-groups (Hundhausen et al., 2009), and, in the case of the Alice environment, motivating for girls (Kelleher et al., 2007).

5.4.3 Instructional Design

The instructional design of materials in the IT Girls Day workshops drew heavily from the principles of Cognitive Load Theory (Mayer and Moreno, 2003; Sweller and Cooper, 1985) which focuses upon the information processing constraints presented by human cognitive architecture. The instructional materials consisted of a series of short public presentations of worked examples defining specific programming activities to be constructed in each of the two simple programming environments, Alice and Lego Mindstorms NXT. In each case the public presentation of a worked example was followed by the students attempting to replicate the process just demonstrated, with assistance provided to any student who requested it. After the entire cohort had completed the task in question the next task would be demonstrated publicly as a worked example, with the students then attempting it. The cyclic process of public demonstration of a worked example followed by students' replication was continued until all the workshop tasks had been completed. In broad terms, this process of public demonstration began with the simplest of programming tasks and moved to those of increasing complexity. Further details of the tasks, task sequence, instructional processes and participant activities are provided below in 5.4.3.1 and 5.4.3.2.

5.4.3.1 Mindstorms Robots Workshop

In the Mindstorms workshop, students worked either alone or in pairs at one computer with one shared robot (of the three available robots), connected to the computer with a USB cable. The instructor used a large LCD monitor screen which was connected to a computer with the Mindstorms software installed to demonstrate instructional activities to the students. The students were introduced to the

Mindstorms robots in “humanoid form” (see Figure 5-2 below), shown how to handle the robot and identify the motors and sensors.

The Mindstorms kit consists of a central "brick" which has the processing hardware and software, a display screen and ports for sensors, as well as buttons to navigate the software and run/stop programs (see Figure 5-1).



- 1: "NXT" brick – the 'brain' of the robot
- 2: Touch sensor – enables the robot to feel
- 3: Sound sensor – enables the robot to react to noise
- 4: Light sensor – enables the robot to detect light and colour
- 5: Ultrasonic sensor – enables the robot to 'see', measure distance to an object, and react to movement
- 6: Servo motors – enables the robot to move.

Figure 5-1 Mindstorms brick, sensors and motors

These components, coupled with “technics”-type Lego pieces and gears, allow the user to build several forms of robots. For the workshops in this study the kits were assembled into humanoid form, both to be more easily recognisable for the students (see Figure 5-2 below) and to enable easy access to sensors and motors, which are less accessible on other forms of the Mindstorms robots.



Figure 5-2 Example of a Mindstorms Humanoid Robot

After familiarising themselves with the physical robots, students were introduced to the Mindstorms NXT software and led through a series of programming activities by the instructor. Each programming activity was designed to introduce the smallest amount of new information possible.

The instructor demonstrated and described a small worked example on the screen. The students then repeated the example using the software on the computers, downloaded it to the robots and ran their program. After each student (or pair of students) completed the activity successfully, the next worked example was demonstrated.

The approach used was designed to reduce extraneous cognitive load, and thus facilitate effective learning. Worked examples were used because worked examples have been proven to be a more effective teaching approach for *novices* in several technical domains than problem solving (Cooper and Sweller, 1987; Mwangi and Sweller, 1998; for example see Sweller and Cooper, 1985).

The instructor's presentation used verbal explanations at the same time as the programming process was demonstrated on the screen. Printed materials were not given to the participants, and instead only the verbal explanations were given, in

accordance with the modality principle (Mousavi et al., 1995). This states that students will learn better (as measured by both retention and transfer) from graphics combined with narration (Tindall-Ford et al., 1997) and simple animations combined with narration (Mayer and Moreno, 1998) (both techniques using visual and auditory channels) than from graphics combined with printed text (using only the visual channel).

The split attention principle posits that students will learn better when the words and pictures are physically and temporally integrated (Chandler and Sweller, 1991; Mayer and Anderson, 1991), rather than ‘split’ in either time or place. Therefore the explanations of each step were provided *at the same time* as the on-screen demonstrations.

In addition, although printed materials were not given to the girls, the instructor followed a predefined script (which is reproduced in Appendix A.5) designed using the segmentation principle (Mayer and Chandler, 2001), which advocates delivering content in small learner-paced segments of delivery, moving from simple examples to more complex ones. For example, the first example exercise involved using one program block only, to make the robot ‘speak’. Of necessity, this exercise also involved explanations of the compiling and downloading process. After the students were familiarised with this process using one block, and had performed the activity successfully themselves, the instructor moved to the next activity involving two programming blocks – necessitating explanations about the concept of sequence.

The Mindstorms NXT programming environment is highly visual (see Figure 5-3). Programs are created by dragging programming “blocks” to a timeline, and then setting properties of each block. The programs are executed in sequence along the

timeline. There can be up to three sub-timelines operating in one program, to allow for concurrent actions.

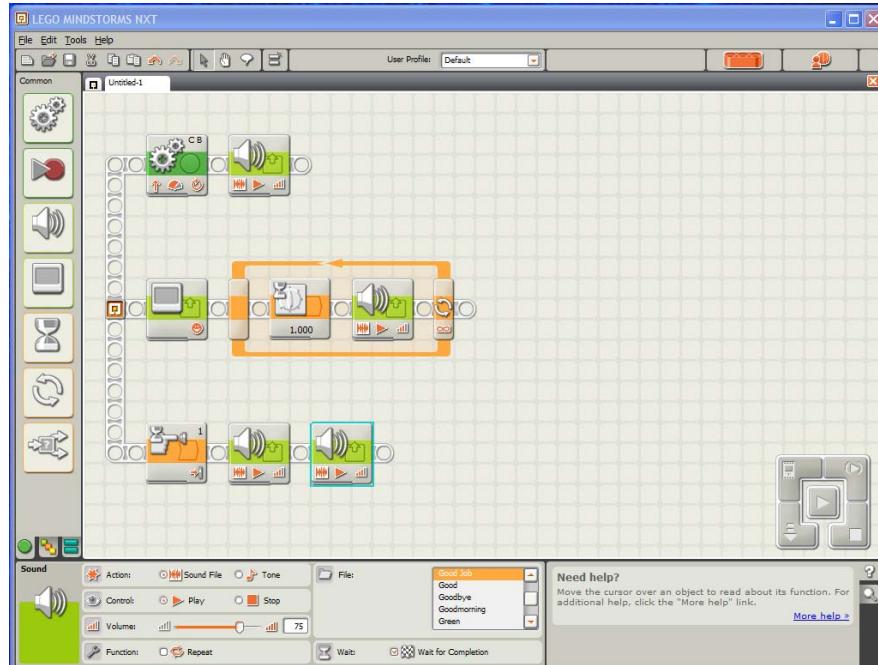


Figure 5-3 Example of Mindstorms interface

More complex programs can be constructed by using loop and switch/decision structures, as well as “wait” blocks which can be likened to event handling in more traditional languages.

It was decided that due to the limited time available for each workshop, the concepts of sequence, looping and event handling (using the sensors) would be covered and that decision structures would be omitted. A sequence of activities was devised that would lead the students through using the programming environment, simple block use and setting of properties, sequence, looping, events (sensor triggers) and more complex combinations of these concepts.

The first workshop activity - equivalent to the ubiquitous “Hello World” first program - involved dragging a sound block to the timeline and then choosing a verbal phrase through the property panel (see Figure 5-4). This first program was then saved

and downloaded into the robot using the control block (see Figure 5-5), and run using the robot's control buttons. The purpose of the first program was to introduce the use of the programming blocks, and show how to compile and download the program to the robot.

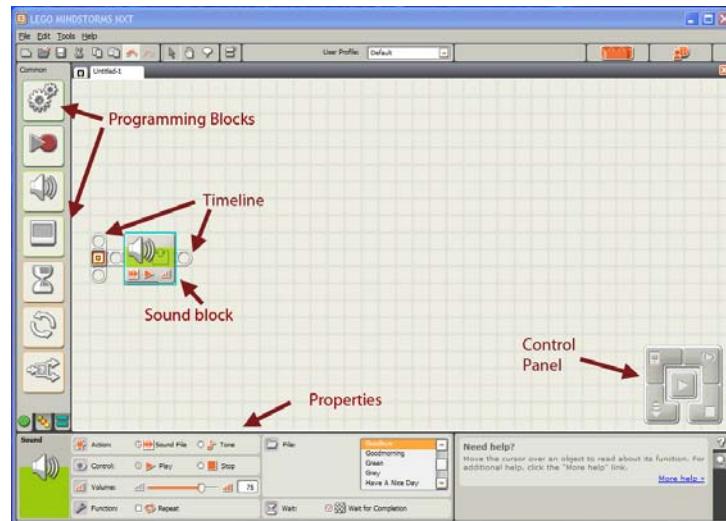


Figure 5-4 Making the robot talk

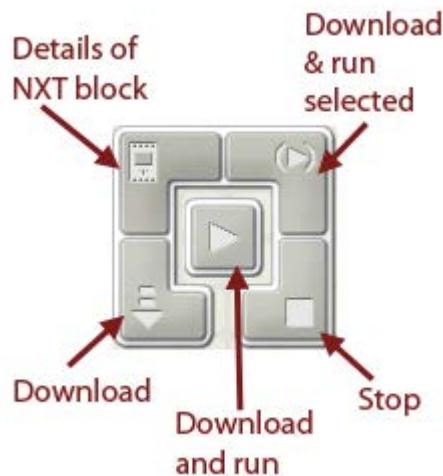


Figure 5-5 Mindstorms Control Panel

The remaining activities are itemised below, along with their purpose.

- ❖ Make the robot say a predefined phrase and then display a picture on the display block [demonstration of sequence and order of program execution];

- ❖ Make the robot display two pictures with a time wait in between them [demonstration of more complex sequence, and settings of properties; introduction of the timer wait block];
 - ❖ Make the robot display a “beating heart” by using two pictures (large heart image and small heart image) repeatedly in conjunction with timer ‘wait’ blocks [introduction to loops, and variants of loops – infinite, timed, count];
 - ❖ Make the robot respond to the touch sensor by saying a predefined sound/word/phrase [introduction to events/sensors];
 - ❖ Make the robot move its arms a predefined number of times [introduction to a new block – ‘move’ – and use of its properties];
 - ❖ Make the robot move its arms a predefined number of times and then wait until a sound is registered by the sound sensor, and walk forward a predefined number of steps [use of sequence, events and properties];
- Make the robot repeat sounds or movement as a wider sequence of actions [consolidation of looping and sequence].

Students were then given free time to complete their own programs (see Figure 5-6). They were encouraged to use their mobile phones to trigger the sound sensor and to explore how the various sensors, programming blocks and combinations worked.

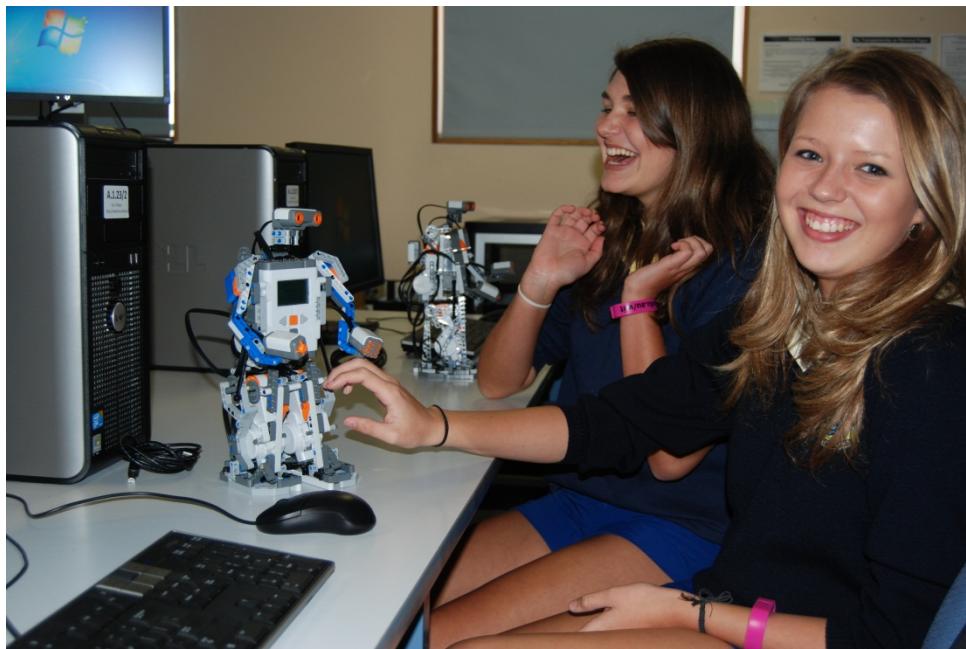


Figure 5-6 Students with robots they were programming

5.4.3.2 Alice workshops

In the Alice workshop, students worked individually on computers, rather than in pairs, as in the Mindstorms workshop. In a similar way to the Mindstorms workshop, the instructor demonstrated worked examples on a large screen to the students as a group, with them then working on individual computers to apply the example to their own work. They were first introduced to the Alice environment as a program that allowed them to create their own “worlds”. Examples of a linear animation and then an animation with some interaction were given, to familiarise the students with what was possible to achieve with simple programming using Alice.

Alice is designed as a language best suited as a first introduction to object-oriented programming (Carnegie Mellon University, 2006) (see Chapter 3). The interface is highly graphical (see Figure 5-7), and as with the Mindstorms NXT programming environment, the process of creating programs is simplified by removing the necessity to remember syntax.

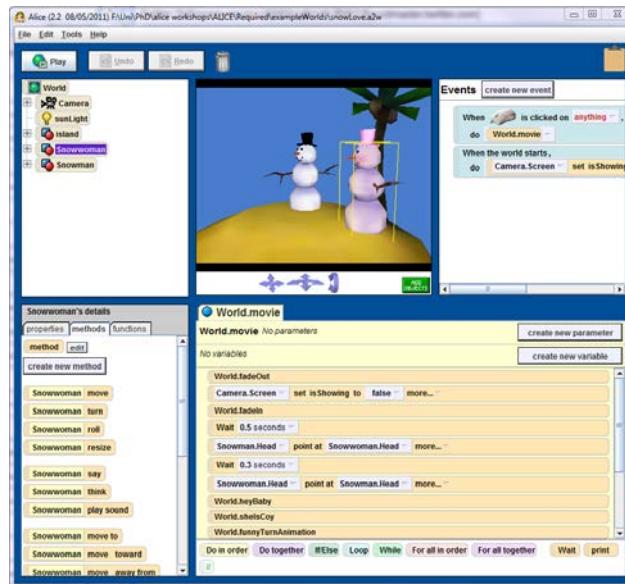


Figure 5-7 Alice interface

In the Alice environment, objects within the user's 'world' are listed with their components in an object list. The methods and properties associated with an object are easily accessible, and methods can be included in the code by dragging and dropping code blocks (see Figure 5-8).

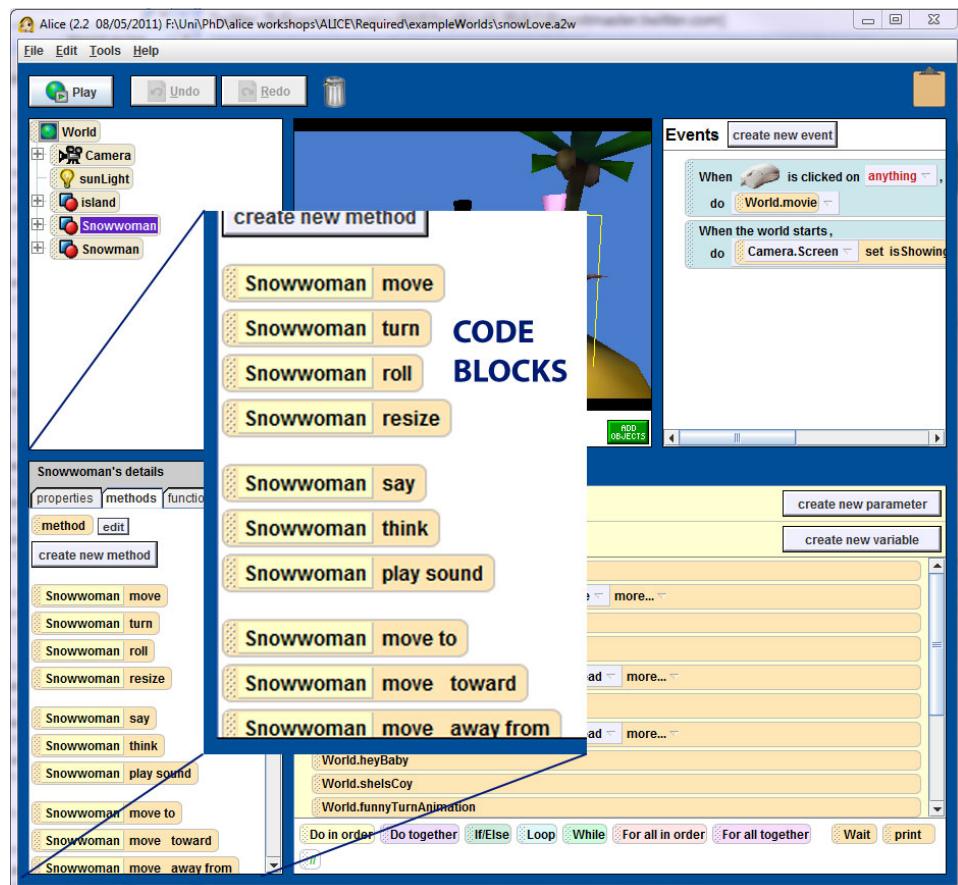


Figure 5-8 Alice - Object method code blocks

More complex code structures (“do together” or “loop”) are accessed by using code blocks at the bottom of the Alice window (see Figure 5-9). Events can be added using the event pane. While properties and other customisation are set by using the keyboard, most of the coding is completed via drag and drop and drop-down list option choice. Objects are also added to the world by choosing from a library, and then dragging and dropping into the ‘world’, and then adjusted using the mouse.

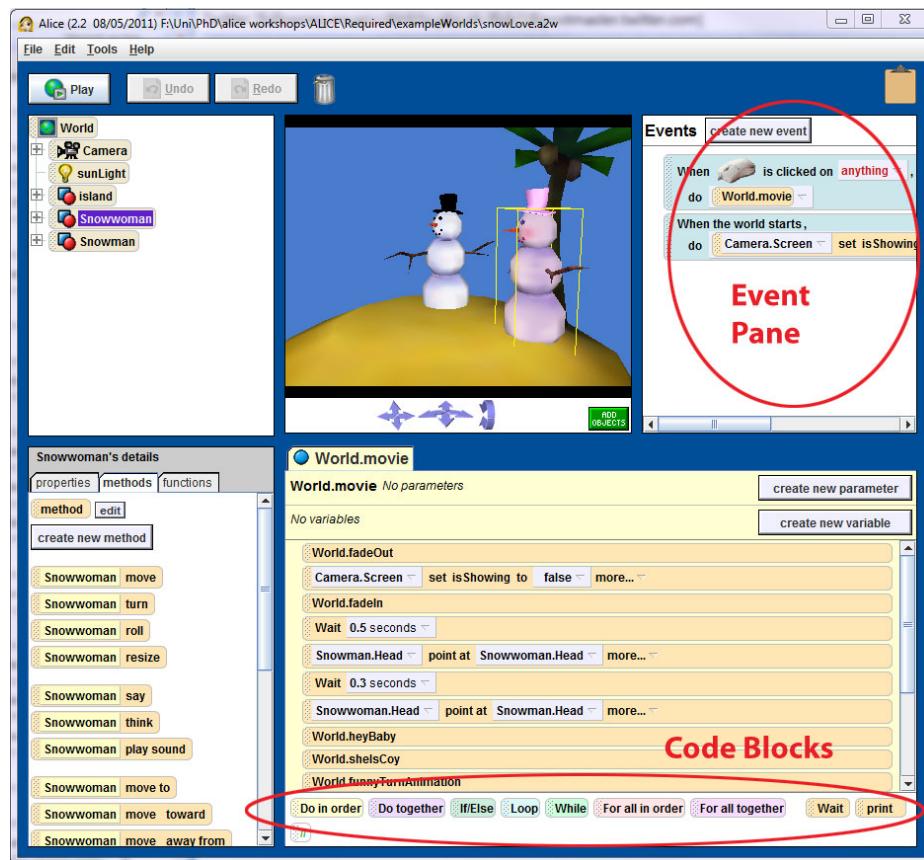


Figure 5-9 Alice - code blocks and events

During the workshop, the instructor demonstrated a series of worked examples in Alice, starting with creating a blank world and placing objects within it. Students were then directed to make their own world on their computer, and to add objects. They were told that their aim was to eventually animate some or all of these objects.

They were then led through a series of programming activities, with demonstrations by the instructor that were followed by them applying these actions to the individual “world” they had created. These demonstrations were designed in the same way as those used in the Mindstorms workshop, using Cognitive Load Theory principles. This involved presentations using multimedia with concurrent narration to avoid split attention effects rather than printed text (modality principle), and small worked examples, with the material delivered in small learner-paced segments (segmentation principle) that moved from simple to more complex examples.

After the initial Alice world was created and objects were added to the world by the students, they were given instruction on the components of the world - the objects, and the parts that made up each object. Properties of an object or part of an object were examined as well as the pre-defined methods that belonged to a particular object. Students were encouraged to use the properties to change objects in their own worlds.

The instructor then led the students through adding a world "first method" (similar to a constructor method for a program class) which created a linear animation sequence in the example world. Students were directed to add animation to their own worlds.

Further code blocks such as the 'do together' block were demonstrated to create more complex code structures. Students were encouraged to explore some of the other structures available as code blocks (eg: loops) and to ask questions of the instructor if they could not determine how to perform a particular action.

As a final worked example, students were shown how to create an interaction with the mouse (event handling), and then how to apply it to their own world.

5.5 Results and Discussion

This section summarises the results of the questionnaire responses in five sections. Section 5.5.1 "Age, Computer Literacy and Programming Experience" summarises the demographic responses to the pre-workshop questionnaire. Section 5.5.2 discusses the responses to the questions asked on both questionnaires regarding gender. Section 5.5.3 reports on the results of the responses of the pre- and post-

workshop questionnaires about the participants' perception of the difficulty of programming (generally) as well as the participant responses on the difficulty of programming with the Mindstorms NXT and Alice programming software. The responses to the questionnaire questions on self-efficacy are reported in Section 5.5.4. Finally, responses to the questions asked on both days about the mental effort expended by participants are summarised and analysed in Section 5.5.5.

One student in the second workshop did not fill out a pre-workshop questionnaire and her post-workshop questionnaire was excluded from the analysis.

5.5.1 Age, Computer Literacy and Programming Experience

At both days the girls ranged in age from 14 to 16 years with the majority being 15 years old ($n = 19, M = 15.0, s = 0.67$). All felt that they had an average level of computer knowledge ($n = 19, M = 4.8, s = 1.42$ on a 9 point Likert scale). Only 3 students indicated that they had any prior programming experience. Of these, two of these students reported that they had "average" programming experience, and one "very low" programming experience. Mean for the group was 1.6 ($n = 19, s = 1.35$) on a 9 point Likert scale where 1 = "no experience" and 9 = "expert".

These results indicate that all participants attending both IT Girls Days were homogeneous with respect to age and computer programming experience, specifically being about 15 years of age and novices in computer programming.

5.5.2 Hypothesis 1 (Gender bias)

H_{A1}: Participants will perceive females to be more capable as computer programmers after the workshop than before the workshop.

H_01 : (Null Hypothesis) There will be no difference in participants' perception of female capability as programmers from before the workshop to after the workshop.

Analysis

At the beginning of the *first* IT Girls Day, the girls were asked to indicate how much they agreed with the statement "Generally, girls are better at programming than boys" on a 9 point Likert scale (where 1 = strongly disagree and 9 = strongly agree). After completing both of the programming workshops, the girls were asked the same question as part of the post-workshop questionnaire. It was anticipated that the opinion of the girls would change as a result of the positive female programmer role models the girls had observed during the workshops.

In the pre-workshop questionnaire five of the ten girls indicated "neither agree nor disagree" to the statement "Generally, girls are better at programming than boys". These girls did not change their answer in the post-workshop questionnaire. However four girls answered "strongly disagree/disagree" (1 or 2 on the scale) [that is, they believed that girls were *not* better at programming than boys] and one girl answered "strongly agree". In the post-workshop questionnaire, of these four, three that strongly disagreed with the statement did not change their opinion and one changed their response to "neither agree nor disagree". The girl who "strongly agreed" that girls were better than boys at programming, modified her answer after the workshops to "neither agree nor disagree".

No significant difference was observed between the answers of the girls before and after the workshops [Wilcoxon Signed Rank test: $W=0$, $n_{s/r} < 10$, not significant]. Note that for all statistical analyses reported throughout this thesis a significance level of $p = 0.05$ is used, unless otherwise indicated. The Wilcoxon Signed Rank test is

used throughout this thesis for non-parametric tests comparing repeated measurements, and is valid for relatively small sample sizes.

More interesting than the actual answers to this specific survey question about gender and programming were the observed reactions of the girls while answering this question. Frowns, confronted body language (pushing back from the desk, away from the questionnaire) and one girl asking "do we have to answer this question?" led the author to conjecture that the phrasing of this survey question about gender in a culture where the programming field is dominated by men may have been confronting to these students. For this reason, the question was changed for the second IT Girls Day.

On the ***second*** IT Girls Day, girls were asked to indicate how much they agreed/disagreed with the statement "Generally, *boys* are better than *girls* at programming". The responses to this question from the second group were quite different from the first group of girls. Of the nine girls who answered this question in the second workshop, seven strongly disagreed with the statement both before and after the workshops. The other two participants provided responses below the scale midpoint which showed they tended to disagree with the statement.

There was no significant difference between the answers of the girls before and after the workshop [Wilcoxon Signed Rank Test: $W=0$, $n_{s/r} < 10$, not significant]. From these results from both days, the null hypothesis H_0 cannot be rejected.

Although the differences between cohorts in answers to this question between Day 1 and Day 2 are interesting, they cannot be compared, as there is a confound of participants being from different schools. The participants in the first workshop were from a different school to the participants in the second workshop. Therefore comparison between the two sets of respondents needs to be considered carefully as it is unknown to what extent the attitudes of each set of respondents have been

influenced from their school base, as opposed to the wording of the question presented to them regarding gender. This question regarding gender is further explored in the study described in Chapter 6.

Comments to the open-ended questions showed the participants found the day to be of benefit and the staff were supportive and provided a positive experience. The numbers of comments were too small to perform a thematic analysis, but two comments explicitly showed a high identification with the program (“Thank you for the experience. We are the IT Girls ☺ Love you guys”) and intent to pursue a program of IT study (“Cya next year! ☺ ^_^”). All comments are included in Appendix A.6.

When participants were asked “What could we have done better?” only 4 students from 20 responded with suggestions such as “More time with everything, especially Alice” and “longer on each task, and less explanation”. All other comments were overwhelmingly positive.

5.5.3 Hypothesis 2 – Programming Difficulty

H_{A2}: Participants will perceive computer programming to be less difficult after the workshop than before the workshop

H₀₂: (Null Hypothesis) Participants will have no difference in their perception of the difficulty of programming from before the workshop to after the workshop.

Analysis

The girls in the workshops were asked to indicate how strongly they agreed with the question “I think programming generally is difficult”. The responses to the post-workshop questionnaire generally showed a lower level of agreement than those

to the pre-workshop questionnaire. The Wilcoxon Signed Rank test was the statistical test used to examine this and it was found that respondents perceived programming to be less difficult after having completed the workshop ($W = -148$, $n_{s/r} = 17$, $z = -3.49$, $p < 0.001$].

Therefore, the null hypothesis H_02 can be rejected in favour of the alternative hypothesis, that participants did experience a change in their perceived difficulty of programming in the direction of "easier".

5.5.3.1 Alice and Mindstorms

Perceptions of the difficulty of programming using Alice and Mindstorms NXT software were also captured on the pre- and post-workshop questionnaires.

The questionnaires included items asking the students to indicate their expectation of the difficulty of programming with Alice. Students indicated that they expected programming with Alice would be moderately easy, and that they were highly interested in programming with it. The responses to these items on post-workshop questionnaires showed that there was no statistically significant difference in interest, although a trend was evident towards a higher level of interest which may represent a real difference [Wilcoxon Signed-Rank Tests: Interest: $W = 63$, $n_{s/r} = 16$, $z = 1.62$, $p = 0.0526$]. However the girls' perception of the difficulty of programming with Alice had significantly changed in the direction of "less difficult" [Wilcoxon Signed-Rank Test: $W = -70$, $n_{s/r} = 14$, $z = -2.18$, $p = 0.0146$].

The students were asked how much they agreed or disagreed with the statement "I found the Alice software difficult to use" (where 1 = strongly disagree, 9 = strongly agree). They generally found the Alice software easy to moderately-easy to use (mean = 2.9; $s = 1.71$).

There was no statistically significant difference in the attitudes of students towards ‘interest in programming’ of the Mindstorms robots evident from the pre- and post-workshop questionnaire responses [Wilcoxon Signed Rank test: $W = -18$, $n_{s/r} = 15$, $z = -0.5$, $p = 0.3085$]. There was a significant difference in the perceived difficulty of programming the robots [Wilcoxon Signed Rank test: $W = -171$, $n_{s/r} = 18$, $z = -3.71$, $p = 0.0001$] in the direction of ‘easier’. This was not an unexpected result, given that it would be expected that interest would be high as participants have ‘self-selected’ in electing to attend the workshops. In regard to the ‘difficulty’, if respondents had not had any prior experience with robotics programming then it would be expected that it would be perceived as a difficult task. Therefore the aim of the workshop to ‘demystify’ some of the misconceptions surrounding programming has been met.

The mean of responses to the difficulty of programming the Mindstorms robots ($M = 3$, $s = 1.52$) in the post-workshop questionnaire showed that the participants found it generally easy to do.

5.5.4 Hypothesis 3 – Self Efficacy

H_A3 : Participants will perceive their self-efficacy in computer programming to be higher after the workshop than before the workshop.

H_03 : (Null Hypothesis) Participants will have no change in self-efficacy in computer programming from before the workshop to after the workshop.

Analysis

The participants were asked to indicate their level of confidence with programming by agreeing or disagreeing with the statement “I feel confident with programming” (1 = strongly disagree, 9 = strongly agree). The girls’ confidence in programming showed

significant improvement after completing the workshop [Wilcoxon Signed-Rank Test, $W = 178$, $n_{s/r} = 19$, $z = 3.57$, $p < 0.001$].

From these results, the null hypothesis H_03 can be rejected in favour of the alternative hypothesis, that participants did experience an increase in self-efficacy.

Summary of Perceptions of Programming Difficulty and Self-Efficacy

The workshops had the effect of decreasing the perception of the difficulty of programming (H_2), and raising students' self-efficacy in computer programming (H_3). The students were also asked to indicate their level of agreement with the statement "I enjoyed the Mindstorms NXT workshop". The responses to this item showed that students enjoyed the Mindstorms NXT programming workshops ($M = 7.85$; $s = 1.39$ where 9 = strongly agree). Similarly, responses regarding level of agreement with the statement "I enjoyed the Alice 3D Worlds workshop" also indicated that the Alice software was highly enjoyable to use ($M = 8.2$; $s = 0.93$).

5.5.5 Mental Effort

At the *first* IT Girls Day students were asked to rate their perceived mental effort while performing various activities in both the Mindstorms NXT and Alice workshops. Likert scales were again used, with 1 = "no mental effort" and 9 = "extreme mental effort". If the students enquired about the meaning of mental effort, the answer "how hard you have to think about something" was given. The students were asked to answer:

- How much mental effort do you estimate you used on understanding and processing what you needed to do on each exercise? [Intrinsic aspect of cognitive load]
- How much mental effort do you estimate you used on navigating and using the Mindstorms NXT [or Alice] programming software? [Extraneous aspect of cognitive load]
- How much mental effort do you estimate you used on learning from the program and reinforcing previous concepts? [Germane aspect of cognitive load]

Students reported moderate levels of mental effort used during the Mindstorms workshop. The results are reported in Table 5-1. There was no statistically significant difference between the responses for each of the three mental effort measures (Friedman Test, $\text{csq}_r = 0.35$, $\text{df} = 2$, $p = 0.8395$).

	Intrinsic	Extraneous	Germane
Median	5	5.5	5
Mode	6	6	4
Min	2	4	4
Max	7	7	7

Table 5-1 Mental Effort during 1st Mindstorms Workshops

Students also reported moderate levels of mental effort during the Alice workshop (see Table 5-2).

	Intrinsic	Extraneous	Germane
Median	5.5	6	5
Mode	6	7	5
Min	3	2	4
Max	6	7	8

Table 5-2 Mental Effort during 1st Alice Workshops

There was no statistically significant difference between the Likert ratings reported for each of the three mental effort measures (Friedman Test, $\text{csq}_t = 0.6$, $\text{df} = 2$, $p = 0.7408$). The Friedman test was used throughout this thesis when a non-parametric version of the parametric repeated measures ANOVA test was needed.

Note that even though the activities in both the Mindstorms NXT programming workshops and Alice programming workshops were designed to be achievable by using small worked examples, guided exercises and learning environments, students participating in the first IT Girls Day reported slightly above average mental effort was required against all three measures of intrinsic, extraneous and germane cognitive load for programming in both Alice and Mindstorms NXT environments.

To attempt to rank these competing aspects of mental effort, the post-workshop questionnaire on the *second* IT Girls Day was altered to request respondents to rank the three comments below in order of the mental effort required to achieve each of the tasks.

"Rank order these in terms of the mental effort you used when completing the activities in the Alice [or Mindstorms] workshop. You should rank 1 for the activity that took the MOST mental effort, 2 for the next and 3 for the activity that took the LEAST mental effort.

__ understanding/processing what you had to do on each exercise.

__ navigating and using the Alice [or Mindstorms NXT] programming software.

__ learning from the program and reinforcing previous concepts."

Students were also asked to rank these in order of importance to themselves.

The number of students who ranked each of the statements as 1, 2 or 3 is summarised in Table 5-3 (Mindstorms NXT) and Table 5-4 (Alice) below:

Rank	Mental Effort			Importance		
	Intrinsic	Extraneous	Germane	Intrinsic	Extraneous	Germane
1	5	2	3	3	5	2
2	2	5	3	6	2	2
3	3	3	4	1	3	6

Table 5-3 Mental Effort & Importance Ranking – 2nd Mindstorms Workshops

Rank	Mental Effort			Importance		
	Intrinsic	Extraneous	Germane	Intrinsic	Extraneous	Germane
1	4	4	2	6	3	1
2	3	2	5	3	3	4
3	3	4	3	1	4	5

Table 5-4 Mental Effort & Importance Ranking – 2nd Alice Workshops

Although there was no statistical pattern detected in any of these rankings (using

Fisher Exact Probability Test: Mindstorms mental effort ranking: $p = 0.5798$;

Mindstorms, importance of aspects ranking: $p = 0.1200$; Alice mental effort ranking:

$p = 0.8298$; Alice, importance of aspects ranking: $p = 0.1585$), trends were seen in the

importance assigned to the dimensions requiring mental effort. The Fisher Exact

Probability Test was used throughout this thesis when an alternative to the Chi-square test was needed as a result of the small size of the sample.

Many educators would likely argue that the most important part of a learning activity is learning from the problem and reinforcing previous concepts (“germane” in the table above). Yet most students ranked this as ‘least important’ to them while completing the activities. In the Mindstorms group, half of the students reported that their greatest source of mental effort was expended on understanding and processing the problem statement.

The limitations of asking students to estimate their own mental effort, or cognitive load in this way are addressed in Section 9.6.13.

5.6 Further Discussion

Both of these IT Girls Days can be considered to be successful, in that they positively impacted the participants’ confidence with programming, and reduced their perceptions of the difficulty of programming (and perhaps the field of Information Technology). This was achieved by providing activities that the participants identified as being enjoyable within the context of a program designed for female students and presented by women. Other authors (Craig and Horton, 2009) have suggested that hands-on workshops would improve the attitude of girls towards IT and the current work reported here supports this observation.

Despite the low numbers of workshop trials (2) and the low number of participants (20) it should be noted that the statistical analyses returned a number of statistically significant findings that have been reported in the previous sections in this chapter. Indeed, of the 19 participants who completed both questionnaires a total of

17 shifted their measure on the Likert scale regarding the perceived level of difficulty with respect to programming towards the direction of “easy”. This alone speaks for the success of the workshops in presenting an intervention that modifies girls’ perceptions of programming in a positive manner.

The nature of the workshop interventions and the specific factors and dynamics involved in causing such a shift were not specifically determined. While the surface cues of the workshops specifically use female presenters, and specifically had only female students present, there are deeper structural aspects within the design of the workshops. The programming software, the instructional materials, and the class activities had all been selected, designed and engineered with the intent of making introductory programming “achievable” at both a conceptual and a practical (implementation) level by reducing the cognitive load associated with learning programming. The specifics of how such designs may have facilitated the learning and perceived easiness of computer programming is of broad interest and potentially much broader application. Further work is needed into the specific factors that may influence the girls’ perception of the programming environments as ‘easy’ or ‘difficult’.

Also of interest were the responses to the survey questions regarding gender, that is the level of agreement with “Generally, boys are better than girls at programming” and “Generally, girls are better than boys at programming”. Even though there was no change as a result of the workshops on either day, there was a difference in the responses between the groups. Whether this was due to the wording used in the gender question, or to other factors inherent in the different pools of participants, is worthy of further investigation.

Note that all participants in these two workshop days were self-selected and are likely to have already held an interest in Information Technology and computer programming. This may have been sparked by the careers visit to their high school or previous exposure to information about IT. They were also from schools in regional and outer regional areas, which although disadvantaged (low socio-economic profile), were not isolated or remote.

Chapter 6 reports on an IT Girls program that was conducted with female students from an isolated school, who did not self-select to participate in the program.

Chapter Six: IT For All Girls

6.1 Introduction

In the previous chapter, two local IT Girls Day outreach programs were described, along with their positive results in modifying attitudes to programming, and raising self-efficacy in programming in the participants. The students participating in these programs were self-selected as having interest in Information Technology after careers visits to their respective schools, and the program was dependent on students travelling to the university campus for the day's activities.

In 2010 the author initiated a program to take the one day campus activities of the IT Girls Day to regional or remote students within their own school context, and to offer the workshops to all female students, not merely those who self-selected. This was done with specific focus on:

1. enabling an outreach program for regional and remote high schools, and
2. testing whether the positive outcomes reported by the local self-selected students would be in evidence amongst a wider population of students.

6.2 Background

6.2.1 Justification for location choice

The choice of location for this program was strongly influenced by the Australian Government's interest in improving regional participation in higher education (Commonwealth of Australia, 2010). In this report concern was expressed at the low and declining level of involvement in higher education: "Of the Australian population aged 15-64 years, 27.9 per cent live in regional or remote areas, whereas only 19.2 per cent of the higher education student population indicate that they are

from regional or remote backgrounds.” (Commonwealth of Australia, 2010). The report identified the following factors as important considerations for a student deciding to continue their studies beyond high school:

- ❖ Location based factors: access, cost, schooling;
- ❖ Individual level factors: socioeconomic status, aspirations.

The report examined data from the Australian Bureau of Statistics (ABS) 2006 census and found that “lower levels of education and occupation appear to be largely associated with lower participation” (p.18) in higher education with economic resources a less important factor. The report also identified perceived value, costs, and ‘prior academic achievement’ as other possible factors. Indeed the government has stated that "by 2020, 20 per cent of domestic undergraduate students must be from low SES (socio-economic status) backgrounds" (Commonwealth of Australia, 2010) and has allocated AUD\$433 million funding to achieve this.

6.2.2 Lightning Ridge

We decided that it would be informative to offer our IT Girls Day program to a high school in a remote area in contrast to the inner regional area where our campus is located, and where previous IT Girls Day programs have been conducted. For a definition of remote and regional areas see the ABS website on the remoteness structure (2010) and DoctorConnect (Dept of Health and Aging, 2011) for an interactive map showing the different remoteness classifications on a map of Australia. The small mining town of Lightning Ridge was chosen for a number of reasons:

- It was possible to drive there within a day;
- The school was neither too big for the program nor too small;

- The school's career advisor was amenable;
- The town is relatively close to a number of other towns so it was possible for other schools to become involved (students from Goodooga also attended); and
- The local school serves a broad and diverse community (see Table 6-1).

Table 6-1 demonstrates key demographic differences between Lightning Ridge and Coffs Harbour and includes figures from Sydney for comparison (Australian Bureau of Statistics, 2006).

Urban Centre	Lightning Ridge	Coffs Harbour	Sydney
Region Classification	Remote Region (RA4)	Inner Regional area (RA2)	Major City (RA1)
Population	2, 602	26, 353	3,641,422
Indigenous Persons	554 (21.3%)	968 (3.7%)	34,279 (0.9%)
Born overseas	500 (19.2%)	3,227 (12.2%)	1,239,659 (34%)
Median weekly family income	\$575	\$931	\$1,386

Table 6-1 Key Demographics

Another issue for people from remote regions is the distance from university. Table 6-2 presents details regarding the distance of Lightning Ridge to various cities and towns that include a university campus. As can be seen from Table 6-2, Lightning Ridge is a considerable distance from the nearest university (CSU at Dubbo) and 710km from the Coffs Harbour campus of SCU.

City (University)	Distance	Type
Dubbo (Charles Sturt University)	380km	Inner Regional
Armidale (University of New England)	530km	Inner Regional
Toowoomba (University of Southern Qld)	610km	Inner Regional
Coffs Harbour (Southern Cross University)	710km	Inner Regional
Sydney	728km	Major City
Brisbane	736km	Major City

Table 6-2 Distance from Lightning Ridge to nearest Universities (details sourced from Google Maps)

Table 6-3 compares school data for two typical Inner Regional high schools in the Coffs Harbour area with the Lightning Ridge and Goodooga schools. Note that Orara and Toormina are high schools with students from years 7 to 12 (secondary) while Lightning Ridge and Goodooga are K-12 (combined). Of interest is the relatively high number of VET qualifications awarded at these remote schools. This is due to a strong vocational focus at both schools and the proximity of a TAFE (Technical and Further Education) campus at Lightning Ridge.

In the following table ‘NAPLAN’ is the acronym for the National Assessment Program – Literacy and Numeracy. NAPLAN tests are conducted annually for all Australian students in Years 3, 5, 7 and 9. Test results allow school literacy and numeracy results to be compared, with higher NAPLAN results indicating better outcomes. ‘Attendance’ refers to the average attendance rate for all students up to and including Year 10 students.

School	Total Enrolments	Indigenous	Attendance	Completed senior school	Awarded VET ²	NAPLAN
Orara (7-12)	638	11%	86%	68	22	518-567
Toormina (7-12)	947	10%	88%	80	32	536-575
Lightning Ridge (K-12)	389	41%	85%	9	14	498-545
Goodooga (K-12)	49	96%	90%	1	2	n/a

Table 6-3 School Characteristics 2009 (Australian Curriculum Assessment and Reporting Authority, 2010)

² Vocational Education and Training (VET)

Lightning Ridge also falls within Southern Cross University's designated feeder region: "... bordered in the south by the NSW Central Coast, west to the NSW/SA border, north to the Darling Downs, Queensland, and east to include the Greater Gold Coast." (Universities Admissions Centre, 2010), so it was seen as a suitable choice for this program.

6.2.3 Framework

The IT Girls Day program takes a three pronged approach to address issues of information, gender and attitudes towards programming:

6.2.3.1 Information:

The program includes a career and study paths session that emphasises the wide variety of careers available in IT. This session also emphasises that IT can be a creative career, and that often IT workers are situated in teams, rather than the stereotype of working alone at a keyboard. Study paths for both University and articulation from VET courses to TAFE and then to University are also outlined. This information session is presented in conjunction with the school's careers teacher, who gives information about traineeships and TVET courses available in Year 11 and 12.

6.2.3.2 Positive Role Models:

The teaching profession has a higher proportion of women than men, yet IT teachers are more likely to be males. Teachers are among students' first IT role models, and this lack of female representation can indicate to students that females are generally not as competent at IT as males (Margolis, 2002). Experience with some in-school programs has shown that for some students, these programs are the first time they have met females who had chosen to pursue higher degrees in IT (Fisher et al., 2009). For this reason, the IT Girls Day program is conducted by current female

students and women who teach and work in IT, who demonstrate technical capability and enthusiasm for the field.

6.2.3.3 Hands-on Activities:

Lynn et al. (2003) suggests that female students show increased interest and confidence in technology when they are exposed to computers in single-sex classes with supportive teaching. The IT Girls Day program gives opportunity for students to do hands-on programming with Lego Mindstorms Robots (<http://www.mindstorms.lego.com>), programming in the Alice 3D programming environment (<http://www.alice.org>) and also some manipulation of graphics in Adobe Photoshop. The instructional design of these activities is intended to reduce extraneous cognitive load, by the use of short, simple, self-contained worked examples, a balance between theory and practical hands-on activities and suitably interesting and simple programming environments. The purpose of these exercises is two-fold; to promote self-efficacy and confidence in the participating students, and to reduce fear regarding the difficulty of programming.

6.2.4 *Other programs*

Several similar programs have been initiated by other organisations to encourage female students to consider studying IT. These programs typically are conducted ‘on-campus’ so participants will have access to the technology needed.

For example, North Sydney Institute of TAFE conducts a **Digi-girls** program which is designed to promote ICT to girls and provide career information. The program is aimed at Year 10 girls but has previously also included girls from years 9, 11 and 12. The Digi-girls program consists of two days of teaching and hands-on exercises, culminating in presentation of the work the girls have done to parents and

school representatives. The program is run on the TAFE campuses (North Sydney Institute of TAFE, 2011).

In Queensland, the CSIRO Education Program runs the **Sunshine Coast Science, Engineering and Technology Expo**. The expo program is aimed at Year 5 and Year 9 students and held at the University of the Sunshine Coast's science faculty. Year 5 students attend a one-day "Labs on Legs" workshop which covers science subjects as well as robotics. Year 9 students are split into separate boys and girls groups and attend a careers panel session as well as participating in a CSIRO workshop (Walsh, 2008).

In Victoria, Swinburne, Monash and Deakin Universities have partnered to conduct a longer-term project in secondary schools. The **Digital Divas** program runs for a semester as a single-sex elective program for Year 8 students, and at time of writing has been expanded to 12 schools around Melbourne. The program uses a mix of engaging curriculum, informal mentoring by university students and role models of ICT career women in the classroom to encourage students to pursue career paths in ICT (Fisher et al., 2009).

Each of these programs focuses on schools in city or inner regional areas. Two other programs have attempted to reach more rural and regional areas of Australia.

The now defunct **Go Go Gidgits Online Computer Club** (Learning Space, 2008) was a part of Education Queensland's Girls and ICT strategy (2003-2008), and was supported by a number of organisations and companies. Members of the club completed computing exercises and then uploaded them to an online space, where they were awarded points. The members with the most points at the end of each 6 month period won prizes. The program closed in 2008 but was open to female school students in years 3 to 12. Around 600 members from schools across Queensland took

part in Go Go Gidgits, including students from remote areas. Although the program had a chat room and a ‘role models’ section that included women working in the IT industry, the online presentation of the club meant that it lacked the face-to-face role model aspect of the IT Girls Program.

A program currently running that has recently reached out to regional girls is the **Robogals** program (Robogals, 2011). Robogals is a student-run organisation that visits schools to introduce girls to engineering and technology through robotics workshops. Based in Melbourne, the program has so far been successful in city areas, but since 2011 has been reaching out to train female university students in regional areas to act as ambassadors for technology. These student ambassadors are trained to visit schools within their regions and promote science and technology to young girls. At time of writing, it is unclear whether these promotional activities will include hands-on activities for students. The Robogals program is also restricted by the availability of female student volunteers in regional university areas who are available to complete the training program.

In contrast to the above programs, the IT Girls Day program has now successfully taken technology to a remote school, and included presentations by female role models and hands-on activities for all female students, not just students who self-select to participate in the program.

6.3 Methodology

6.3.1 Logistics

Discussions between the author, as Women in Technology (WIT) program coordinator and the school careers teacher determined that it was preferable to run three days of the program with the students. The author, another university staff

member, a student ambassador and a technical support person travelled to Lightning Ridge taking all necessary equipment, other than computers, which were made available locally, and ran the workshops within Lightning Ridge Central School. All of the female high-school students were able to attend the program, rather than only a select group, as had occurred in previous IT Girls Days involving visits to campus.

Lightning Ridge School made available the use of a relatively large computer lab for the duration of the stay, coupled with a smaller computer lab which was used for some of the workshop exercises. They also authorised installation of needed software on their computer systems, which negated the need to transport computers to the school.

The local program format had to be modified to fit the differing time constraints of the remote school (start to end of a school day), as well as including careers and study information on the same day as the hands-on activities. The school had set morning tea and lunch times, and the program had to be fitted around these. We found that the periods allocated for the hands-on activities were different to the timings we had previously used. For this reason we decided to add another type of hands-on activity - manipulating images in Photoshop - and combine this with the Mindstorms Robotics session to create two shorter sessions as well as one longer session on Alice 3D programming. This enabled the students to have greater time to explore in more detail the more complex Alice 3D environment.

The program was conducted over six days - two days being used for travel, one day for meeting School staff, setting up software and familiarising University staff with the school, and three "IT Girls Day" programs. It was also decided after consulting with school staff on the first day, that short careers sessions should also be conducted for the male students, so that they would not feel excluded. The technical

support person from Southern Cross University was male, so he volunteered to conduct these sessions at the end of the day while the careers and study information sessions were being held for the girls. All of the male students in years 7 to 11 participated in the careers information sessions. Altogether 55 female students aged 11 to 17 years were included in the program. The same program was conducted on each day, with a different school year cohort each day.

6.3.2 Program

The Lightning Ridge program was based upon the successful IT Girls Day activities described in Chapter 5. Recall that these original IT Girls Days activities involved a careers visit to Year 9 students at a local high school, where IT careers information and study paths were outlined. Interested female students were then invited back to the campus for the IT Girls Day, with a maximum of 12 students able to be accommodated in any single day. In Lightning Ridge, the careers sessions needed to be incorporated into the day's activities, and the sizes of the groups plus the purchase of an additional two Mindstorms kits meant that Mindstorms workshop participants could individually program a robot instead of working in pairs.

Each IT Girls Day program began with students completing a pre-program questionnaire with demographic questions (name, age and year at school), and 9 point Likert scale questions about their level of computer knowledge or skill, their programming experience, their level of knowledge of IT careers and whether they had considered a career in IT.

Participants were also asked questions to ascertain their perceptions of the difficulty of programming, their confidence with programming and attitude towards Alice 3D worlds and Lego Mindstorms NXT (perceived difficulty and confidence)

and attitude toward the difference between females' and males' competency in programming. At this stage, half of the students were given version A of this questionnaire and the remainder version B. The only difference between these two questionnaires was the phrasing of the question on gender. See Section 6.9 of this chapter for more information on this question. Both pre-workshop questionnaires are included in Appendix B.1 and B.2 of this thesis. The completion of the questionnaire was followed with a short introduction to the staff and the reason for the staff's visit, within a large computer lab.

The students were then divided at random into two groups, being careful to have equivalent numbers of students given version A and version B of the questionnaire in each group. One group (Group A) was taken to a smaller computer lab where they participated in an Alice 3D programming workshop detailed in Section 6.3.3. The other group (Group B) was divided into two smaller groups, one of which participated in a Lego Mindstorms NXT programming workshop (see Section 6.3.4 for details of this workshop) within the large lab, while the other group went outside to take photos which were then manipulated with Adobe Photoshop software on the computers on the "other side" of the large computer lab. The Photoshop students and Mindstorms students in Group B then swapped workshops while the Alice 3D students in Group A continued in the other computer lab.

After the first set of workshops, all students completed a questionnaire dependent on which programming group they were in (see Appendix B.3 and B.4) and then were released for recess. After recess, Group A was divided into two groups for the Photoshop and Mindstorms activities, and Group B went to the smaller computer lab for the Alice 3D session. After these hands-on activities, each group

completed a second questionnaire about the programming session in which they had just participated.

Program staff then cooked a BBQ “sausage sizzle” for lunch for the students who had participated in the program, and made themselves available for informal discussions with the students.

After lunch, the students returned for a careers and study information session. A video on IT careers was shown and then staff led a discussion about the types of IT careers available, concentrating on such positive aspects as flexibility, variety of careers and travel opportunities, and attempting to dispel the stereotyped images that the students may have had about IT workers.

IT study paths were also discussed, in varying detail dependent on student age. After the information session, the participating students completed one of two versions of a post-program evaluation questionnaire (see Appendix B.5 and B.6), which repeated the questions about knowledge of IT careers, intention to pursue an IT career, perceived difficulty of programming, confidence with programming, and perception of gender difference in programming. Again, the only difference between the two questionnaires was the order of the wording of the gender question.

Open questions were also included in these post-workshop questions, about what was enjoyed most about the workshops, what improvements could have been made, and an open opportunity to offer any other comments. Students were given a sample bag with information about IT careers, lollies, information about study options and the WIT program, and a CD with Alice 3D install and tutorial files. The CD became a coveted item with University staff witnessing several of the boys in the school offering to barter with the girls for a copy.

6.3.3 Alice Workshops

The instructional material used for the remote IT Girls Days Alice workshops was very similar to that used for the on-campus IT Girls Days. The workshop was again presented in small groups (up to 10 students), with a female instructor. Students worked individually on computers, and were given instruction as a group (short worked example) and then instructed to apply the instruction to their own “world” in Alice. As in the local IT Girls Day workshops, participants were given instruction on:

- ❖ Creating a world;
- ❖ Adding objects to the world;
- ❖ Identifying and changing properties of objects and sub-objects;
- ❖ Creating a linear animation sequence using a ‘world’ method;
- ❖ Using code blocks such as “do together” to create more complex code structures;
- ❖ Exploring other code blocks such as loops;
- ❖ Creating an interaction with the mouse (events).

Students were encouraged to personalise their creations as much as possible, and had comparatively more ‘free’ time (approximately 25 mins compared to 10 minutes) on exploring the code blocks and personalising their world than the on-campus students had experienced.

6.3.4 Mindstorms Workshops

Students had less time (40 minute workshops compared to 1 hour) in the Mindstorms workshops than experienced by participants in the local IT Girls Day workshops. Students worked by themselves with a Mindstorms robot and a computer, and the author demonstrated each small worked example (as in Section 5.4.2.3) and then each student completed the activity with her robot. As in previous workshops,

the activities built from simple activities involving the placement of one programming block and setting of one property, to complex programs with loops, events and longer sequences. Students were given a short time at the end of the instructional period to further explore the programming blocks and to create their own program.

6.4 Hypotheses

The hypotheses that were to be tested in this study are listed below:

- ❖ **H1 (Career Aspirations):** Participants will report higher likelihood to pursue a career in Information Technology after the workshop than before the workshop.
- ❖ **H2 (Information):** Participants will report increased knowledge of Information Technology careers after the workshop than before the workshop.
- ❖ **H3 (Programming Difficulty):** Participants will perceive computer programming to be less difficult after the workshop than before the workshop.
- ❖ **H4 (Self-efficacy):** Participants will report their self-efficacy in computer programming to be higher after the workshop than before the workshop.
- ❖ **H5 (Gender bias):** Participants will perceive females to be more capable as computer programmers after the workshop than before the workshop.

6.5 Day 1 Results and Discussion

6.5.1 Age, Computer Literacy and Programming Experience

On the first IT Girls Day (Years 10 and 11), 17 girls participated. Of these participants, two participants declined to finish the day, and one participant had very limited English skills. These participants' results have been omitted from the study. The remaining participants on this first IT Girls Day ranged in age from 15 to 17 (mean=15.9, $s = 0.66$). All participants reported that they had an average level of computer knowledge (median = 5 on a 9 point Likert scale), and none or very little programming experience (median = 2 on a 9 point Likert scale asking "How much programming have you done?" where 1 = 'none' and 9 = 'Lots!').

These results indicate that all participants attending the first IT Girls Day were homogeneous with respect to age, computer knowledge and computer programming

experience, specifically being about 16 years of age, average computer knowledge and novices in computer programming.

6.5.2 Hypothesis 1 (Career Aspirations)

H_{A1} (Career Aspirations): Participants will report higher likelihood to consider a career in Information Technology after the workshop than before the workshop.

H₀₁: (Null Hypothesis) Participants will report no change in their intention to consider a career in Information Technology after the workshop from before the workshop.

Analysis

Participants were asked to indicate on a 9-point Likert scale how much they agreed or disagreed with the statement "I am considering a career in IT" (1 = strongly disagree, 9 = strongly agree). Note that this question was not asked of the participants of the campus-based IT Girls Days (see Chapter 5), as those girls were self-selected after participating in the careers information sessions in their schools. It was hypothesised that the participants from the Lightning Ridge School, who did not self-select to participate, would be more likely to consider a career in IT after the workshop activities than beforehand.

Medians for the Likert scale results for career aspirations, knowledge of IT careers, confidence (self-efficacy) and perceived difficulty of programming for all three IT Girls Days are given in Table 6-8 in Section 6.8, and comparisons between the individual days are discussed in that section.

The attitudes of the participants towards a career in IT improved after completing the workshop. The likelihood of choosing a career in IT was significantly

higher after the workshop [Wilcoxon Signed-Rank Test, $W = 55$, $n_{s/r} = 10$, $z = 2.78$, $p = 0.0027$].

From these results, the null hypothesis H_01 can be rejected in favour of the alternative hypothesis, for this group, that participants were more likely to consider a career in IT after the workshop, than before the workshop.

6.5.3 Hypothesis 2 (Information)

H_A2 (Information): Participants will report increased knowledge of Information Technology careers after the workshop than before the workshop.

H_02 : (Null Hypothesis) Participants will report no change in their knowledge of Information Technology careers after the workshop from before the workshop.

Analysis

Participants on the first day reported a significantly higher level of knowledge of IT careers ("I know a lot about careers in IT") after the IT Girls Day than at the beginning of the day. Measures were tested using Wilcoxon Signed-Rank Test: $W=78$, $n_{s/r} = 12$, $z = 3.04$, $p = 0.0012$.

From these results, the null hypothesis H_02 can be rejected in favour of the alternative hypothesis, for the first day participants, that participants reported increased knowledge of IT careers. This is particularly important for this age group (Year 10 and 11) when they are choosing subjects for the Higher School Certificate, and moving towards the transition between school and further study, or school and the workforce.

6.5.4 Hypothesis 3 (Programming Difficulty)

H_A3 (Programming Difficulty): Participants will perceive computer programming to be less difficult after the workshop than before the workshop.

H₀3: (Null Hypothesis) Participants will have no change in the perceived difficulty of programming after the workshop from before the workshop.

Analysis

Participants were asked to indicate their level of agreement with the statement "I think programming generally is difficult" from 1 = strongly disagree to 9 = strongly agree.

Participants shifted their perception of the difficulty of programming from a median of 6.5 before the workshops to a median of 3 afterwards. The change was statistically significant at $p < 0.05$ (Wilcoxon Signed-Rank Test: $W = -53$, $n_{s/r} = 12$, $z = -2.06$, $p = 0.0197$). The null hypothesis H_0 is therefore rejected in favour of the alternative hypothesis, that participants experienced a change in the perceived difficulty of programming in the direction of "easier".

6.5.4.1 Alice Programming

Participants were also asked about their opinion of the difficulty of programming with the Alice software, before and after the workshop. At the beginning of the IT Girls Day, they were asked how much they agreed/disagreed with the statement "Programming with Alice (3D Worlds) sounds difficult to me" (Likert scale, 1 = strongly disagree, 9 = strongly agree). Median score for this question was 7, prior to the workshop.

After the Alice workshop, participants were asked how much they agreed/disagreed with the statement "Programming Alice to do things is difficult" (9

point Likert scale, 1 = strongly disagree, 9 = strongly agree). The median score after the workshop was 3.5.

The difference in scores before and after the workshop is statistically significant (Wilcoxon Signed Rank test: $W=-65$, $n_{s/r} = 12$, $z = -2.53$, $p = 0.0057$).

6.5.4.2 Mindstorms Programming

The corresponding questions were asked regarding programming with the Mindstorms software – “Programming robots to do things is probably difficult” and “programming robots to do things is difficult”. The median scores for these questions were 7.5 and 3, respectively. The difference in scores before and after the workshop in the direction of “easier” was again statistically significant (Wilcoxon Signed Rank test: $W=-57$, $n_{s/r} = 11$, $z = -2.51$, $p = 0.006$).

This group of participants had a significant change in their perception of the difficulty of programming in the direction of “easier” on all measures used.

6.5.5 Hypothesis 4 (Self-Efficacy)

H_A4 (Self-Efficacy): Participants will report their self-efficacy in computer programming to be higher after the workshop than before the workshop.

H₀4: (Null Hypothesis) Participants will experience no change in self-efficacy after the workshop from before the workshop.

Analysis

Participants were asked to indicate how much they agreed or disagreed with the statement "I feel confident with programming" on a 9-point Likert scale where 1 = "strongly disagree" and 9 = "strongly agree". Before the workshop, the median score

was 1, and afterwards the median was 6. This shift in scores was statistically significant (Wilcoxon Signed Rank test: $W=66$, $n_{s/r} = 11$, $z = 2.91$, $p = 0.0018$).

The null hypothesis H_0 is therefore rejected in favour of the alternative hypothesis, that participants experienced their self-efficacy in computer programming to be higher after the workshop than before the workshop.

6.5.6 Other Benefits

6.5.6.1 Enjoyment

Enjoyment of all of the activities was uniformly high – “I enjoyed the Alice workshop” Likert scale measures had a median of 9 (“strongly agree”), “I enjoyed the Mindstorms workshop” Likert scale measures returned a median of 8 (9 = “strongly agree”), and “I enjoyed the Photoshop workshop” Likert scale measures returned a media of 8.5 (9 = “strongly agree”). Of interest is that the programming activities results in similar enjoyment to the non-programming activity of Photoshop manipulations.

6.5.6.2 Interest

This first group of participants had a statistically significant change in their interest in programming with Alice - “programming with Alice (3D worlds) sounds interesting to me” – from before the workshops to after the workshops, in the positive direction, using Wilcoxon Signed Rank test: $W=62$, $n_{s/r} = 11$, $z = 2.73$, $p = 0.0032$. There was no statistically significant change in interest in programming with the Mindstorms robots (Wilcoxon Signed Rank test: $n_{s/r} < 10$).

This may be a ceiling effect, as the participants' interest was high in using the Mindstorms robots before the workshop (median = 7.5) and stayed high after the workshop (median = 7.5).

6.5.6.3 Participant Comments

Participants were asked what they enjoyed most about the activities, what the organisers could have done better, and for any other comments they would like to offer. Most responses were very positive:

- “it is fun you get to learn a lot of new stuff about everything and a lot about computers”;
- “doing everything it was mad”;
- “that they were hands-on and we had free-reign over it”;
- “It was creative and interesting. I learned things today I've never even thought about before”;
- “Alice was very interesting”.

Several girls commented that they would have liked more time with the robots and Alice, and one girl commented that having printouts of the pictures created in the Photoshop workshop would have added to the day's enjoyment. In the open comments, several girls gave unprompted thanks for the visit, and two girls offered comments on the IT career information specifically: “thank you for letting me know about IT” and “Thank you for identifying this career for me”.

6.5.7 Mental Effort Measures

Participants were asked to indicate their mental effort expended on various aspects of the programming activities, in a similar fashion to the local IT Girl Day

participants described in Chapter 5. The participants were asked these questions immediately following each workshop (Alice and Mindstorms). As the author had found that the participants in the local IT Days had often asked for the definition of “mental effort” in this section of the questionnaire, statements were re-worded as below:

- ❖ “I had to think hard to understand what I had to do in each exercise” [Intrinsic cognitive load];
- ❖ “I had to think hard to navigate and use the Alice software”/” I had to think hard to navigate and use the NXT programming software” [Extraneous cognitive load];
- ❖ “I had to think hard to learn from the program and understand concepts” [Germane cognitive load].

Participants were asked to indicate their agreement or disagreement with the statements, on a 9 point Likert scale, where 1 = “strongly disagree” and 9 = “strongly agree”.

Medians and ranges of all measures are given in Table 6-4 below. The levels of mental effort experienced by the girls while participating in Alice and Mindstorms activities were not significantly different (Friedman’s 2-way Analysis of Variance test: Alice: $csq_r = 2.04$, $df = 2$, $p = 0.3606$; Mindstorms: $csq_r = 2.25$, $df = 2$, $p = 0.3247$).

	Intrinsic	Extraneous	Germane
Alice programming	Median = 3.5 Min = 1, max = 8	Median = 4 Min = 1, max = 8	Median = 5 Min = 1, max = 8
Mindstorms programming	Median = 2.5 Min = 1, max = 6	Median = 2.5 Min = 1, max = 7	Median = 4.5 Min = 1, max = 9

Table 6-4 Measures of mental effort on Alice and Mindstorms - Day 1

There was a wide range in the mental effort experienced by the girls - even though the activities were designed to reduce cognitive load for these participants, many participants reported significant mental effort, while others reported no to little mental effort.

6.6 Day 2 Results and Discussion

6.6.1 Age, Computer Literacy and Programming Experience

The second IT Girls Day at the school included participants from Years 8 and 9, and one girl from Year 10. The age range of this cohort was from 13 to 16 years, with the mean age 14.3 (standard deviation 0.7). The participants on this second day all felt they had an average level of computer knowledge (median 5 on 9 point Likert scale) and little to no programming experience (median 1 where 1 = “little or none” on the scale).

These results indicate that all participants attending the second IT Girls Day were homogeneous with respect to age, computer knowledge and computer programming experience, specifically being about 14 years of age, average computer knowledge and novices in computer programming.

Four participants did not complete the day (2 students truanted and 2 students needed to leave before the end of the day), and these participants’ results have been excluded from the analysis of the results. The remainder of the participants (16 girls) completed the same activities as the girls on the first day.

6.6.2 Hypothesis 1 (Career Aspirations)

H_A1 (Career Aspirations): Participants will report higher likelihood to pursue a career in Information Technology after the workshop than before the workshop.

H₀1: (Null Hypothesis) Participants will report no change in their intention to pursue a career in Information Technology after the workshop from before the workshop.

Analysis

Before the workshop, the median score (using a 9-point Likert scale for the question outlined in Section 6.5.2) was 1.5, and after the workshop the median score recorded was 5.5. The likelihood of choosing a career in IT was significantly more after the workshop [Wilcoxon Signed-Rank Test, W = 101, n_{s/r} = 14, z = 3.15, p = 0.0008].

From these results, the null hypothesis H₀1 can be rejected in favour of the alternative hypothesis, for this younger group, that participants were more likely to consider a career in IT after the workshop, than before the workshop.

Again, this is important as girls in the latter part of Year 9 are starting to think about careers and deciding on subject areas for the HSC dependent on these career choices.

6.6.3 Hypothesis 2 (Information)

H_A2 (Information): Participants will report increased knowledge of Information Technology careers after the workshop than before the workshop.

H₀2: (Null Hypothesis) Participants will report no change in their knowledge of Information Technology careers after the workshop from before the workshop

Analysis

Participants on the second day reported a significantly higher level of knowledge of IT careers ("I know a lot about careers in IT") after the IT Girls Day activities than at the beginning of the day. Measures were tested using Wilcoxon Signed-Rank Test: $W=105$, $n_{s/r} = 14$, $z = 3.28$, $p = 0.0005$.

From these results, the null hypothesis H_02 can be rejected in favour of the alternative hypothesis, for the second day participants, that participants reported increased knowledge of IT careers.

6.6.4 Hypothesis 3 (Programming Difficulty)

H_A3 (Programming Difficulty): Participants will perceive computer programming to be less difficult after the workshop than before the workshop.

H_03 : (Null Hypothesis) Participants will have no change in the perceived difficulty of programming after the workshop from before the workshop.

Analysis

On the second day, participants were again asked to indicate their level of agreement with the statement "I think programming generally is difficult" from 1 = strongly disagree to 9 = strongly agree.

There was a trend downwards in participants' perception of the difficulty of programming - from a median of 4.5 before the workshops to a median of 3.5 afterwards - however this shift was not statistically significant at $p < 0.05$ (Wilcoxon Signed-Rank Test: $W=-5$, $n_{s/r} = 11$, $z = -0.2$, $p = 0.42$). The null hypothesis H_03 therefore cannot be rejected for this group of participants.

6.6.4.1 Alice Programming

Participants were also asked about their opinion of the difficulty of programming with the Alice software, before and after the workshop, as in Section 6.5.4.1. Median score before the workshop was 6, and after the workshop was 5. The difference in scores before and after the workshop was not statistically significant (Wilcoxon Signed Rank test: $W=-39$, $n_{s/r} = 15$, $z = -1.09$, $p = 0.1379$).

6.6.4.2 Mindstorms Programming

Again, the corresponding questions were asked regarding programming with the Mindstorms software. The median scores for these questions were 6.5 and 5, respectively. The difference in scores before and after the workshop in the direction of “easier” was again not statistically significant (Wilcoxon Signed Rank test: $W=-34$, $n_{s/r} = 14$, $z = -1.05$, $p = 0.1469$).

6.6.5 Hypothesis 4 (Self-Efficacy)

H_A4 (Self-Efficacy): Participants will report their self-efficacy in computer programming to be higher after the workshop than before the workshop.

H₀4: (Null Hypothesis) Participants will experience no change in self-efficacy after the workshop from before the workshop.

Analysis

Agreement with the statement "I feel confident with programming" was reported with a median of 2.5 before the workshop and 5 after the workshop (on the same Likert scale as reported in Section 6.5.5). The shift in scores was again statistically significant (Wilcoxon Signed Rank test: $W=104$, $n_{s/r} = 15$, $z = 2.94$, $p = 0.0016$).

The null hypothesis H_04 is therefore rejected in favour of the alternative hypothesis, that participants experienced their self-efficacy in computer programming to be higher after the workshop than before the workshop, for this second group of participants.

Interestingly, the perception of the difficulty of programming (generally, as well as the difficulty of programming with Alice and Mindstorms) did not significantly change with this second group of participants, however their self-efficacy in their ability to program did shift significantly in a positive direction. This demonstrates the capacity of these programs to raise self-efficacy whilst not fostering the belief that programming is effortless.

6.6.6 Other Benefits

6.6.6.1 Enjoyment

Participants on this second “IT Girls Day” also experienced high levels of enjoyment. Medians for Alice programming, Mindstorms programming and the Photoshop activity were 8.5, 9 and 9 respectively. Again, the enjoyment experienced by the participants during programming activities was comparable to that experienced during the non-programming activity of Photoshop image manipulations.

6.6.6.2 Interest

There was no significant difference in the interest level in programming Alice after the workshop compared to before the workshop, for this second group of participants (medians 7 before and 7.5 afterwards, Wilcoxon Signed Rank test: $W=36$, $n_{s/r} = 11$, $z = 1.58$, $p = 0.0571$). Interest was high throughout.

There was a significant difference in interest in Mindstorms after the workshop - medians 6.5 before the workshop and 7.5 afterwards, Wilcoxon Signed Rank test: $W=82$, $n_{s/r} = 14$, $z = 2.56$, $p = 0.0052$.

Several participants on this day expressed reluctance to use the robots initially, which may have contributed to the differences seen in interest between Alice and Mindstorms. Interest and enjoyment of both Alice and Mindstorms stayed relatively high throughout.

6.6.6.3 Participant Comments

When asked what they enjoyed most about this day's activities, 8 of the 16 participants made mention of Photoshop activities, however 3 participants commented that they enjoyed Alice the most, 2 commented on programming the robots, and 4 participants said "All of them!". When asked what the organisers could have done better, one participant commented "Robotics could have been more interesting" and another "Alice was a bit boring". Considering that these workshops were presented to *all* female students, rather than those who had expressed an interest in IT, the minimal number of negative comments is remarkable.

Other general comments offered included "Alice is awesome. I want to do it again." And "A very good and enjoyable day =)".

6.6.7 Mental Effort Measures

On Day 2, the participants were again asked about "how hard" they had to think whilst completing the Alice and Mindstorms activities. For details of the questions, see Section 6.5.7. Medians and range for each of the aspects of cognitive load are reported in Table 6-5 below:

	Intrinsic	Extraneous	Germane
Alice programming	Median = 5 Min = 1, max = 9	Median = 4 Min = 1, max = 9	Median = 4.5 Min = 1, max = 9
Mindstorms programming	Median = 3.5 Min = 1, max = 9	Median = 3.5 Min = 1, max = 9	Median = 3 Min = 1, max = 9

Table 6-5 Measures of mental effort on Alice and Mindstorms - Day 2

There was no significant difference between intrinsic load, extraneous load and germane load reported for each of the two programming workshops (Friedman's 2-way Analysis of Variance test: Alice: $\text{csq} = 1.53$, $\text{df} = 2$, $p = 0.4653$; Mindstorms: $\text{csq} = 0.41$, $\text{df} = 2$, $p = 0.8146$).

Although the medians of the measures reported for all three aspects of mental effort were moderate, the individual range of scores varied from 1 to 9, showing that again, some participants were experiencing significant mental effort while completing the exercises.

6.7 Day 3 Results and Discussion

6.7.1 Age, Computer Literacy and Programming Experience

On the third IT Girls Day, 16 girls from Years 6 and 7 at the Lightning Ridge School, and 2 girls from Goodooga School participated. Unfortunately the participants from Goodooga School needed to leave before the end of the workshop, and their partially-completed questionnaires have been excluded from the analysis of the results. The mean age of the Year 6 and 7 girls was 12.1 (standard deviation = 0.81) and the girls reported that they had average computer literacy (median = 5 on 9-point

Likert scale) and little to no programming experience (median = 1 - “none” on a 9-point Likert scale).

These results indicate that all participants attending the third IT Girls Day were homogeneous with respect to age, computer knowledge and computer programming experience, specifically being about 12 years of age, average computer knowledge and novices in computer programming.

6.7.2 Hypothesis 1 (Career Aspirations)

H_{A1} (Career Aspirations): Participants will report higher likelihood to pursue a career in Information Technology after the workshop than before the workshop.

H₀₁: (Null Hypothesis) Participants will report no change in their intention to pursue a career in Information Technology after the workshop from before the workshop.

Analysis

The median score for this measure before the workshop for the third group of girls (in Years 6 and 7) was 3.5 (on the 9-point Likert scale) and 4.5 after the workshop.

The likelihood of choosing a career in IT was significantly more after the workshop (Wilcoxon Signed-Rank Test, W = 50, n_{s/r} = 12, z = 1.94, p = 0.026)]. From these results, the null hypothesis H₀₁ can be rejected in favour of the alternative hypothesis, for this youngest group, that participants were more likely to consider a career in IT after the workshop, than before the workshop. Note that the previous two group results were significant at the p < 0.01 level, but this result is significant at the

p<0.05 level. This reduced impact could be expected in a group who are younger, and most probably have not started considering a career at this stage.

6.7.3 Hypothesis 2 (Information)

H_A2 (Information): Participants will report increased knowledge of Information Technology careers after the workshop than before the workshop.

H₀2: (Null Hypothesis) Participants will report no change in their knowledge of Information Technology careers after the workshop from before the workshop.

Analysis

Participants on the third day again reported a significantly higher level of knowledge of IT careers ("I know a lot about careers in IT") after the IT Girls Day activities than at the beginning of the day. Measures were tested using Wilcoxon Signed-Rank Test: W=110, n_{s/r} = 15, z = 3.11, p = 0.0009.

From these results, the null hypothesis H₀2 can be rejected in favour of the alternative hypothesis, for the third day participants, that participants reported increased knowledge of IT careers.

6.7.4 Hypothesis 3 (Programming Difficulty)

H_A3 (Programming Difficulty): Participants will perceive computer programming to be less difficult after the workshop than before the workshop.

H₀3: (Null Hypothesis) Participants will have no change in the perceived difficulty of programming after the workshop from before the workshop.

Analysis

Participants on the third day were again asked to indicate their level of agreement with the statement "I think programming generally is difficult" from 1 = strongly disagree to 9 = strongly agree.

In a similar manner to the results for Day 2, there was a trend downwards in participants' perception of the difficulty of programming - from a median of 3.5 before the workshops to a median of 1.5 afterwards - however this shift was again not statistically significant at $p < 0.05$ (Wilcoxon Signed-Rank Test: $W=-55$, $n_{s/r} = 16$, $z = -1.41$, $p = 0.0793$). The null hypothesis H_0 therefore cannot be rejected for the participants on Day 3.

6.7.4.1 Alice Programming

Participants were again also asked about their opinion of the difficulty of programming with the Alice software, before and after the workshop, as in Section 6.5.4.1. Median score before the workshop was 5, and after the workshop was 1.5. The difference in scores before and after the workshop was statistically significant (Wilcoxon Signed Rank test: $W=-106$, $n_{s/r} = 15$, $z = -3$, $p = 0.0013$).

6.7.4.2 Mindstorms Programming

As in Section 6.5.4.2, the corresponding questions were asked regarding programming with the Mindstorms software. The median scores for these questions were 6 and 1, respectively. The difference in scores before and after the workshop in the direction of "easier" was again statistically significant (Wilcoxon Signed Rank test: $W=-122$, $n_{s/r} = 16$, $z = -1.05$, $p = 0.1469$).

Interestingly, these participants' experience of the Alice and Mindstorms programming activities were that they were easier than expected, and yet the downward trend in reported perception of the "difficulty of programming generally" was not statistically significant. This could be due to a floor effect, as the initial

perception of the difficulty of programming was not high (median 3.5) in this group.

The participants in this third group may also have been predisposed to believe that the day would be easier from contact with older participants who had already experienced the program, in this relatively small school, or perhaps these participants did not associate ‘Mindstorms’ and “Alice” with programming initially.

6.7.5 Hypothesis 4 (Self-Efficacy)

H_A4 (Self-Efficacy): Participants will report their self-efficacy in computer programming to be higher after the workshop than before the workshop.

H₀4: (Null Hypothesis) Participants will experience no change in self-efficacy after the workshop from before the workshop.

Analysis

Again, participants were asked how much they agreed with the statement "I feel confident with programming". Before the workshop activities, the median score was 3.5, and afterwards the median was 9. This shift in scores was statistically significant (Wilcoxon Signed Rank test: W=128, n_{s/r} = 16, z = 3.3, p = 0.0005).

The null hypothesis H₀4 is therefore rejected in favour of the alternative hypothesis, that participants experienced their self-efficacy in computer programming to be higher after the workshop than before the workshop, for this third group of participants.

6.7.6 Other Benefits

6.7.6.1 Enjoyment

Enjoyment of the Alice workshops, Mindstorms workshops and Photoshop activity was extremely high for this group of participants (median of 9 for all measures, where 9 = “strongly agree” with “I enjoyed the [Alice/Mindstorms/Photoshop] workshop”).

6.7.6.2 Interest

There was no significant difference in interest in Alice or Mindstorms programming from before the workshops to after the workshops (Wilcoxon Signed Rank test: Alice: $W=27$, $n_{s/r} = 11$, $z = 1.18$, $p = 0.119$; Mindstorms: $n_{s/r} < 10$ medians = 9(pre) and 9(post)). Interest stayed high throughout.

6.7.6.3 Participant Comments

This group of girls were extremely positive towards the day’s activities and this is reflected in their comments. When asked what they enjoyed most, girls identified working with Alice, working with the robots and “that it was only girls”:

- “Everything!! It was just THE best and funnest experience”;
- “The robots and that I learnt more about computers and technology”;
- “That it was only girls and the robots I think”;
- “That it was fun and educational”;
- “We got to do a good variety of things”.

The suggestions for how the organisers could improve the delivery again made mention of “more time” and “I don’t think there was anything you could have done better. I loved it the way it was”. General comments included “When is the next time?” and “Thank you for letting me join in, I had an awsome [sic] time thanks”.

6.7.7 Mental Effort

Day 3 participants were asked about the mental effort expended during the Alice and Mindstorms workshops, as detailed in Section 6.5.7. Medians and range for each measure/workshop (intrinsic, extraneous and germane in Alice and Mindstorms) are show in Table 6-6.

	Intrinsic	Extraneous	Germane
Alice programming	Median = 1.5 Min = 1, max = 9	Median = 3 Min = 1, max = 9	Median = 2.5 Min = 1, max = 9
Mindstorms programming	Median = 1.5 Min = 1, max = 9	Median = 2 Min = 1, max = 5	Median = 2 Min = 1, max = 5

Table 6-6 Measures of mental effort on Alice and Mindstorms - Day 3

By visual inspection, medians of measures in this group are lower than the previous two groups, however some individual participants still experienced comparatively high mental effort. There is no significant difference between intrinsic, extraneous and germane load expended on each of the activities (Friedman's 2-way Analysis of Variance test: Alice: $\text{csq}_r = 0.66$, $\text{df} = 2$, $p = 0.7189$; Mindstorms: $\text{csq}_r = 0.09$, $\text{df} = 2$, $p = 0.956$).

Although the reported mental effort for Alice activities appears on the surface to be higher than for the Mindstorms activities for this group of participants, the difference was not statistically significant.

For all three days, the difference between the mental effort for Alice and that reported for Mindstorms was not significant on all three measures, using Wilcoxon Signed-Rank test (see Table 6-7 below).

	Intrinsic	Extraneous	Germane
Day 1	$n_{s/r} < 10$	$n_{s/r} < 10$	$n_{s/r} < 10$
Day 2	$p = 0.1867$	$p = 0.3859$	$p = 0.2709$
Day 3	$p = 0.2148$	$p = 0.0885$	$p = 0.1112$

Table 6-7 p-values for differences between Alice & Mindstorms mental effort measures

Further comparisons between the days are considered in Section 6.8 below.

6.8 Comparison between IT Girls Days

This study effectively reports on three separate IT Girls Days. Different cohorts of participants were seen over the three days. While the most obvious difference between these three cohorts is the year in which students currently exist, the university staff noticed a large difference in apparent computer ability (literacy) between the Year 6 to 9 participants compared to the Year 10 and 11 participants. Year 10 and 11 participants often displayed difficulties doing basic computer tasks such as saving files on the desktop, finding files in order to open them and even logging on to their student account. In comparison, Year 6 to 9 participants needed little direction and help from staff for these basic tasks and so had more time to spend on the workshop activities themselves.

The difference between these cohorts is likely due at least in part to a government initiative to provide laptop computers to Year 9 students (Dept of Employment Education and Workplace Relations, 2011). As it happens, these particular Year 10 and 11 students had pre-dated this initiative, and so did not receive laptops. The Year 8 and below students had not yet started the year of schooling where these laptops would be provided, however they had had more access to updated

computer labs, also as part of this Government initiative. For this reason each day has been treated separately. Medians of reported measures on each of “Career Aspirations”, “Knowledge of IT Careers”, “Confidence in Programming” and “Difficulty of Programming” for each of the three days are reported in Table 6-8.

	Career Aspirations		Knowledge of IT Careers		Confidence in Programming		Difficulty of Programming	
	Pre	Post	Pre	Post	Pre	Post	Pre	Post
Day 1	1.5	5.5	2.5	6	1	6	6.5	3
	significant		Significant		Significant		significant	
Day 2	1.5	5.5	2	5.5	2.5	5	4.5	5.5
	Significant		Significant		significant		Not significant	
Day 3	3.5	4.5	1	5.5	3.5	9	3.5	1.5
	Significant		Significant		Significant		Not significant	

Table 6-8 Medians of 9-Point Likert test questions in pre- and post-questionnaires

6.8.1 Career Aspirations

It was hypothesised that there would be a positive shift in career aspirations as a result of the IT Girls Day, and this was true for each of the three groups of girls. Interestingly, the median score for this measure before the workshop for the third group of girls (in Years 6 and 7) was 3.5 before the workshop, in comparison to a median of 1.5 for the first and second day. However when Day 1 and Day 3, and Day 1 and Day 2 were compared, using Mann-Whitney U tests, the difference was not significant (Day1/Day 3: $U_A = 143.5$, $z = -1.29$, $p = 0.0985$; Day 2/Day 3: $U_A = 160$, $z = -1.19$, $p = 0.117$). Note that the Mann-Whitney U test has been used throughout this thesis wherever a non-parametric alternative to a t-test is required.

6.8.2 Knowledge of IT Careers

On all three days the girls reported an increase in their knowledge of IT careers – one of the purposes of the program. The three days were not compared statistically for this measure as by inspection the increases in the measures were similar.

6.8.3 Confidence in Programming

On all three days the girls experienced a significant increase in self-efficacy regarding programming. The median scores pre-workshop appear to trend upwards over the three days, and there was found to be a significant difference in the pre-workshop measure of “confidence with programming” from Day 1 to Day 3 (Mann-Whitney U test: $U_A = 165.5$, $z = -2.2$, $p = 0.0139$). In a relatively small school (and community) such as Lightning Ridge, it is possible that older siblings and friends reported their enjoyment and success with programming activities to younger children, who may have then been comparatively more positive towards the activities before starting their IT Girls Day. An alternative explanation is that the younger cohort (Day 3) had more exposure to computers generally, and this may have impacted on their initial self-efficacy regarding programming.

6.8.4 Difficulty of Programming

This “flow-on” effect from older siblings and friends may also explain the differences in results for “perceived difficulty of programming” between the three days. Participants on Day 1 experienced a shift in their perception of the difficulty of programming in the direction of “easier” over the course of the day. However

participants on Day 2 and Day 3 did not experience this shift in perception, despite their positive comments, increase in self-efficacy, and enjoyment of the day.

The difference in pre-program measures on Day 1 and Day 2 were tested, and found to be not statistically significant (Mann-Whitney U test: $U_A = 97$, $z = 0.6$, $p = 0.2743$). In contrast, the difference in pre-program measures between Day 1 and Day 3 were statistically different (Mann-Whitney U test: $U_A = 50$, $z = 2.56$, $p = 0.0052$). Again, this could have been a function of communications between students at the school about the perceived difficulty of the workshops, or could be an artifact of the nature of the differences between the cohorts used across the three days with respect to their exposure to computers within the formalised school environment.

6.8.5 Mental effort measures

Participants on each of the three days experienced no significantly different levels of mental effort on each of the three aspects of cognitive load – intrinsic, extraneous and germane.

For each of these dimensions of cognitive load in each of the activities (Alice and Mindstorms), the mental effort experienced by the groups of participants were compared using a Mann-Whitney U test. There were no significant differences between days for each of these mental effort measures, for Alice or Mindstorms.

Of interest is the wide range in mental effort reported by individuals on all three days for all cognitive load aspects and for both Alice and Mindstorms activities. This indicates that there is a wide variety of capabilities with programming amongst novices, and that some of these individuals will experience high levels of mental effort even while others are experiencing little to none whilst doing the same activities.

6.9 Gender bias results (Hypothesis 5)

H_A5 : Participants will be more positive towards females as programmers after the workshop than before the workshop.

H_05 : (Null Hypothesis) Participants will not be more positive towards females as programmers after the workshop than before the workshop.

Analysis

Participants on each day were randomly divided into two groups, one which completed the Alice activities first, and the other that completed the Mindstorms and Photoshop workshops first. Each of these groups was randomly divided into 2 subgroups. The first of the subgroups was given Questionnaire 1A, and the second given Questionnaire 1B. The questionnaires were identical with the exception of the wording of the gender question. Questionnaire 1A asked participants to indicate how much they agreed or disagreed with the statement “Generally, girls are better than boys at programming” and Questionnaire 1B reversed this statement with “Generally, boys are better than girls at programming”. Participants answered this question on a 9-point Likert scale with 1 = “strongly disagree” and 9 = “strongly agree”. Each of these subgroups were given the identical question on the post-workshop questionnaire, to determine if the participants’ opinions had changed as a result of the workshop.

As the numbers in each subgroup were small, the 3 days in the program have been combined into two groups A (22 participants) and B (24 participants) for analysis.

Within Group A (“Generally, boys are better than girls at programming”), there was no significant difference between pre-workshop and post-workshop scores

(Wilcoxon Signed Rank Test: $W=28$, $n_{s/r} = 14$, $z = 0.86$, $p = 0.1949$). Within Group B (“Generally, girls are better than boys at programming”), there was no significant difference between pre-workshop and post-workshop scores (Wilcoxon Signed Rank test: $W=15$, $n_{s/r} = 11$, $z = 0.64$, $p = 0.2611$).

To test the equivalence of the reversal of the question, Group B scores were reversed so that $1 = 9$, $2 = 8$ and so on. The two pre-workshop datasets were then compared using Mann-Whitney U tests and were found to be significantly different ($U_A = 345$, $z = -1.77$, $p = 0.0384$) at $p < 0.05$ level. The order of the wording of this question has had an effect on the participants’ answers regarding gender.

The hypothesis to be tested regarding gender was “Participants will be more positive towards females as programmers after the workshop than before the workshop.” In hindsight, the Likert scale questions about comparative merits of boys and girls as programmers did not address this hypothesis. A better question would have asked the participants to agree/disagree with the statement “Girls can be good programmers” before and after the workshop activities, and compared the two results. Nonetheless this question about gender has raised some interesting queries about order of wording of gender comparison questions, and could be used as the basis for further study.

In this case, the null hypothesis cannot be rejected in favour of the alternate hypothesis.

6.10 Further Discussion

6.10.1 Success of the program

This chapter has outlined a schools outreach program intended to raise the aspirations of female high school students to pursue a career in IT. This objective has

clearly been achieved with the overall number of girls indicating “I would like to have a career in IT” shifting significantly between the start of the day to completion of the day for all three cohorts.

6.10.2 Differences with previous program

This program was based upon a successful program (described in Chapter 5) delivered to female school students on a university campus, which positively impacted the participants’ confidence with programming and reduced their perceptions of the difficulty of programming. The study described in this chapter demonstrates that effective programs can also be delivered to students within their local school context, even when students are not self-selected.

The school which participated in this study is considered to be geographically ‘remote’. Schools in regional and remote areas of Australia are recognised to have less access to university campus visits, simply by definition of their geographical location. This study demonstrates that visits by universities to schools can provide effective interventions to alleviate some of the negative effects of geographic isolation.

Another factor considered to be highly important within the current study is with regards to the profiles of the students attending the IT Girls Day Outreach Program. These were not self-selected, as has occurred in the past with students needing to go to the effort of visiting the university campus. This time the program was available to all female students within, and as part of, their normal school day. All of the female students attending school on those days, from Years 7 to 11, participated in the program. All of the Year 6 students would have participated but numbers were limited by the number of spaces available in each day’s program.

Consequently the cohort was not “pre-loaded” with an interest or bias towards studying IT. The author was surprised by the high number of participants who displayed not only a positive shift in aspirations, but delivered high values in their post-workshop aspiration values. Specifically, of the 46 girls completing a day's activities, there were just 2 participants who, in the pre-workshop questionnaire, rated their interest in pursuing a career in IT at 6 or more on the 9 point Likert scale. This figure shifted to a total of 21 out of 46 girls rating at 6 or more in the post-workshop questionnaire.

6.10.3 Why did this work?

Considering that the focus of this program was primarily programming workshops, and that programming is considered to be difficult, frustrating and in general not enjoyable for novices, why did this program have such positive results? In designing, developing and delivering the program, care had been taken to:

- ❖ Select suitably interesting and simple programming environments;
- ❖ Design and develop clear, complete, direct instructions regarding the activities to be undertaken;
- ❖ Present short, simple, self-contained worked examples;
- ❖ Ensure a balance between theory and practical hands-on activities;
- ❖ Ensure that corrective feedback was given regarding performance, often via the software in how it functioned in response to the coding;
- ❖ Present the activities in an environment that was comfortable socially, with students working in small groups with friends;
- ❖ Have all activities presented by female staff;

- ❖ Ensure, as far as possible, that students had fun and could perceive that they had achieved (Photoshop and Alice presented images and environments on-screen that students had built, and the Lego robots behaved as students had programmed).

The reasons for success of the current study may fall into any one of these areas, a combination of interactions between them all, or just a subset of them.

The results of the current study go far beyond mere aspirational changes. All three groups indicated significant shifts upwards both in their knowledge of career prospects in IT and in their confidence to program. All three groups also displayed trends downwards in perceived difficulty of programming with one of the three groups returning significant differences.

The author is of the view that the success of the outreach program lies primarily in the effectiveness of the teaching and learning materials and activities, based on cognitive load theory principles, through which students were able to experience first-hand success in programming environments, without experiencing high levels of cognitive load (in most cases). This, however, remains to be tested.

Moreover, even if the success of the program does lie in the cognitive load theory-based instructional design of the materials and activities, it remains to be tested what, specifically, drove these effects. This defines an area for further studies. After all, computer programming has long been considered to be an area of relatively high complexity and relatively slow rates of obtaining mastery (McCracken et al., 2001). Yet, in the current study, high school students with little previous exposure to programming are reporting high levels of programming confidence, in less than a day's activities. The program reported in the current study clearly delivers in the area of teaching and learning computer programming.

Further study was needed to explore the differences between the positive experience reported by the school students (even those who did not self-select to participate in the activities) and that reported in the literature regarding university students' experiences. To this end, a survey was conducted of instructors in introductory programming courses offered by Australian Universities to determine the types of instructional activities given to tertiary students studying programming. The results of this survey are reported in Chapter 7.

Chapter Seven: Australian Universities Survey

7.1 Introduction

The local IT Girls Day program and the remote version of the program were successful in introducing the basics of programming to novices with participants reporting high enjoyment, positive changes in self-efficacy and reduced perception of the difficulty of programming. This was achieved using simplified teaching environments (Alice and Lego Mindstorms) and carefully designed, sequenced and paced instructional materials.

There has been much debate in the academic community about what languages, environments and paradigms should be used for university students' first exposure to programming (Bruce, 2004; Cooper et al., 2003; Decker and Hirshfield, 1994; Kelleher and Pausch, 2005)

Comparing the positive results of the IT Girls Day experiences with the many reports about the difficulty of teaching novices about computer programming in a university environment, the author decided to survey the instructors of introductory computer programming courses in Australian universities, to ascertain what languages and environments were currently being used, whether teaching languages and/or environments were being used, and the reasons for the choice of these languages and environments. Of particular interest were the university instructors' opinions about how Australian students were coping with introductory programming courses. Given the wide range in mental effort while completing introductory programming tasks reported by the remote students in Chapter 6, questions were also to be asked about

the level of mental effort reported to be used by these university students.

Additionally, for the purpose of enabling comparisons based upon level of expertise, it was decided that the mental effort, as experienced by these instructors while teaching introductory programming, was also to be collected.

7.2 Background

7.2.1 Previous Censuses

In 2001 a census of introductory programming courses at Australian Universities was conducted by a researcher from a Southern Queensland university (De Raadt et al., 2002). This census reported on the languages and environments/tools being used, the reasons for the choice of language, student numbers and the paradigm being taught. The census covered 57 courses at 37 of the 39 Australian universities. The other two universities did not offer programming courses.

This census was repeated in 2003 (De Raadt et al., 2004) and expanded to include New Zealand universities. The 2003 census examined trends in languages and reasons for language choice, paradigms taught, tools and environments used, as well as new questions on text employed, method of delivery to on-campus students, instructor experience and information about the teaching of problem solving strategies.

The censuses gave a picture of declining student numbers in programming courses, languages that were, on the whole, chosen for industry relevance rather than pedagogical reasons, and instructors who were loath to use an IDE for teaching in many cases. More detailed results of these two censuses have been reported and compared to the findings of the current study, in the results and discussion sections of this chapter.

7.2.2 The 2010 Study

In the latter months of 2010 the census was to be repeated with all Australian universities. Participation was not as high as had been hoped, with a participation rate of 28 universities from the 39 across Australia that offered programming courses. A total of 44 out of 73 available programming courses were covered. While no longer a census, this study contains a large sample of the available first programming courses offered. The sample size was large enough to make meaningful observations on longitudinal trends in language, tools and paradigms and to identify reasons for any such changes over the 10 year period. The basis for constructing interview questions and for conducting the study are described in the next section “Methodology”, followed by an in-depth discussion of the results and the implications for teaching introductory programming.

7.3 Methodology

7.3.1 Identifying participants

The previously collated list of participants from the 2001 and 2003 studies was used as a starting point for building a list of contacts for the current study. Information from university websites was also used to identify potential university programs and to collect contact numbers of administrative staff or academic staff responsible for those programs.

Once identified, each academic responsible for a particular introductory programming unit was sent an introductory email outlining the past two censuses. Participants were then contacted by phone within seven days to invite participation in

the new census, and to arrange a convenient time for a phone interview of around 10 to 15 minutes duration.

All phone interviews were audio-recorded with the participant's permission. Notes on responses and comments were also entered manually into paper-based census forms as a backup to the recordings. Audio recordings were later transcribed and the data analysed.

7.3.2 Terminology

It was found that the terminology used for a unit of study that is completed by students towards a degree varies between institutions, for example "subject", "course" or "unit" were used. The interviewer used the terminology particular to each institution when conducting an interview to reduce possible confusion from ambiguity. In the remainder of this chapter, the results are reported using the description "course" for the basic "block" of study (usually studied over the period of a semester or session, in conjunction with other courses of study), to be consistent with the last two censuses.

7.3.3 2001 and 2003 Census Questions

The emphasis of the 2001 census was regarding "the teaching of non-commercial languages, as opposed to teaching wide-spread commercial languages, in first programming courses" (De Raadt, 2001). The census document (see Appendix C.1) asked questions about the number of students undertaking a course, for what type of students (computer science/IT, Engineering, Business or Other) the course was designed, the language used, why this language was chosen, whether there was plans to change the first language, what environment and/or tools were used to support

teaching of the language, what paradigm was being taught, and what languages were taught in the rest of the degree program.

The second census, in 2003, repeated several of these questions, and added others (see Appendix C.2). The questions about type of students and what languages were taught in the rest of the degree were removed, and questions were added on change of language between 2001 and 2003, textbook(s) used in the course, amount of time on-campus students spent in lectures, tutorials and practicals, instructor teaching experience, and a focus area on the teaching of problem solving.

7.3.4 Changes in the 2010 Study

The 2010 study (see Appendix C.3) repeated questions from the 2001 and 2003 censuses which probed language and paradigm choice, duration of lectures, tutorials and practical lessons, instructor experience, textbook use, problem solving strategies and use of development tools. Although some of these questions were not as relevant as others to the topic of this thesis, they were included for the sake of continuity, for future research. The author consulted with Michael de Raadt, the first author of both the 2001 and 2003 censuses, on the structure of the 2010 study, and added questions on external/distance study availability and assessment delivery on his advice. It was decided to leave the questions regarding the 2003 census's focus area (problem solving) in the 2010 study.

The orbit of enquiry of these two censuses was expanded to also include a focus area regarding the mental effort (Paas et al., 2003b) expended by instructors and students in understanding and learning aspects of programming using the language(s) used in each course, and comparative difficulty of various languages and environments. On this basis, the following question was asked:

- ❖ “How difficult do you think this language is for students to learn?” (9 point Likert scale, 1 = ‘extremely easy’, 9 = ‘extremely difficult’).

In the 2003 census, instructors were asked for their reason(s) for choosing a particular language. Results of this question, repeated in the 2010 study, are given in Section 7.6.4. Participants were not asked for their reasons for choice of environment/tools, or how those environments and/or tools were used, in the 2003 census. In the 2010 study, if an interviewee indicated that he or she used an environment or tool beyond a simple editor and command line compiler, the following questions were asked:

- ❖ “Why was this environment/tool chosen?”;
- ❖ “Is this environment/tool used for an initial part of the programming course only, or throughout the first programming course?”;
- ❖ “Is the environment/tool used in any other courses in the degree? If so, how many? Is it used in a different way in subsequent courses?”;
- ❖ “How difficult do *you* find the environment to use (on a scale of 1 to 9 where 1 = ‘extremely easy’ and 9 = ‘extremely difficult’)?”;
- ❖ “On average how difficult do you believe *the students* find the environment to use?” (9 point Likert scale, as above).

A section was added to the study questionnaire, examining the mental effort expended by instructors and students in each of the three areas previously devised in the IT Girls Day studies:

- ❖ Understanding and processing the problem statement;
- ❖ Navigating or using the environment, tools or language; and
- ❖ Learning from the problem and reinforcing previous concepts.

All participants were asked about the level of mental effort expended when solving a novice programming problem on each of these three measures, for themselves and for an ‘average student’ in their courses. This mental effort should be lower for

instructors than students, given that the students are novices and the instructors are experts (Chase and Simon, 1973). As a result of the reported low success rate of introductory programming courses in the literature, instructors were also asked about to assess the level of mental effort expended by students in the bottom 10% of their course. The 10% figure was arbitrarily chosen to capture a snapshot of these low-performing, often failing, students, although the actual amount of failing students may have been higher.

The results of the 2010 study are reported below, with comparison to the previous two censuses. For more accurate comparison, only the data from Australian universities in the 2003 census has been used - data from New Zealand universities has been excluded.

7.4 Results - Participation rate

The 2010 study covered 44 of the 73 programming courses offered by Australian universities. A total of 28 of the 39 universities offering programming courses participated in the study (see Table 7-1). Note that the number of introductory programming courses in 2010 was 73, but only 44 courses (the number in brackets in the below figure) participated in the study.

	2001	2003	2010
Australian Universities	39	40	40
Universities teaching programming	37	39	39
Introductory Programming courses	57	71	(44) 73
Total students in study (approx.)	19900	16300	7743
Average students per course	349	229	176

Table 7-1 University/Course Summary

Although the total number of programming courses offered has changed little since the 2003 census, instructors stated that often business-based IT programming courses, computing science programming courses and engineering programming courses had been amalgamated into one course. The reasons for this offered by participants in several institutions were that management had strategically decided to merge different courses and associated student cohorts to gain efficiencies of economies of scale. These actions had been in response to declining numbers of students. At the same time, programming courses have appeared in non-traditional programming areas such as visual arts, keeping the number of introductory programming courses relatively static.

Three instructors declined to participate. Several instructors were employed casually and were not available on campus at any time apart from their face-to-face teaching commitments, two instructors had retired recently and were not available for comment, and some instructors (and administrative staff) were unavailable during the 3 month period of conducting interviews for this study. All but one of the participants agreed to the audio-recording of the interview. This participant agreed to be interviewed with hand-written notes being taken on paper-based census forms.

7.5 Results – Student numbers

The average number of students/course continued to fall over the period 2003 to 2010. The rate of declining number of students studying programming is substantial. Average numbers of enrolments per course have halved, falling from 349 in 2001 to just 176 in 2010. This follows a general trend in declining student enrolments in all areas and levels of ICT education - vocational, undergraduate and

postgraduate - with 17436 total domestic ICT enrolments in 2001 falling to just 7470 domestic students in 2008 (Australian Computer Society, 2010).

This is despite strong growth prospects for employment in the IT industry, with 7 out of 11 ICT job designations in the “Job Prospects Matrix” (DEEWR, 2011) considered to have ‘strong’ to ‘very strong’ growth prospects for future employment. The widening gap between increasing demand and declining domestic student numbers may give rise to a significant short-fall of suitably educated and skilled IT professionals in the near future.

7.6 Results – Languages

7.6.1 How many languages?

There were a total of 20 different languages being taught by the academic staff interviewed, which is a greater diversity of languages than displayed in each of the last two studies. During 2001, only nine languages were recorded, and in 2003 this number had reduced to eight languages. In 2010, the number of languages taught over the duration of a course ranged from 1 to 6 (see Table 7-2), with the vast majority teaching just one language.

No. of languages	1	2	3 to 6
Courses	37	4	3

Table 7-2 Number of languages in a course

7.6.2 Primary languages

Some languages were only used for a short time during a course, with another language being used for most of the course. For example, in one case Alice was used for the first two weeks, followed by Java. In this case, Java could be designated as the “primary” language for the course.

Language	Courses	%age	Weighted by students
Java	16	36.4%	38.4%
Python	6	13.6%	19.2%
C	5	11.4%	11.7%
C#	4	9.1%	8.0%
Visual Basic	4	9.1%	5.1%
C++	3	6.8%	4.8%
Processing	2	4.5%	5.2%
Alice	1	2.3%	0.9%
Fortran	1	2.3%	3.9%
Javascript	1	2.3%	1.5%
Matlab	1	2.3%	1.3%

Table 7-3 Primary Languages in 2010

If only these ‘primary’ languages are counted, 12 languages were used by instructors.

One of these languages is "MaSH", a language created as a staged subset approach to Java (Rock, 2011). For the purposes of this study, this language has been counted as a variant of Java.

The number and percentage of courses using each primary language is shown in Table 7-3 below. As some courses have very few students and others have hundreds, as well as the raw number of courses using each language, a weighted percentage by students has been calculated and is also displayed in Table 7-3.

7.6.3 Top languages

The set of the top three languages has changed since the 2003 census. Java continued to hold the place of the most popular language. C++ and Visual Basic have decreased in popularity and fallen out of the top three. Python, under development since 1990, was the second most popular language in 2010, followed by C. According

to the TIOBE³ programming language index (TIOBE Software, 2011), industry use of Python had the greatest growth in popularity of any language in the years of 2007 and 2010. The relatively new language C# is the fourth most popular language, seemingly replacing Visual Basic, which has dropped to just 5.1% of student capture. This apparent replacement is not surprising, as both Visual Basic and C# are Microsoft products and C# is considered to be more modern and popular than Visual Basic (TIOBE Software, 2011).

Language	2001	2003	2010
Java	40.4%	40.8%	36.4%
Python	0.0%	0.0%	13.6%
C	7.0%	12.7%	11.4%
C#	0.0%	0.0%	9.1%
VB	24.6%	26.8%	9.1%
C++	14.0%	11.3%	7.0%
Processing	0.0%	0.0%	4.5%
Fortran	0.0%	1.4%	2.3%
Javascript	0.0%	0.0%	2.3%
Matlab	0.0%	1.4%	2.3%
Alice	0.0%	0.0%	2.3%
Haskell	5.3%	4.2%	0.0%
Eiffel	3.5%	1.4%	0.0%
Delphi	1.8%	0.0%	0.0%
Ada	1.8%	0.0%	0.0%
jBase	1.8%	0.0%	0.0%

Table 7-4 Language comparison by course numbers

³ TIOBE Software is a Coding Standards Company. TIOBE stands for “The Importance Of Being Ernest” (TIOBE Software, 2012).

A comparison of the language use across the 2001, 2003 and 2010 studies is provided in descending order by percentage of courses (Table 7-4) and by student numbers (Table 7-5).

Figure 7-1 and Figure 7-2 chart the changes in popularity of the top 4 languages in each of these three studies (Java, VB, C++ and Haskell in 2001, Java, VB, C++ and C in 2003, and Java, Python, C and C# in 2010), by percentage of courses offering the language (Figure 7-1), and then weighted by students (Figure 7-2).

Language	2001	2003	2010
Java	43.9%	44.4%	39.0%
Python	0.0%	0.0%	19.5%
C	5.5%	10.6%	11.9%
C#	0.0%	0.0%	8.2%
Processing	0.0%	0.0%	5.3%
VB	18.9%	16.4%	5.2%
C++	15.2%	18.7%	4.9%
Fortran	0.0%	0.7%	3.9%
Javascript	0.0%	0.0%	1.5%
Matlab	0.0%	1.0%	1.3%
Alice	0.0%	0.0%	0.9%
Haskell	8.8%	6.0%	0.0%
Eiffel	3.3%	2.1%	0.0%
Delphi	2.0%	0.0%	0.0%
Ada	1.7%	0.0%	0.0%
jBase	0.8%	0.0%	0.0%

Table 7-5 Language comparison by student numbers

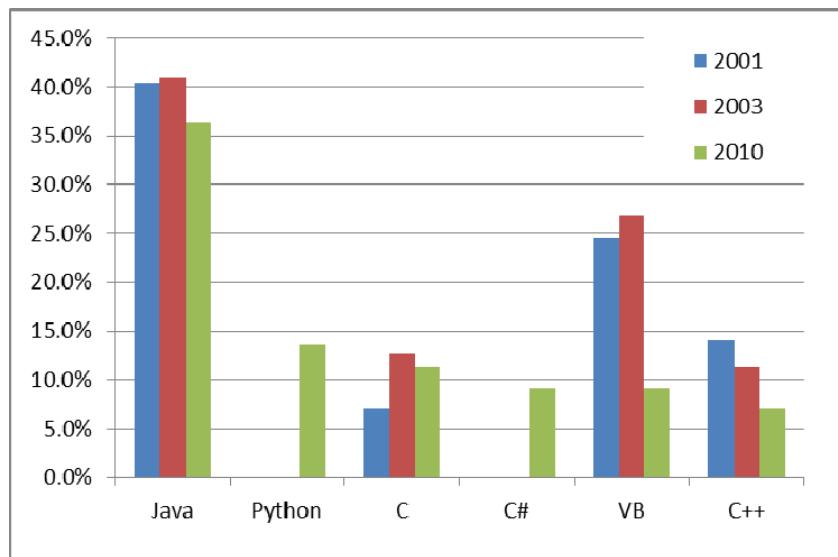


Figure 7-1 Top 4 languages of each study by number of courses

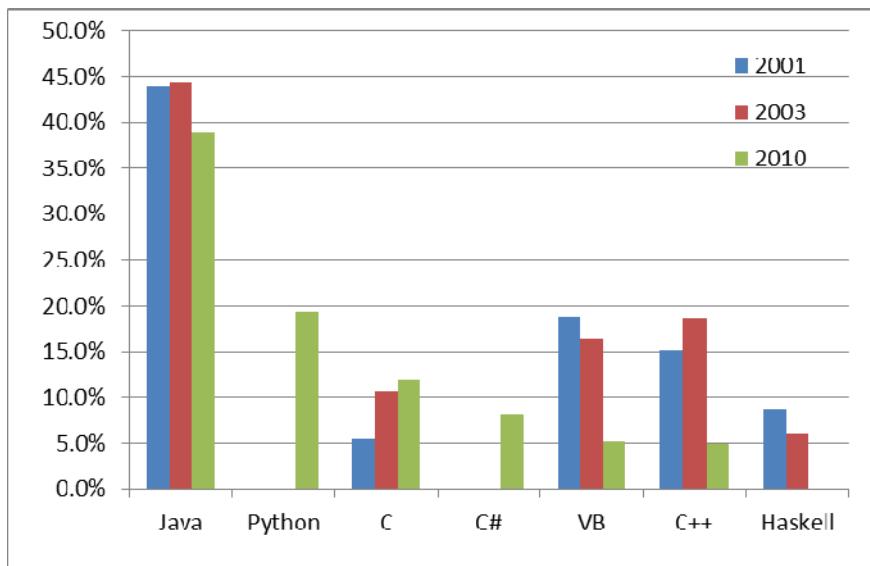


Figure 7-2 Top 4 languages of each study by student numbers

Java continues to dominate, while Python and C# have appeared on the charts and captured a significant percentage of language share. More interesting to this study than the choice of actual language is the reason for language choice, by instructors. This author was particularly interested in the number of instructors choosing a particular language for pedagogical reasons, rather than for industry relevance or other reasons.

7.6.4 Reasons for language choice

Instructors were asked in the 2001 census and the 2010 study about the reasons for their choice of language. More than one reason could be offered.

In 2001 the most commonly provided reason for choosing a language was industry relevance and/or marketability to students (with 56.1% of participants identifying this as a reason). The second most commonly provided reason for choosing a language was “pedagogical benefits”, with one third (33.3%) of the instructors presenting this as a reason for language choice. The results of the 2010 study presented a substantial shift in the frequency of these reasons being given for language choice, with industry relevance and marketability declining (to 48.8%) and pedagogical benefit rising (to 53.5%).

Reason	2001	2010
Used in industry / marketable	56.1%	48.8%
Pedagogical benefits of language	33.3%	53.5%
Structure of degree/dept politics	26.3%	32.6%
OO language	26.3%	16.3%
GUI interface	10.5%	7.0%
Availability/Cost to students	8.8%	4.7%
Easy to find appropriate texts	3.5%	2.3%
OS/Machine limitations of dept	1.8%	4.7%
Online community and help available	0%	9.3%
Platform independence	0%	9.3%
Extensions/Libraries available	0%	7.0%
Interpreted language	0%	4.7%
Ease of installation	0%	2.3%

Table 7-6 Reasons for language choice

The 2010 survey also identified the emergence of several new reasons. Instructors mentioned “the availability of a community and online help”, “extendibility and libraries available”, “platform independence” (as opposed to “limitations of

OS/machines” as in the 2001 census), “ease of installation”, and “interpreted language” (with no need to compile). Some of these reasons reflect the rise of the open source community over the time period, as well as perhaps a greater choice of operating systems by students. Table 7-6 shows the change in percentages of instructors’ reasons for language choice between 2001 and 2010, ordered by frequency in 2001. Figure 7-3 shows the comparative frequencies of reasons given in the 2001 census with the 2010 results, ordered by the 2010 results.

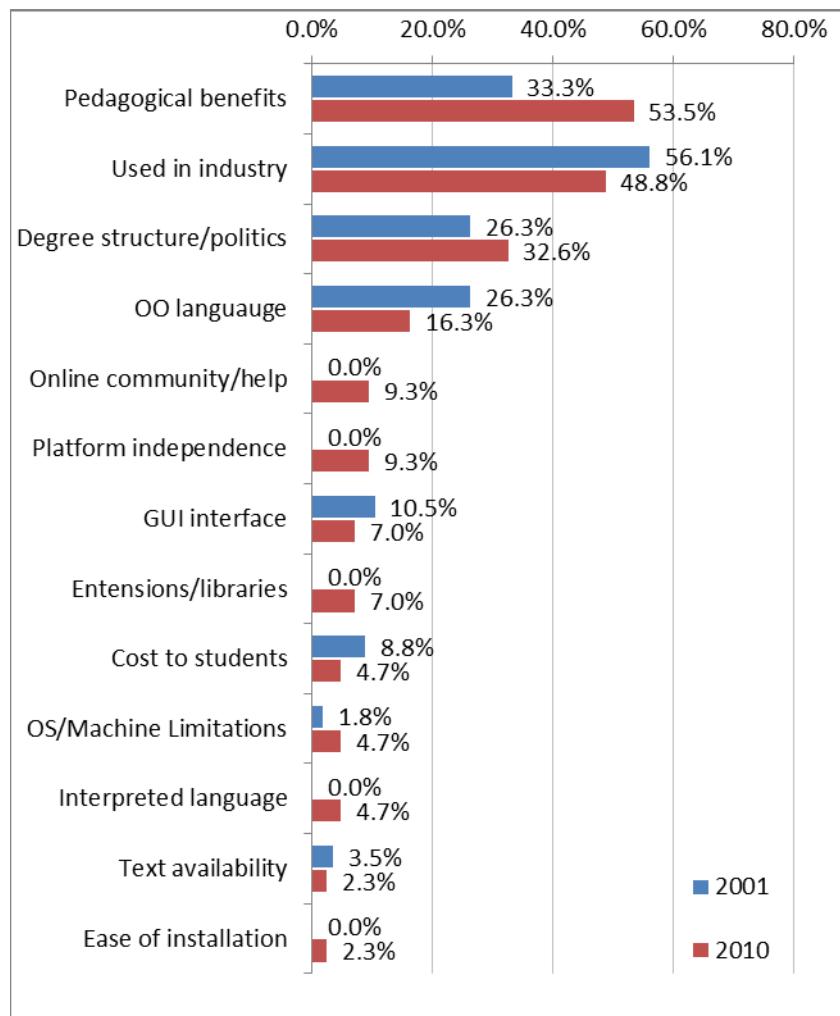


Figure 7-3 Trends in reasons for language choice

Figure 7-3 clearly shows that although “industry relevance and marketability to students” is still seen as important, the reasons for choosing a language have shifted to

be more strongly inclusive of pedagogical factors associated with both ease of learning and availability of support.

The intersection of industry relevance and pedagogy can be seen in the reasons given by those instructors who chose Python as their teaching language. As previously noted, Python is one of the more popular languages according to the TIOBE Programming Community Index (TIOBE Software, 2011), which indicates the popularity of programming languages in industry (numbers of skilled engineers world-wide and third-party vendors) and in training courses. However *all* of the instructors who chose Python commented that the reason they chose the language was because it was perceived as easier for students to learn. Only one instructor gave ‘industry relevance’ as a secondary reason for the choice of Python, and this was due to its association with the Google Apps engine. The comment was also made by one of these instructors that “we have to cater for prep [sic] students and keep IT students happy, so we have to find a balance between these”.

“Structure of degree/department politics” was the only other reason given in 2001 that increased in frequency in the 2010 study. As previously mentioned, in several cases two or more introductory programming courses in various disciplines such as engineering, business and computer science had been merged into one course. The language that was chosen for this one course became either a legacy language from one of the previous courses, or a language chosen to try to cater to a broader profile of students pushed into a narrower stream of programming teaching. Several instructors expressed frustration at the need to cater to a wide range of students with differing backgrounds, experience and capabilities. Typical comments included “We see students with a range of skills - from no experience to some with some computing in high school” and “IT students are not the same as CS students”.

An increasing number of instructors are clearly searching for ways to make learning programming “easier” and more effective for novices, and the reasons for language choice reflect this shift. The next section of the survey asked about environment choice, and reasons for the choice of environment. Were these environments chosen for pedagogical reasons, or because they were used commercially?

7.7 Results – Environments

7.7.1 *Choice of IDE/tools*

Some languages (including Alice and Processing) require the use of a specific environment, so there is no choice of environment involved in these cases. Instructors who did have a choice of environments or tools chose a wide variety or none at all. Microsoft Visual Studio was the most popular IDE. The use of the teaching environment BlueJ continued to increase, from 4% in 2001 to 17.5% in 2010. Within the ‘other’ category, note should be made of the "IDLE" IDE (for use with Python), at 12.5% and Eclipse, used by 7.5% of instructors.

The largest change has been the movement away from using text editors and command line compilers only - from approximately 45% of these instructors using no IDE/tool in 2001 and 2003, to just 20% in 2010. Figure 7-4 shows the trends in environment use over the time period of the three studies.

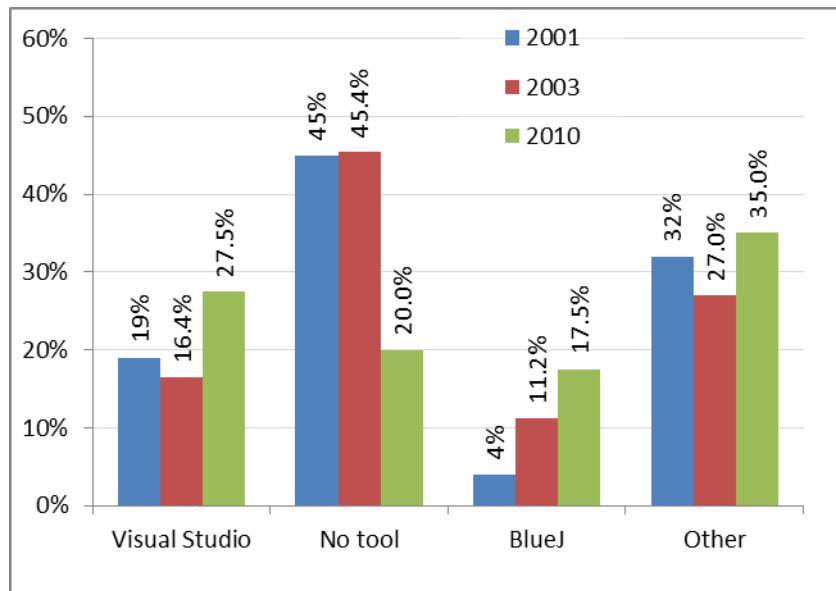


Figure 7-4 Trends in environment use

Only two instructors reported using a specialised learning environment (such as Alice or BlueJ) for an initial part of the course, followed by a more industry-standard IDE such as Visual Studio or Eclipse. Others used a learning environment throughout the first programming course.

7.7.2 Reasons for choice of environment

At the time of the 2003 census, the majority of instructors were choosing to *not* use an IDE, if they were not forced to do so by their choice of language. Although the specific question about reasons for choice of environment was not asked in that census, anecdotally this was due to the perceived overhead of instructing students on that tool compared to using a simple editor and command line compiler. During the 2010 study, instructors were directly asked about the reason or reasons that they had chosen a particular IDE or tool. The top 10 reasons for choosing an environment/tool (excluding those where the language is an integrated part of the environment, such as Alice and Processing) is given in Table 7-7. The “packaged with language” reason

includes those languages that include an IDE in the downloadable package, but which can be used with other IDEs. For example, Python includes the IDLE IDE but plugins for Python are available in Eclipse and other IDEs, so it is not tied to one environment in the same way as Alice or Processing.

Reason	Count	% courses
Pedagogical reasons	15	41.7%
Packaged with language	14	38.9%
Cost to students	13	36.1%
OS limitations of dept.	7	19.4%
Uncomplicated/Ease of use	6	16.7%
Industry relevance	4	11.1%
Supports OO paradigm	4	11.1%
Visual cues/Visual debugger	4	11.1%
Student motivation	3	8.3%
Open source	3	8.3%

Table 7-7 Top 10 reasons for environment choice

Other reasons included “associated text is very good”, “cross-platform”, “plugins available”, “ease of installation” and “GUI”. Some instructors offered reasons as to why they did not choose to use an IDE, even though this was not explicitly asked as part of the study. These included “students need to become familiar with the command line” and “we don’t have time for that”.

The most frequent reason given for choice of environment (provided by 41.7% of instructors) was pedagogically-based. This is consistent with the reasons provided by participants for choice of language, where pedagogical reasons was also identified as the most frequent reason for choice (of language).

Associated comments offered by instructors emphasised that some environments assisted learning by allowing students to concentrate on the concepts of programming, rather than the specifics and nuances of a language:

- ❖ “can think about objects instead of thinking about the language itself”;
- ❖ “[an] introduction without the stress of having to worry about syntax rules. It reinforces what happens when you use a loop so the frog jumps once or it just keeps jumping. You have to understand the concept, so its concept reinforcement.”;
- ❖ “We found in the past that students were having trouble with understanding what all the features of the main method are, and some of the initial concepts we had to abstract away – we had to say ‘treat this as magic, you have to do it’”;
- ❖ “mostly to encourage students who have no programming background, and strengthens the concepts of loops, and iterations and all those things”.

Although these instructors have not commented on the concept of extraneous cognitive load directly, they have described the advantages of reducing the amount of elements students have to learn which are not directly related to the topic being taught - such as syntax - while learning basic concepts such as objects and code structures (sequence, iteration and selection).

Some instructors stressed that the environment they had chosen was simple whilst still including tools to reduce the complexity of compiling and building:

- ❖ “it’s really a smarter text editor with buttons for compiling and building. Nothing like Visual Studio or Eclipse. ...so we can concentrate on language syntax”;
- ❖ “To reduce the amount that students had to learn in order to get to the heart of programming”;
- ❖ “just its simplicity”;
- ❖ “simple, not confusing”.

These comments echo the previous comments about reducing complexity, from a different angle. If learning the language syntax is the objective of the instruction, then the necessity of using a separate process for compiling and building causes extraneous cognitive load due to the split attention effect. In this case, using a simple button to compile and build reduces the mental effort required and frees cognitive resources to apply to the germane task of learning the syntax.

In contrast, other instructors pointed at an IDE's perceived intrinsic superiority to simple editors/command line compilers, due to the inclusion of helpful tools:

- ❖ “why get them to travel by horse when they can travel by car?”;
- ❖ “the tools that it provides for students are wonderful compared to using just a basic text editor - they offer nothing”.

There has been an apparent shift from viewing the use of an IDE as an additional overhead, to seeing it as helping to reduce the amount a student has to learn, or as a tool to help the student to learn. Whether this viewpoint is always correct is debatable, and may be a function of what is being taught (central concepts to programming in general, or a language specific syntax), the student’s ability and experience, and aspects of the language or environment itself.

7.8 Results - Other teaching aspects

7.8.1 Paradigm taught

In common with the 2001 and 2003 censuses, the 2010 study showed that most instructors choose to teach using a procedural paradigm. Some instructors - 8 from the 44 interviewed - also reported that they taught primarily procedurally but introduced some object-oriented concepts. As an example, they may have mentioned objects or used the other terminology of object-oriented programming to prepare

students for further courses that covered OO programming. Some used objects but “in a procedural way”. For the purposes of comparison and consistency with previous studies, these have been designated under the procedural paradigm.

Some instructors described how they used either a mix of paradigms - for example they started with procedural and then used the last 6 weeks to cover OO concepts - or suggested that they taught more than one language and taught each language in different ways. The number of courses teaching in each paradigm is given in Table 7-8.

Paradigm	Courses	%age
Procedural	24	54.5%
Object-Oriented	11	25.0%
Mixture	8	18.2%
Functional	1	2.3%

Table 7-8 Paradigms taught

Longitudinal trends in paradigms taught are shown in Figure 7-5. The downwards trend in the numbers of instructors teaching objects-first is clearly shown and the use of the functional paradigm has nearly disappeared.

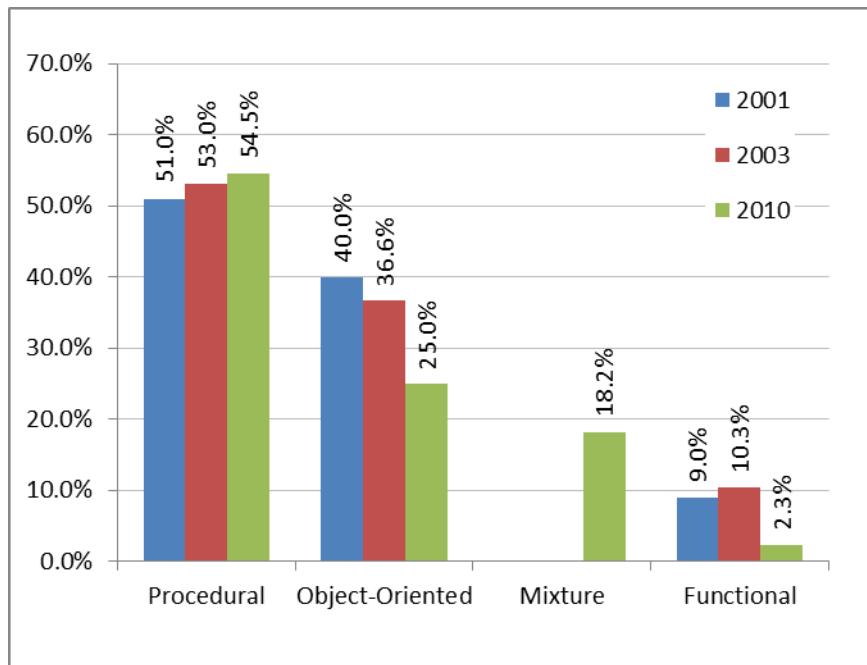


Figure 7-5 Trends in paradigms taught

7.8.2 On-campus hours

Instructors were asked about the time that on-campus students spent in lectures, tutorials and practicals each week. Although the average hours spent in each did not differ greatly from 2003 (see Table 7-9), several institutions stated that they either have no lectures at all, instead presenting in a more interactive ‘workshop’ format, or that all lectures were available only online via online classroom software and/or audio or video recordings. Most instructors commented that all course materials were available online and that often students did not attend classes, instead choosing to download materials from home, even when the course was not offered by distance education.

Several interviewees commented on the success of the ‘workshop’ approach: “Workshops are a much more successful way of teaching programming. Give them small bits and then have them do it straight away.”

	Lecture	Tutorial	Practical	TOTAL
2003	2.2	0.6	1.8	4.6
2010	2.1	1.1	1.2	4.4

Table 7-9 Hours in class on-campus

The delivering of information in small meaningful chunks to reduce cognitive load is called the ‘segmentation effect’ (Mayer and Chandler, 2001), and could be one of the reasons for the observed effectiveness of the workshop approach.

7.8.3 Instructor Experience

Instructors were asked how many years they had taught introductory programming. The average amount of experience has risen since 2010 (see Table 7-10). However it should be noted that casual teachers were often not available for interviews, and this may have skewed the results.

	Minimum	Average	Std Dev	Maximum
2003	0.5	8.6	7.2	30
2010	2	12.3	7.3	30

Table 7-10 Instructor experience in years

7.8.4 Texts used

Of the 44 courses in this current study, 11 used no text at all, and many instructors commented that they encouraged students to use online resources or to find their own text. Most of the remainder used language- or environment-specific texts (eg: "Programming in Visual Basic 2010" (Bradley and Millspaugh, 2011)), with no real commonalities. Four more generic texts - “Connecting with Computer Science” (Anderson, Hilton and Ferro 2010), “Simple Program Design” (Robertson 2000), “Programming Language Concepts” (Sebesta 2007) and “A simple and generic introduction to OO Algorithm Design” (Robey undated) - were used, each in only one course.

As mentioned previously, many instructors commented on an increased reliance on online resources. While only 11 courses were offered by distance education, most of the instructors offered the information that all of the resources were available online to all students, often using Moodle or Blackboard learning management systems. Several courses included recorded videos of lectures and/or tutorials in the resources available to students.

7.8.5 Problem Solving Strategies

As a follow-on from the focus area in the 2003 census, instructors were again asked how much time they dedicated during class time to teaching and discussing problem solving strategies within the context of programming.

The average percentage time given to problem solving, along with standard deviations, have remained stable from 2003 to 2010, and are presented in Table 7-11. The percentage of times given to problem solving in both lectures and tutorials remain unchanged in this period.

	Lecture		Tutorial	
	Average	Std Dev	Average	Std Dev
2003	29%	22%	46%	36%
2010	29%	26%	44%	33%

Table 7-11 Percentage of class time dedicated to problem solving

It is worth noting that while the means between years are very close for both lectures and tutorials, the standard deviations on these measures are relatively large, indicating substantial variations between different courses in how much time they dedicate explicitly to problem solving.

Some participants indicated that ‘problem solving’ had been moved into a separate course and so they did not deal with these strategies in the programming

course. In contrast, others stated that problem solving was implicit in everything they did in the lectures and tutorials, although they did not explicitly teach problem solving strategies.

Although the average percentage of time given to focusing upon problem solving has remained static from 2003 to 2010, there may be differences in how problem solving is embedded within curriculum structures, classroom delivery and learning activities between these two dates. This information, however, lies beyond the bounds of this thesis.

Of interest is that the level of emphasis on problem solving has remained constant, despite experimental evidence in other domains (for example, mathematics, geometry and physics) that for novices, worked examples are more effective than teaching problem solving strategies (Atkinson et al., 2000). Yet problem solving appears to be the teaching method of choice.

This emphasis on problem solving theoretically adds to the cognitive load experienced by novices and may effectively block their learning of computer programming - *even if* students are successfully solving the problems.

7.9 Results - Mental Effort Measures – Instructors and Students

7.9.1 Novice programming, mental effort and cognitive load

In a similar fashion to the local IT Girls study and the Remote Schools study, instructors were asked about the mental effort expended by themselves and their students while solving a novice programming problem.

Learning the three generic concepts of sequence, iteration and selection is an integral part of all first programming courses. A novice programmer will need to

acquire this knowledge base as schemas and automate them, whereby they can be applied with relatively low levels of conscious attention.

Novice programming problems that require the student to learn and use any of these three concepts can be said to have three sources of mental effort (or cognitive load):

- ❖ Understanding and processing the problem statement, and deciding on the best algorithm and structures to use to solve the problem;
- ❖ Navigating or using the environment, tools or language in an attempt to solve the problem; and
- ❖ Learning how to best use these structures so they can be used in further, more difficult problems.

If learning the basic generic concepts of sequence, iteration and selection is the objective of having students solve these novice programming problems, then these three aspects broadly equate to the three identified sources of *cognitive load* (Sweller, 1988) on working memory while learning: intrinsic, extraneous and germane. The term ‘mental effort’ – the level of conscious focus of attention one has to give to a task, whether it is cognitive, physical or a bit of both - is used in the current study as a means of evaluating the cognitive load associated with these various aspects of learning programming.

7.9.2 Measures of mental effort

As part of this study, instructors were asked about their three sources of mental effort expended whilst solving a novice programming problem:

- ❖ understanding and processing the problem statement [intrinsic cognitive load];
 - ❖ navigating or using the environment, tools or language [extraneous cognitive load];
- and

- ❖ learning from the problem and reinforcing previous concepts [germane cognitive load].

Instructors were asked to rate their own levels of mental effort on each of these three dimensions using a 9 point Likert scale, where 1 = "no mental effort" and 9 = "extreme mental effort". Instructors were subsequently asked to rate their expectations regarding these three dimensions of mental effort for an average student in their introductory programming course, and then again, for a student in the "bottom 10%" of the course performance. This self-reported assessment of perceived mental effort is taken as a measure of the cognitive load associated with a task performance.

There were some participants who did not immediately offer a response for all, or an aspect, of this series of questions, particularly for the "bottom 10% of performance". These were removed from the first series of analyses, but will be commented upon further below.

Table 7-12 shows the median and mode for each of these cognitive load areas, for instructors, the average student and students in the 'bottom 10%'. The data was heavily skewed, so measures of central tendency provided are modes and medians.

	instructor	average student	bottom 10% student
intrinsic	mode	2	6
	median	2	6
extraneous	mode	2	5
	median	2	5
germane	mode	2	7
	median	2	5.5

Table 7-12 Mental effort scores

An initial analysis was performed on the mental effort scores given by participants (instructors), for each of these three components of cognition --

understanding the problem statement, using the environment, and reinforcing previous concepts. A series of comparisons was made between firstly the self-rating of the participant (a first year programming instructor) and the rating anticipated to be experienced by an ‘average student’; and then between the anticipated average student’s rating and the anticipated level to be experienced by a student in the bottom 10% of the class’s performance. The comparisons were performed using a series of Wilcoxon Signed Rank tests (one-tailed) for all measures. Table 7-13 shows the results of these comparisons.

In summary, these results indicate that for *each* of these three sources of cognitive load, the instructors in the introductory programming courses rated their own levels of required mental effort to be low, and that they expected that average students would need to exert higher levels of mental effort than themselves - ‘above average’.

Instructor < average student (greater mental effort)					
	W	Ns/r	z	p	n
Understanding and processing the problem statement	811	43	4.39	<0.0001	43
Navigating/using the environment, tools or language	838	41	5.43	<0.0001	43
Learning from the problem/reinforcing previous concepts	647	40	4.34	<0.0001	42
Average < bottom 10% student (greater mental effort)					
	W	Ns/r	z	p	n
Understanding and processing the problem statement	207	24	2.95	0.0016	25
Navigating/using the environment, tools or language	276	23	4.19	<0.0001	25
Learning from the problem/reinforcing previous concepts	144	20	2.68	0.0037	21

Table 7-13 Wilcoxon Signed-rank test - comparisons of mental effort

Additionally, for *each* of these three sources of cognitive load, the participants rated the anticipated mental effort to be experienced by a student in ‘the bottom 10%’ to be higher again, compared to an average student, in the rating of ‘high’ to ‘extreme’ mental effort.

These results are consistent with the argument provided earlier regarding cognitive overload. Students who are in the bottom 10% of performance are perceived by the introductory programming instructors to be effectively swamped in mental effort on each of the three measures. As these three areas compete for cognitive resources it is unlikely that students can actually allocate such high levels to all three components, despite their desires to do so, and despite the instructors’ perceptions that students are attempting to do so. The statistical analysis, therefore, merely shows that instructors assess that there is a *significant difference* between their own cognitive load (typically low) and an average student’s cognitive load while solving the problem (moderately high), and again, assess that there is a *significant difference* between the cognitive load experienced by an average student and that experienced by a student in the bottom 10% (“extreme”).

Human cognitive processes are limited. Students have no option but to try and understand and respond to the programming problem statement, as this is the primary task to which they have been explicitly directed to engage with. The student also has no option but to navigate, as best he or she can, the programming language and syntax, as this is the environment which they have explicitly been directed to operate within. Given that these two tasks are each placing very heavy processing burdens upon mental resources it may be expected that the third area, those of germane activities, are effectively strangled of processing resources.

Recall that not all participants gave a score for the mental effort measures, instead choosing to provide comment. The second layer of analysis focuses upon the verbal comments provided by all participants regarding their perceptions of mental effort for the different student cohorts.

7.9.3 Participants that did not give a score for all or any of the cognitive load aspects

A total of 44 participants took part in the survey, and all of these participants answered the questions regarding the mental effort experienced by themselves on each of the three factors while solving a novice programming problem. One of the participants declined to answer the questions regarding average and ‘bottom 10%’ students as he/she indicated that he/she believed “that the ‘average student’ could be designated as any of the scores on the Likert scale”. We consider this to indicate a misinterpretation of our line of enquiry, and no further consideration is given to the responses provided by this participant. Of the 43 remaining participants, one declined to give a score for the “germane” aspect for the “average student”, whilst offering the scores for the other dimensions.

Only 25 participants gave any scores for the “bottom 10%” and often this was coupled with qualifying statements. Only 21 of these participants were willing to specify a score for mental effort for the germane aspect for students in this bottom 10% of performance in the course.

7.10 Results - Mental Effort Measures – the bottom 10%

7.10.1 Participants' Comment Analysis

Many participants indicated that providing an estimate of mental effort for lower performing students was a much more complex scenario than could be expressed by a simple numerical answer to a Likert scale question. Even those who did give a score often added comments to qualify their answer. These comments were analysed further to give insight into the cognitive load being anticipated to be experienced by these students.

7.10.2 Methodology of further analysis

Truncated versions of each interview transcript were created for the purposes of further analysis, temporarily disregarding data on languages, textbooks, on-campus hours and other aspects of the survey that had no bearing on mental effort measures. Data retained included mental effort measures for “students in the bottom 10% of the course” (where given), any transcribed comments that offered insight into mental effort (or lack of such) used by students, and any other comments that were offered by participants throughout the course of the interview that had bearing on possible reasons for success or failure of students in the bottom 10% of the course.

Interviews were confidential and participants were encouraged to be open, frank and fearless. On this basis the researcher has redacted phrases that may be considered to be disrespectful of students as clients of the education system. Thematic analysis of the comments has been performed, but in some cases specific terminology has been withheld. In these cases, “[redacted]” has been used. This indicates terminology (not profanity) that may be interpreted as offensive to the people thus identified and labelled.

7.10.3 Student Profiles - Ghosts, Idlers and Strivers

This survey asked for Likert scale measures for mental effort for students in the bottom 10% of course performance. If the instructor asked the interviewer for clarification of type of student targeted, then the interviewer (using the terminology used by the instructor) indicated that the mental effort expended by a less capable student who is trying was required.

[Instructor] “by effort do you mean students that apply themselves that are [redacted]? Or students that don’t apply themselves?”

[Interviewer] “apply themselves that are [redacted]”

However, some instructors offered two values (one for students who did not try, and those who were trying), and others stated that one simple measure couldn't be given and instead offered comments. For this reason, the comments associated with all of the types of student profiles, as well as mental effort measures for the student profile of “students who are less capable but are trying”, have been analysed.

The interview summaries were examined for commonalities between participants regarding the profiles of students composing the bottom 10% of their course. Responses indicated that 34% (15/44) of participants explicitly indicated that more than one student profile existed in the student composition of the bottom 10% of their course. All participant summaries were examined to identify the range of profiles of these students. The resulting profiles have been given names to simplify discussion for the remainder of this chapter.

The profiles of students that were identified by instructors were:

- ❖ ‘Strivers’ - less capable students who are actually trying - identified by 75% of participants;

- ❖ ‘Idlers’ - those students who attend class but who do not try - identified by 40% of participants;
- ❖ ‘Ghosts’ - students who do not attend class/ are not seen by instructors - identified by 14% of participants.

Each profile type was then examined separately using thematic analysis across participants to determine possible reasons given by participants for lack of success by these students. Note that participants have been identified throughout the remainder of this chapter by transcript number -- for example [1] -- to show that comments have been sourced from a range of participants.

7.10.4 Ghosts - students who are not there

A relatively small number of participants (14%) identified that some or all of the students in the bottom 10% of the course were never seen by the instructor, or disappeared early. This cohort of students has been designated “Ghosts”. Reasons offered for this disappearance revolved around the students’ perceived lack of motivation and desire to participate in the course or their perceived lack of choice in studying programming as part of their degree structure. Comments provided regarding the “Ghost” cohort include:

- ❖ “Because they haven’t chosen to do programming, they are resentful and not motivated and they find it difficult to get their head around it to do it.” [3]
- ❖ “Enrolled because [they] “have to” - they are required to be enrolled full-time. Students are embarrassed to tell parents they’ve made the wrong decision about a university course (as opposed to a TAFE course) and just stop going. Only 1% of students choose to do the unit, for the rest its core. They don’t really want to be there.” [25]

- ❖ “they don't want to come onto a university campus, they see this as an enormous burden that gets in the way of their crappy part-time job, and basically their commitment is (on average) pathetic” [31]
- ❖ “its coming back to the students' desire and motivation to learn, and that's the main problem we are having at the moment. I don't think that teaching or learning programming is a difficult thing, I think it's the students that are the problem at the moment.” [2]

7.10.5 Idlers – students who do not apply themselves

Instructors identified two separate profiles of students who *did* attend classes or were ‘seen’, within the bottom 10% of students. A cohort of ‘students who attend class but don't try’ were identified by 40% of participants. The author has designated this profile of students as “Idlers”. “Strivers” - the remaining profile of students - are examined in Section 7.10.6.

Some of the reasons offered by instructors for the Idlers’ lack of progress were similar to those offered for the Ghosts.

Students’ lack of motivation and desire to learn was given as a reason in many cases:

- ❖ “[they] don't take the course seriously” [1]
- ❖ “You have the students that don't work and don't put any effort in, and I have to separate that whole cohort out and say that there is very little mental effort because they don't do anything. In workshops they are on Facebook or chatting or getting the answers from their mates rather than working it out for themselves.” [5]
- ❖ “They don't reflect and they don't pay attention to the specifications. They don't take time to look at the problem at hand ... ” [34]

- ❖ “There is very little mental effort because they don’t do anything. In workshops they are on Facebook or chatting or getting the answers from their mates rather than working it out for themselves.” [5]
- ❖ “Don’t try. Hand in other people’s assignments, don’t attend, don’t do tasks, are on Facebook.” [23]
- ❖ “just don’t try hard enough” [24]
- ❖ “not motivated enough to try” [26]

Several instructors offered the reason for this lack of effort to be that some students were forced to do the unit by their degree requirements, had no intent of going further, did not enjoy programming and that these students didn’t apply themselves to learning:

- ❖ “Different cohorts: Programming students and education/business degrees – the second lot don’t try” [23]
- ❖ “Only about 20% of students [in the course] intend to progress with programming” [23]
- ❖ “Students enroll and discover they don’t like programming, so the effort they expend is minimal” [28]

Section 7.5 reported on the falling numbers of students in programming courses and ICT programs as a whole. Some universities have responded to this trend by lowering entrance score cut-offs, and instructors identified this as a source of “Idlers” those students who perhaps were not suited to university study, but had managed to enter the course:

- ❖ “quite a low entry bar into our computing degrees” [29]
- ❖ “we have ‘OP [Overall Position Tertiary Entrance Rank] ridiculous’ as a large cohort” [31]

More concerning were the comments offered by some instructors that indicated that these “Idlers” were predisposed to believe it was impossible to succeed, or became frustrated so early in the course that they gave up:

- ❖ “Students come in and say "I've heard programming is so bad and it's the end of the world" ... And so they are *defeated before they get here* [emphasis added]. But they just don't do any work ...They are completely directionless. ” [34]
- ❖ “There are some students who are too frustrated with it to really try that hard, and dismiss the learning. “ [38]

7.10.6 Strivers – Students who are less capable but are trying

Most (75%) instructors indicated that the bottom 10% of students in their course contained students who were less capable but were trying to succeed. These students have been designated as “Strivers” for the purposes of this thesis. Some of these instructors gave Likert scale measures for Strivers, while others offered comments, and some offered both measures and comments. Table 7-14 shows the mode, median and range for the measures collected. Recall that the scale represented 1 for “no mental effort” up to 9 for “extreme mental effort”.

Mental effort	n	Mode	Median	Range
Understanding and processing the problem statement	22	9	8.5	min = 6 max = 9
Navigating and using the environment, tools or language	22	9	8	min = 5 max = 9
Learning from the problem/reinforcing previous concepts	21	9	8.5	min = 5 max = 9

Table 7-14 Mental Effort for Strivers

Instructors also often offered indicative comments about the level of mental effort for Strivers, regardless of whether they gave a Likert scale number. These comments are offered below to show the general agreement that for this profile of

students the mental effort required for all three aspects was very high to extreme, and in some cases “off the scale”. These include:

- ❖ “off the scale” [11]
- ❖ “putting all their effort in” [15]
- ❖ “try very very hard” [16]
- ❖ “increases everything” [20]
- ❖ “expend more energy for borderline pass - more effort in getting that borderline pass than others use to get higher marks” [28]
- ❖ “very high level of mental effort” [30]
- ❖ “factor of 10 at the very minimum” [34]
- ❖ “try but can’t solve the problem at all” [35]
- ❖ “a lot more effort” [41]

Given that these Strivers are often being reported as trying as hard as they can, it is important to consider the reasons given for why they find it so difficult, or fail (if any reasons were given).

One group of instructors offered the opinion that students had an innate aptitude (or lack of aptitude) for programming and that if this was missing, those students would never understand programming - no matter how hard they tried. These comments are presented below:

- ❖ “some of them pick it up quickly, some take longer and some never pick it up at all.” [7]
- ❖ “they struggle - they genuinely try and they put a lot of work in but it’s not how their brains are wired” [22]
- ❖ “There are those that get it … and some people for some reason just don’t get it and they are hopelessly lost, and they just never seem to be able to get it at all.” [23]

- ❖ “some students have some sort of mental block as far as programming is concerned.

They might find it difficult to even follow a set of sequenced step-by-step instructions.” [27]

- ❖ “they try really hard but the penny doesn’t drop” [22]

Other instructors commented that the Strivers lacked literacy, comprehension and analysis skills prior to entering the introductory programming course, and that this compounded the difficulty of learning the material for this cohort:

- ❖ “they have very poor literacy and comprehension skills” [15]
- ❖ “[Programming] is really a bit harder than the other units - it requires some analysis and some mathematical nous almost. And they simply don’t have it.” [34]

Strivers from a non-English-speaking background had a particularly hard time, according to some participants. Some of these students may be more capable (than their programming performance indicates) but the additional cognitive load imposed by working in English as a second language means that they fail to learn or perform:

- ❖ “high level of overseas students - sometimes what seems plain to us is not plain to them” [27]
- ❖ “often their problem may be that they are not so fluent in English” [41]
- ❖ “Non-English students struggle with dealing with the help systems. They are familiar with synonyms so unless the word is exactly the same as the one they have typed in they cannot find it.” [43]

Often instructors offered comments about the difficulties, confusion and extreme mental effort experienced by the Strivers, but did not offer a reason as to why this was occurring:

- ❖ “Extremely difficult for the bottom ones to understand. It’s hard for them. The bottom ones avoid it [learning from the problem].” [8]

- ❖ “Often have no idea what to do. Shows to me that they are really not able to read the problem and try to understand the components – I guess that’s “off the scale.” [11]

Several instructors commented on the additional load imposed by the complexity of the language syntax or the environment, particularly if students had not encountered them previously:

- ❖ “Some get stuck on syntax - if you look at how it's written, it won't compile and run, so we find it hard to teach them to pinpoint errors in the code. They are distracted by debugging and ultimately they lose the motivation to look further.” [36]
- ❖ “The bottom 10% of students are usually unfamiliar with the programming environment including strict logic, so they have to put in a lot more effort.” [41]
- ❖ “They don't have the [programming] language skills so it's about 9, and the rest of it becomes impossible because they get stuck and they can't go any further.” [44]
- ❖ “What we find with the students is if the environment is too complicated then they don't know the difference between the environment and the language, and if it's too simple, then it's not giving them any help. Like the command line – it's not a good way of teaching because they have to come to grips with the file structure, and things, so you need an environment that takes away those sort [sic] of mechanical elements but still is not full of hundreds of different features that confuse them. ” [7]
- ❖ “it requires an enormous mental effort because it's so new, both the language and the environment” [9]

Each of the above comments refers to the added mental effort imposed by dealing with aspects of the language, environment, and tools (debugger) while attempting to solve a novice problem, yet do not offer any solutions of how this may be reduced in complexity.

Another broad observation by participants is the general lack of concept generalisation displayed. The high cognitive load experienced by these students is

shown in their inability to form schema, and hence to generalise and notice patterns.

No transfer is shown by these students:

- ❖ “Generalisation ... or noticing patterns, is extremely difficult for students. Plenty of times I give them identical problems which to me are identical and then the students see them as completely different problems that are unrelated to previous ones. They just don’t see it. What do they learn from that?” [1]
- ❖ “They can do the same problem 4 times in a row and trip over the same bug. Very frustrating actually.” [22]
- ❖ “they are putting all their effort in, but it’s usually misdirected” [15]
- ❖ “there are students in the bottom portion of the class who will spend a lot more time trying to work out a problem than the average student, but they won’t learn from the experience. And there are students that will.” [38]

The view that students in the bottom 10% of a programming course are simply without the capacity to learn this content, is lamented in a comment by Participant Number 19:

- ❖ “the bottom 10% just can’t do it - they flounder” [19]

The final theme offered by instructors concerned the structure of materials or the course expectations. They indicated that in some cases, the bottom 10% of students are expected to fail, and accommodations are not made for these students.

- ❖ “Most coordinators running these types of courses focus too much on the stronger students: and they should focus on the weaker students.” [41]
- ❖ “The bottom 10% of students wouldn’t be expected to pass the unit anyway” [43]
- ❖ “The assignments are set as a challenge, but not something that is impossible. The good students find them easy, the poor students struggle, and that’s the way it is, we can’t have them too easy or too hard.” [7]

The comments offered by participants may be summarised as follows:

- ❖ many identify multiple profiles of students within the bottom 10% of a programming course;
- ❖ some students (Ghosts) are enrolled but never have any intention of attending classes, let alone learning content;
- ❖ some students (Idlers) are attending classes, but do not apply themselves to designated tasks (for various reasons) and as such, do not apply sufficient mental effort to enable learning of content;
- ❖ some students (Strivers, who were identified by the majority of participants as existing in the bottom 10% of performance within a course) are attending classes, are motivated, are completing (as best they can) designated tasks, are exerting very high to extreme levels of mental effort on everything ... and yet are not demonstrating learning.

The comments from participants indicate that their perceptions of the Strivers are that these students are suffering from excessive mental effort (cognitive overload).

Specific themes identified by participants within this profile included:

- ❖ Lack of ability in problem solving
- ❖ Lack of innate aptitude for programming
- ❖ Lack of literacy, comprehension, and analysis skills
- ❖ Lack of English due to English being a second language
- ❖ Difficulty of the computer language and/or environment being used
- ❖ Lack of capacity to generalise concepts
- ❖ Instructional materials that do not cater for their needs.

7.11 Discussion

The results of the 2010 study are disturbing. Student numbers studying programming are continuing to decline dramatically, and of those students who are retained, many are struggling.

A clear majority of participants who are academics teaching introductory computer programming courses (units) indicated explicitly the view that there are students in the "bottom 10% of performance" of these courses (as well as some average students!) who are applying themselves to the set learning activities, are completing the assignments (as best they can) and yet are failing to learn.

Moreover, the reported measures of perceived mental effort by the most frequent (modal) participant response indicates that these very same students who are failing to learn are putting in *extreme levels* of mental effort. These students cannot be asked to do more. If these students are to have success in learning computer programming then aspects of the instructional design for introductory programming courses must change.

Many participants indicated that movements towards such changes have already occurred as demonstrated by selection of programming language and the increased use of IDEs over the period 2003 to 2010, predominately for pedagogical benefits - that is, to help students' learning – over industry relevance. Yet students still fail to learn, despite their best (mental) efforts.

It is likely that a range of factors contribute to the difficulty of learning computer programming in the current regime. These include the nature of programming itself, the apparent necessity to house activities within a computer language (with associated syntax) and often the inclusion of problem-solving activities.

Cognitive Load Theory warns that the limitations of human cognitive resources means that learning is prone to fail if these resources are not carefully managed in a teaching-learning transaction. The results of this study indicate that many students in the lower end of performance within introductory programming courses need to apply large tracts of their cognitive resources to cope with the intrinsic nature of the to-be-learnt content. These students are also required to deploy large, and possibly unnecessary whilst learning basic concepts, levels of cognitive resources to extraneous aspects of instruction, such as the nuances of a programming language and syntax. With such high levels of cognitive resources already deployed it is likely that there is effectively nothing left to give to the processes of learning.

Cognitive Load Theory has proven to be an extremely effective utility in engineering better instructional designs for traditionally complex and difficult areas of study such as mathematics, physics and electrical circuitry (see Sweller, 1999). Possibilities of improving the instructional design of introductory programming courses may also follow by embracing cognitive analysis of the processes and dynamics associated with learning computer programming. Of primary interest is the potential for selection of language and environment to hold the potential of lowering a source of extraneous cognitive load, to thus free cognitive resources for application to germane usage, with the intent of facilitating learning.

Success in teaching the basic fundamentals of programming structures has already been experienced with the local IT Girls Days as well as the Remote IT Girls cohort of students, with the results of increased self-efficacy in programming, lowered perception of the difficulty of the material, high enjoyment and moderate levels of mental effort expended during the programming exercises. The contrast between this experience and the picture painted by the first course programming instructors is

stark. The disconnect between the top-down (instructors) and the bottom-up (school students) is clear. The author decided that it may be a useful to ask the actual university students studying introductory programming directly about their experience in studying a novice programming course.

To further explore the levels of mental effort experiences and the attitudes of students towards the difficulty of learning programming in a first programming course, a study was conducted with students undertaking a local university introductory programming course. This study is discussed in the next chapter, Chapter 8.

Chapter Eight: Student Survey

8.1 Introduction

The academics surveyed in the first year programming course study detailed in the previous chapter reported that some first year university students are putting in “extreme” levels of mental effort, when trying to solve introductory programming problems. These measures, however, were anecdotal, from the perspective of the instructors. Further perspectives on the level of mental effort and the perceived difficulty of learning programming were sought from current introductory programming university students in a survey. The methodology of this survey and its results are reported in this chapter.

8.2 Background

The introductory programming course chosen for this study was one that has been offered for several years to students at Southern Cross University, an Australian regional university, in the first study period of the first year of a Bachelor of Information Technology degree. Although the course is intended primarily for Information Technology students and Applied Computing students, some students studying business, technology education and other degrees select the unit as part of a major course of study or as an elective.

The course covers “fundamental programming concepts as well as ... system development issues” (Southern Cross University, 2010). A structured programming paradigm is used, and the syllabus includes general application development concepts, variables and constants, decisions and conditions, modularisation, repetition, arrays, database and other file access, user interface design and using multimedia. The

language used in this course is Visual Basic .NET and the course uses the IDE Visual Studio 2010 for program development.

8.3 Methodology

All on-campus and distance education students in the course were invited to participate in an online “SurveyMonkey” survey that was presented in two parts. The first part was presented near the beginning of the session (weeks 3 to 4) and the second part was presented near the end of the session (weeks 12 to 14) of the first study session of the year. The invitation to participate was disseminated via an email from the course assessor, with an information sheet about the survey attached to the email (see Appendices D.1 and D.2). A reminder email was sent to the students in Week 11, requesting that they complete the second part of the survey.

8.3.1 First part (Weeks 3 to 4)

The first part of the survey (see Appendix D.3) asked for the participant’s name and, optionally, their email address for dissemination of the group results of the survey. The survey also asked for:

- the age group of the participant in four categories ('under 20', '21-25', '26-35' and 'over 35');
- gender;
- first language spoken at home as a child;
- course of study ('degree');
- study mode ('on-campus', 'international on-campus' or 'external (distance)'); and
- time spent at university ('first year', 'second year', 'more than 2 years').

Participants were asked a series of 9-point Likert scale questions regarding their computer skill level, programming experience and levels of mental effort in each of

the three dimensions previously used in the IT Girls (Chapters 5 and 6) and Programming Instructor (Chapter 7) surveys. Recall that these three dimensions of mental effort specifically addressed “understanding and processing the problem statement”, “navigating and using the environment, tools or language” and “learning from the problem/ reinforcing previous concepts”. All Likert scales were presented in a horizontal format with low values on the left-hand side of the scale and high values on the right-hand side of the scale.

For the two questions regarding “skill level in using a computer as a tool” and “computer programming experience prior to starting this unit”, these ranged from “new to computers” and “never programmed” to “can use computers for advanced tasks and format professional documents” and “extensive experience in programming” respectively. The mental effort values ranged from “no mental effort” on the left to “extreme mental effort” on the right-most end of the scale.

8.3.2 Second part (weeks 12 to 14)

In the second part of the survey (see Appendix D.4), participants were asked for their name for matching purposes, and then the questions about computing skill level, programming experience and mental effort were repeated. Participants were then asked the open question “How do you think these three areas of mental effort affected you while studying in this unit?”

8.3.3 Matching

At the end of the study period, the two parts of each survey were matched by name and the grade received by each student for the course was matched to the student’s survey responses by a delegated third party unrelated to any teaching of the

Information Technology course. The student names were then removed from the survey data sets.

The resulting de-identified data was analysed to determine relations between the performance and demographic factors collected, compared to the mental effort measures. Specifically, performance (as indicated by final grade), previous programming experience, general computer literacy, first language, gender, age group, study mode, and the year of study, were compared to the mental effort expended on each of the three measures for intrinsic, extraneous and germane cognitive load.

8.3.4 Hypotheses

Using the results of the Australian University Survey reported in Chapter 7 as a basis for the generation of hypotheses it was expected that there would be differences in the levels of mental effort measures based on:

- performance;
- prior programming experience;
- general computer literacy; and
- first language.

Specifically, the following hypotheses follow:

- ❖ **H1 (Performance):** Participants obtaining lower grades will be more likely to experience higher levels of mental effort.
- ❖ **H2 (Prior Programming Experience):** Participants with lower levels of prior programming experience will be more likely to experience higher levels of mental effort.

- ❖ **H3 (General Computer Literacy):** Participants with lower levels of general computer literacy will be more likely to experience higher levels of mental effort.
- ❖ **H4 (First Language):** Participants studying in a second language will be more likely to experience higher levels of mental effort.

Other data regarding participants' demographical profile were also obtained for:

- gender;
- age;
- study mode; and
- year of study.

These had not been commented on within the Australian University Survey by the participating academics. That is, the participants had not indicated the view that any of these factors contributed to the mental effort experienced by students undertaking introductory programming at university. These generate hypotheses that are without specified directions. The following hypotheses follow:

- ❖ **H5 (Gender):** Gender will be a factor determining levels of experienced mental effort.
- ❖ **H6 (Age):** Age will be a factor determining levels of experienced mental effort.
- ❖ **H7 (Study Mode):** Study Mode will be a factor determining levels of experienced mental effort.
- ❖ **H8 (Year of Study):** Year of study will be a factor determining levels of mental effort.

The results reported for mental effort in the Australian Academics Survey (Chapter 7) did not display separation between the three sources of mental effort. That is, the mental efforts for intrinsic, extraneous and germane were reported as being all low for the academics, all moderate for the average student and all very high for the bottom

10% of students. On this basis it was anticipated that the three measures for different sources of mental effort would generally rise and fall together.

8.4 Results and Discussion

8.4.1 Participants

A total of 28 students participated in this study. One of the participants withdrew from the course (and therefore was not awarded a grade) and one student completed the second half of the survey, but not the first part. These two students' data has been excluded from analysis. Although grades are available for the remaining 26 students who completed the first half of the survey, only 18 students completed the second part of the survey. Grades awarded were one of Fail (F / under 50%), Pass (P / 50 – 64%), Credit (C / 65-74%), Distinction (D / 75-84%) and High Distinction (HD / 85%+).

8.4.2 Representative Sample

Although relatively few students completed the survey, visual inspection indicates that they were a representative sample with respect to the grades obtained within the student profile of the course, disregarding those students who withdrew from the course (see Section 8.4.3). The grade distribution of the course and of the survey participants is shown below in Figure 8-1.



Figure 8-1 Grade Distribution Comparison

Females were slightly more likely to participate in the survey (see Table 8-1 below), however the distribution between on-campus students, external (distance study) and international on-campus participants was very similar to the distribution for the course (Table 8-2).

	Male	Female
Survey	73%	27%
Course	81%	19%

Table 8-1 Gender Distribution

	On-campus	International on-campus	External (distance)
Survey	31%	15%	54%
Course	24%	15%	61%

Table 8-2 Study Mode Distribution

8.4.3 Students who failed/withdrew from the course

This introductory programming course had a high 'voluntary attrition' rate of 31% of initial enrolments. This 'voluntary attrition' comprises students who enrolled in the course and then withdrew at any time during the semester, as well as those

students who stayed enrolled, but did not submit any assessments, and did not attend the final examination (the "ghosts" of Chapter 7). In addition, 13% of the students who did complete the course failed.

The data from the student survey is biased (see Section 8.5.1 below) because the students who failed or withdrew from the course are not proportionally represented in the survey sample; however, it *is* representative of the students who completed the course.

Of the two survey participants who failed, both self-identified as having relatively high levels of previous programming experience (6 and 7 on the 9 point Likert scale), high levels of computing skills (6 and 8 respectively), and low levels of mental effort required to solve introductory programming problems (they were not finding the content difficult), yet failed. Due to the low number of 'failing' participants, and the unreliability of their self-reporting, they have been excluded from the performance analysis in Section 8.4.4 below.

Despite the low numbers of participants, it was decided to proceed with the analyses, but to treat this primarily as an "investigative" study seeking to determine the broad validity of the comments gathered from the academics who participated in the Australian Academics Survey (Chapter 7)

8.4.4 Hypothesis 1 (Performance versus Mental Effort)

H_A1 : Participants obtaining lower grades will be more likely to experience higher levels of mental effort.

H_01 : Participants obtaining lower grades will not be more likely to experience higher levels of mental effort.

Analysis

For the purposes of analysing the performance of students compared to their reported mental effort in each of the three dimensions (intrinsic, extraneous and germane), grades have been grouped into Pass and Credit students, versus High Performance students (Distinctions and High Distinctions). Results from the mental effort scores have been grouped into Low (score 1 – 3 on the Likert scale), Medium (4 – 6) and High (7 – 9) categories. Table 8-3 shows the number of participants scoring in each category, mapped against final grade. Note that this is the mental effort expended in the *first few weeks* of the course, while solving introductory programming problems.

Grade	Understanding the problem statement			Using the environment and language			Learning from the problem		
	low	medium	high	low	medium	high	low	medium	high
Pass/Credit	1	10	4	3	7	5	1	6	8
Dist./High Distinction	3	4	2	5	4	0	3	5	1

Table 8-3 Grade versus Mental Effort

Fisher Exact Tests were performed on the data set, with the following results (see Table 8-4). From these results, the null hypothesis H_0 is accepted. Participants with lower grades are not experiencing higher level of mental effort. However, there is a trend at the 0.1 level, indicating that lower grades are associated with higher levels of mental effort, and this may represent a real effect.

	p	Statistical significance
Understanding the problem statement	0.3643	Not significant
Using the environment and language	0.0862	Not significant
Learning from the problem	0.0697	Not significant

Table 8-4 Performance versus Mental Effort Analysis

Although the analysis did not yield statistically significant results at the 0.05 level, the data may suggest a trend indicating lower-performing students reporting that they were using more mental effort, as opposed to the high achieving students, who typically used low to medium levels of mental effort.

This trend, present for “using the environment and language” may represent a real effect and is consistent with Cognitive Load Theory, which argues that lower levels of extraneous cognitive load - in this case, the load associated with navigating and using the environment in the first few weeks of the course - will be associated with higher learning performance.

The trend that lower-performing students used more mental effort than the high achieving students is also present for “learning from the problem”. This is also consistent with the views reported by the academics in the Introductory Programming Survey, whereby lower performing students were considered to be expending higher levels of mental effort across all three measures of cognitive load. Despite the trend indicating higher mental effort being given to the activity of learning, these students have not been successful in gaining the higher grades (Distinction and High Distinction).

8.4.5 Hypothesis 2 (Programming Experience versus Mental Effort)

H_{A2}: Participants with lower levels of prior programming experience will be more likely to experience higher levels of mental effort.

H₀₂: Participants with lower levels of prior programming experience will not be more likely to experience higher levels of mental effort.

Analysis

Interestingly, those students who rated their previous programming experience as high (7, 8 or 9 on the 9-point Likert scale) did not receive high grades in this course (Figure 8-2).

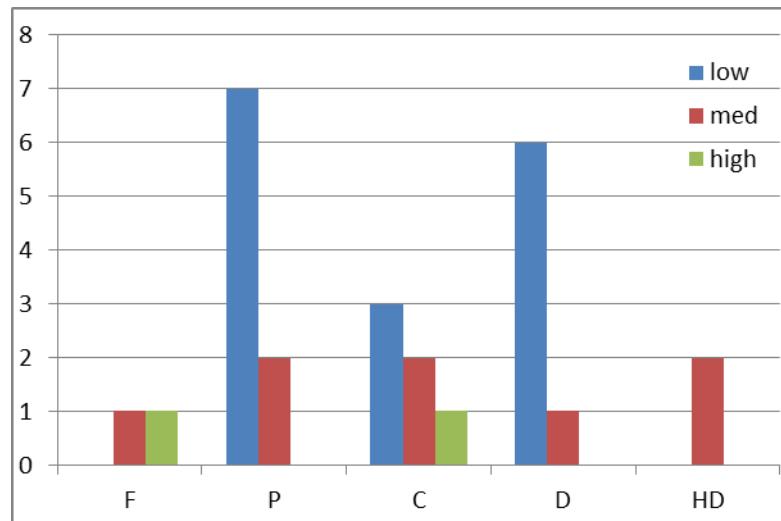


Figure 8-2 Previous Programming Experience versus Grade

For the purposes of analysis, those participants who reported that they had no or minimal programming experience (1 to 3 on the Likert scale) and could be classed as ‘novices’ were compared with those who had more programming experience (4 – 9 on the scale). The numbers of students in each category are shown below in Table 8-5.

Experience	Understanding the problem statement			Using the environment and language			Learning from the problem		
	low	medium	high	low	medium	high	low	medium	high
Novice	1	10	5	4	7	5	1	8	7
Not novice	4	5	1	5	5	0	4	5	1

Table 8-5 Programming Experience versus Mental Effort

Fisher Exact Tests were performed on the data set, with the following results (Table 8-6):

	p	Statistical significance
Understanding the problem statement	0.1161	Not significant
Using the environment and language	0.1839	Not significant
Learning from the problem	0.0705	Not significant

Table 8-6 Programming Experience versus Mental Effort Analysis

A closer inspection of the data indicated that many of the participants reported their mental effort at 4 or below, which is at the lower end of the ‘medium’ category. Re-dividing the data into substantial mental effort (5 or above) and lower mental effort (1 to 4 on the scale) gives the following results (Table 8-7):

Experience	Understanding the problem statement		Using the environment and language		Learning from the problem	
	lower	substantial	lower	substantial	lower	substantial
Novice	3	13	5	11	1	15
Not novice	5	5	7	3	5	5

Table 8-7 Programming Experience versus Mental Effort (2 categories)

Fisher exact tests were performed on this data with the following results (Table 8-8):

	p	Statistical significance
Understanding the problem statement	0.1077	Not significant
Using the environment and language	0.0633	Not significant
Learning from the problem	0.0184	Significant ($p < 0.05$)

Table 8-8 Programming Experience versus Mental Effort Analysis (2 categories)

From these results, the null hypothesis H_0 is rejected. Participants with lower levels of prior programming experience have returned significantly higher levels of germane mental effort, for “learning from the problem”. These participants have also displayed a trend (at the 0.1 level) for expending higher levels of extraneous mental effort on “using the environment and language”, and this may represent a real effect.

These results are consistent with the comments from academics teaching novice programming, which reported that students without prior programming experience are likely to expend higher levels of mental effort.

8.4.6 Hypothesis 3 (*General Computing Literacy versus Mental Effort*)

- ❖ H_{A3}: Participants with lower levels of general computer literacy will be more likely to experience higher levels of mental effort.
- ❖ H₀₃: Participants with lower levels of general computer literacy will not be more likely to experience higher levels of mental effort.

Analysis

Participants were asked to rate their skill level in “using a computer as a tool (for example, for researching using the internet, and producing assignments)”, on a 9 point Likert scale from “new to computers” to “can use computers for advanced tasks, and format professional documents”. Not surprisingly, as these students were pursuing a computer programming course, all participants rated themselves as at least 4 on the scale. To analyse the level of computing skill versus performance and cognitive load, participants were grouped into Low (1-3), Medium (4-6) and High (7-9) computing skill levels. There were no participants in the “Low” group. The relationship between performance and computer literacy is shown in Figure 8-3.

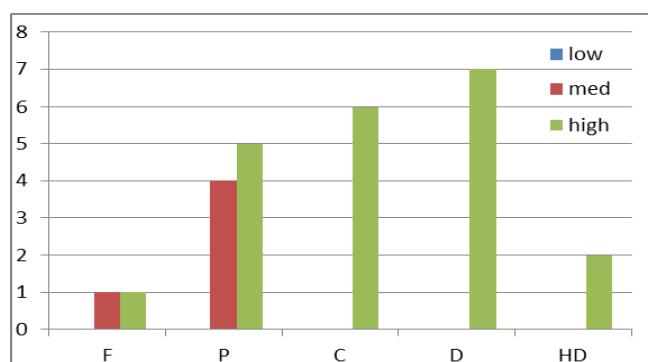


Figure 8-3 Computer Literacy versus Grade

The number of students in each mental effort category, mapped against computer literacy is shown below in Table 8-9:

Literacy	Understanding the problem statement			Using the environment and language			Learning from the problem		
	low	medium	high	low	medium	high	low	medium	high
Medium	0	3	2	0	3	2	0	2	3
High	5	12	4	9	9	3	5	11	5

Table 8-9 Computer Literacy versus Mental Effort

Of note is that the lowest mental effort score across all measures for those participants that reported they had ‘medium’ computing skills (e.g.: “some experience in the internet and producing documents”) was 4. These students were experiencing considerable cognitive load. In contrast, most of the students who reported they had ‘high’ levels of computer literacy experienced low to medium cognitive load. Fisher Exact Tests were performed on the data set, with the following results (Table 8-10):

	p	Statistical significance
Understanding the problem statement	0.4445	Not significant
Using the environment and language	0.1262	Not significant
Learning from the problem	0.3565	Not significant

Table 8-10 Computer Literacy verses Mental Effort Analysis

Visual inspection of the data revealed a similar pattern of mental effort scores to those in Section 8.4.5 above, and so the data was given a similar treatment: “lower” mental effort (1 to 4 on the scale) and “substantial” mental effort (5 or above). The results are given in Table 8-11 below:

Literacy	Understanding the problem statement		Using the environment and language		Learning from the problem	
	lower	substantial	lower	substantial	lower	substantial
Medium	1	4	0	5	0	5
High	7	14	12	9	6	15

Table 8-11 Literacy versus Mental Effort (2 categories)

Fisher exact tests were performed on this data with the following results (Table 8-12):

	p	Statistical significance
Understanding the problem statement	0.5024	Not significant
Using the environment and language	0.0304	Significant ($p < 0.05$)
Learning from the problem	0.2356	Not significant

Table 8-12 Computer Literacy verses Mental Effort Analysis (2 categories)

From these results, the null hypothesis H_03 is rejected. Participants with lower levels of computer literacy have returned significantly higher levels of extraneous mental effort, for “using the environment and language”.

Participants who had lower levels of prior computer literacy had to expend significantly more mental effort on navigating and using the language and environment than those students who had high levels of computer literacy. Cognitive Load Theory argues that higher levels of mental effort in extraneous activities such as using the environment and language may lead to cognitive overload with accompanying reduced performance in learning. The lower grade performance of the participants with lower (in this case “medium”) computer literacy as opposed to those with higher computer literacy is consistent with Cognitive Load Theory.

8.4.7 Hypothesis 4 (First Language verses Mental Effort)

- ❖ H_{A4}: Participants studying in a second language will be more likely to experience higher levels of mental effort.
- ❖ H₀₄: Participants studying in a second language will not be more likely to experience higher levels of mental effort.

Analysis

Programming instructors in the Australian Universities study (Chapter 7) had indicated that they thought that students from a non-English-speaking background experienced higher levels of cognitive load than native English speakers. For this reason, participants were asked to indicate their first language. Nineteen participants had English as their first language, and the remaining 7 another language. (Note that the course in question was delivered in English.) English/Other languages were mapped against performance. The participants who had English as a second language tended to perform more poorly than the native English-speaking students (Figure 8-4)

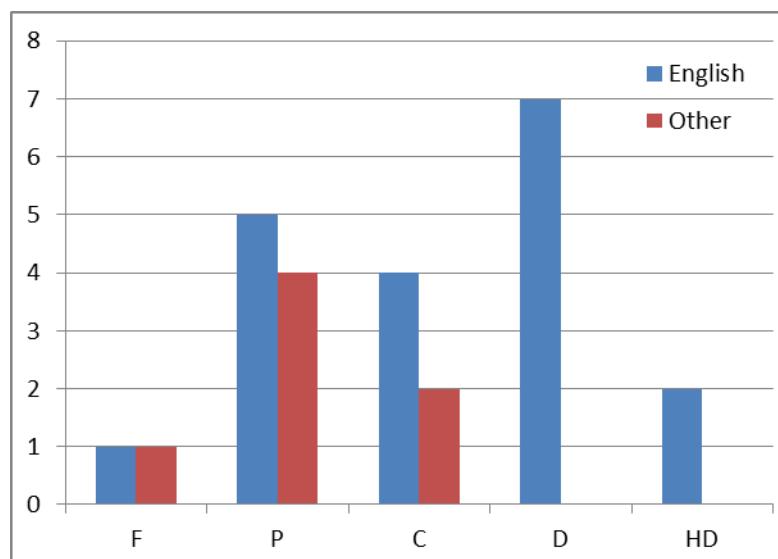


Figure 8-4 Language versus Performance

As before, mental effort measures were grouped into “low”, “medium” and “high” categories and mapped against first language. The number of participants in each category is shown below in Table 8-13.

Language	Understanding the problem statement			Using the environment and language			Learning from the problem		
	low	medium	high	low	medium	high	low	medium	high
English	5	8	6	9	6	4	5	8	6
Other	0	7	0	0	6	1	0	5	2

Table 8-13 Language versus Mental Effort

Fisher exact tests were performed on the data with the following results (shown in Table 8-14). From these results, it appears on first inspection that the null hypothesis H_04 is rejected for both intrinsic (understanding the problem) and extraneous (using the environment and language) levels of mental effort. This, however, is not the case.

	p	Statistical significance
Understanding the problem statement	0.0301	Significant ($p < 0.05$)
Using the environment and language	0.0247	Significant ($p < 0.05$)
Learning from the problem	0.3118	Not significant

Table 8-14 Language versus Mental Effort Analysis

The significant differences obtained in the contingency table do not indicate a directional change, but a “tightening” of central tendencies by those participants studying in a second language. Visual inspection of Table 8-13 indicates that the participants studying in a second language have clustered towards expending medium levels of mental effort. This was not expected.

From these results, the null hypothesis H_04 is accepted. Participants studying in a second language have not returned significantly higher levels in any of the three areas of mental effort.

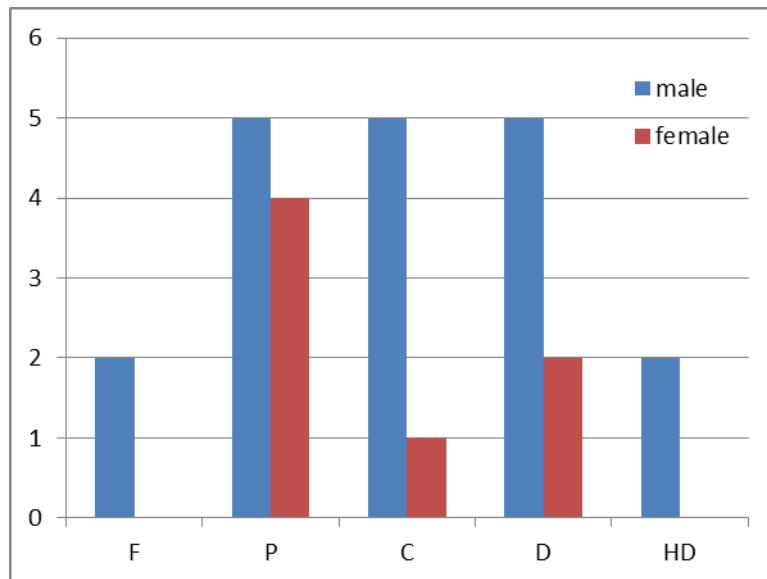
Participants without English as their first language experienced significantly different patterns of mental effort in both “understanding and processing the problem statement”, and “using the environment and language” compared to native English speakers, with clustering demonstrated to the “medium” scores. It is unknown whether this different pattern of scores for mental effort represents an effect based upon second-language features, or some other associated factor such as “culture”. It is interesting to note, however, that there were 5 different first languages represented in this group of 7 ‘other-language’ participants, so the result was not dependent on one particular cohort of culture or language. The clustering towards medium levels of mental effort was not expected, and warrants further investigation.

8.4.8 Hypothesis 5 (Gender versus Mental Effort)

- ❖ H_{A5}: Gender will be a factor determining levels of experienced mental effort.
- ❖ H₀₅: Gender will not be a factor determining levels of experienced mental effort.

Analysis

The 19 males and 7 females that completed the first half of the survey were analysed for relations between gender and performance/mental effort. Gender versus performance is shown in Figure 8-5:

**Figure 8-5 Gender versus Performance**

Again, mental effort measures were grouped and then mapped against gender of the participants. The number of participants in each category is show in Table 8-15:

Gender	Understanding the problem statement			Using the environment and language			Learning from the problem		
	low	medium	high	low	medium	high	low	medium	high
Male	4	13	2	6	9	4	4	10	5
Female	1	1	5	3	3	1	1	3	3

Table 8-15 Gender versus Mental Effort

Fisher exact tests were used to analyse the data with the following results (Table 8-16):

	p	Statistical significance
Understanding the problem statement	0.0061	Significant ($p < 0.05$)
Using the environment and language	>0.9999	Not significant
Learning from the problem	0.8478	Not significant

Table 8-16 Gender versus Mental Effort Analysis

From these results, the null hypothesis H₀₅ is rejected. Participants who are female have returned significantly higher levels of intrinsic mental effort, for “understanding the problem statement” than did males.

This may have been caused by the wording of problems and/or the context of the examples created by the male instructor, or may have another cause. Further research is needed in this area.

8.4.9 Hypothesis 6 (Age versus Mental Effort)

- ❖ H_{A6}: Age will be a factor determining levels of experienced mental effort.
- ❖ H₀₆: Age will not be a factor determining levels of experienced mental effort.

Analysis

Instructors of introductory programming courses reported that mature-age (over 25 years old) students were generally more successful than recent school-leaver students. Participants in this survey were asked for their age group. For the purposes of analysis, participants were grouped into categories of 25 years or less, and 26 years or older.

Age was mapped against final grade (Figure 8-6), and mental effort expended in each of the three areas (Table 8-17).

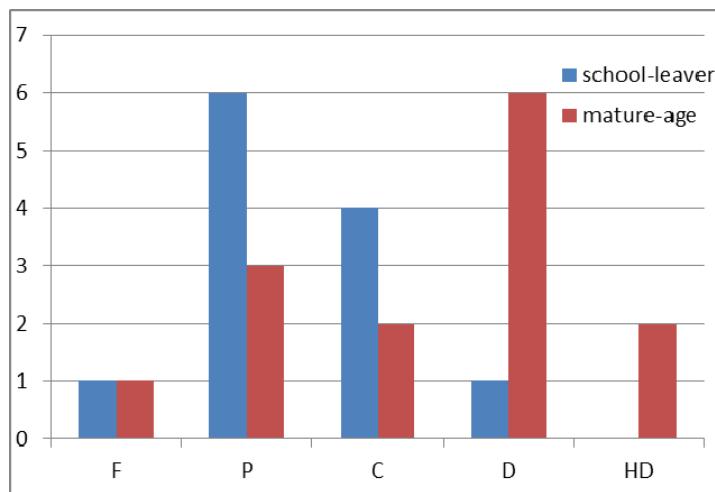


Figure 8-6 Age versus Final Grade

The number of students in each category is shown below (Table 8-17):

Age	Understanding the problem statement			Using the environment and language			Learning from the problem		
	low	medium	high	low	medium	high	low	medium	high
<= 25	4	13	2	6	9	4	4	10	5
26 +	1	1	5	3	3	1	1	3	3

Table 8-17 Age versus Mental Effort

Fisher exact tests were used to analyse the data and results are below in Table 8-18.

	p	Statistical significance
Understanding the problem statement	0.6580	Not significant
Using the environment and language	0.2850	Not significant
Learning from the problem	0.8756	Not significant

Table 8-18 Age versus Mental Effort Analysis

From these results, the null hypothesis H_06 is accepted. Participant's age is not associated with any of the three measures for mental effort.

There was no significant difference between age groups concerning the mental effort expended in each of the three areas, despite the difference in performance by visual inspection. One of the comments offered by the lecturer for this unit indicated that mature-age students tend to read the marking criteria provided more carefully than school-leavers, and produce assignment submissions better targeting these criteria, and hence gain relatively higher marks, all other factors being equal.

8.4.10 Hypothesis 7 (Study Mode versus Mental Effort)

- ❖ H_A7 : Study Mode will be a factor determining levels of experienced mental effort.
- ❖ H_07 : Study Mode will not be a factor determining levels of experienced mental effort.

Analysis

The performance of on-campus (domestic and international) participants was compared to that of external distance participants (Figure 8-7) and study mode was also mapped against mental effort expended in each of the three areas (Table 8-19).

The external students generally performed better than the on-campus students, although there is a confound with the age group, as all but one of the "school-leaver"

students were studying on-campus and all but one of the mature-age students were studying by distance study.

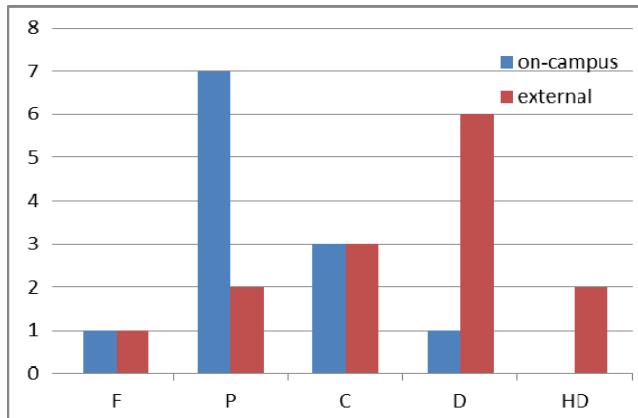


Figure 8-7 Study Mode versus Grade

The number of students in each category is shown below (Table 8-19):

Study Mode	Understanding the problem statement			Using the environment and language			Learning from the problem		
	low	medium	high	low	medium	high	low	medium	high
On-campus	2	9	1	4	5	3	2	7	3
External	3	6	5	5	7	2	3	6	5

Table 8-19 Study Mode versus Mental Effort

Fisher exact tests were performed on the data (results are below in Table 8-20) and there were no significant differences in mental effort measures between the on-campus and external students. From these results, the null hypothesis H_07 is accepted. Participant's mode of study is not associated with any of the three measures for mental effort.

	p	Statistical significance
Understanding the problem statement	0.2045	Not significant
Using the environment and language	0.8794	Not significant
Learning from the problem	0.8756	Not significant

Table 8-20 Study Mode versus Mental Effort Analysis

8.4.11 Hypothesis 8 (Year of Study versus Mental Effort)

- ❖ H_A8: Year of study will be a factor determining levels of experienced mental effort.
- ❖ H₀8: Year of study will not be a factor determining levels of experienced mental effort.

Analysis

From the 26 students that participated in the survey, only four were not in their first year of study. The number of participants therefore was too small to analyse for differences in mental effort between groups, and has not been analysed any further.

8.4.12 Changes in Mental Effort

Participants were asked about their mental effort in each of the three areas both at the beginning of semester, and again at the end of semester. The number of participants in each of the ‘increased’, ‘stayed the same’ and ‘decreased’ categories thorough the semester, mapped against final grade, is shown below in Table 8-21.

Grade	Understanding the problem statement			Using the environment and language			Learning from the problem		
	+ve	0	-ve	+ve	0	-ve	+ve	0	-ve
Pass/Credit	4	2	2	1	2	5	4	1	3
Dist./H. Dist.	5	2	2	5	4	0	4	4	1

Table 8-21 Change in Mental Effort versus Grade

Fisher exact tests were performed on this data with the following results (Table 8-22):

	p	Statistical significance
Understanding the problem statement	>0.9999	Not significant
Using the environment and language	0.0210	Significant (p < 0.05)
Learning from the problem	0.3204	Not significant

Table 8-22 Change in Mental Effort versus Grade Analysis

These results indicate a significant difference in the changes to extraneous mental effort, for “using the environment and language” through the semester based upon performance grade.

The mental effort involved in ‘understanding and processing the problem statement’ was expected to increase over the course of the semester, as more challenging problems were encountered. Generally this was the case. There was no significant difference in the change in mental effort, between participants that received passes/credits and those that received distinctions/high distinctions.

The mental effort involved in “using the environment and language” was expected to decrease over the session *if* there were no more complex parts of the environment introduced, as schemas were formed regarding navigation and use of the environment. Instead, what was observed was a significant difference between those participants that received passes and credits (decreased cognitive load) and those who received distinctions and high distinctions (increased cognitive load). One possible explanation is that students who experienced lower mental effort associated with navigating and using the environment and language early in the semester were more able to form schemas about the environment and language, and were hence more comfortable, as the course progressed, with exploring advanced features and topics, which required more difficult language constructs. This in turn would be anticipated to lead to enhanced performance in programming tasks within this environment, and hence lead to higher grades. However, this interpretation and explanation forms an area for further research. This is also consistent with the argument that less able students, already exerting relatively high levels of mental effort in the early parts of the semester, are not able to find many additional cognitive resources to expend on

the area of navigating and using the environment, as they are already approaching their ceiling.

Considering that there was no significant difference in mental effort change patterns between performance groups in either “intrinsic” (“understanding and processing the problem statement”) and “germane” (“learning from the problem”) dimensions, “navigating and using the environment and language” appears to be a key factor in defining different patterns of mental effort in participants, as the course progressed, which is clearly associated with end-of-course performance as demonstrated by end-of-course grades.

It should be noted that this result is not in accordance with the expectations of the study. The more highly performing students have *increased* their mental efforts in using the language and environment as the course progressed. In hindsight this may be explained by the fact that the course does use an increasing set of language features and tools as the course progresses. This highlights an aspect of programming courses not properly accounted for in the design of this study. As such additional features and tools become available, it may be that only the “more capable” students can focus on, learn, and use, these emerging aspects of the environment. Students who may be already “struggling” with content and concepts may effectively choose to ignore exploring and learning such additional features, and to consider them as “extension” activities which they choose to not pursue. There is no direct evidence for this, but most programming courses do move from relatively simple to more complex aspects, and it should be anticipated that there will be differential impacts upon students based upon how well they have, or have not, learnt features-to-date.

8.4.13 Student comments

On the second part of the survey, after answering questions regarding their level of mental effort in each area, participants were asked the open question “How do you think these three areas of mental effort affected you while studying in this unit?” (Note that the term “unit” is used in this university for “course”). A full list of comments is provided in Appendix D.5. A total of eighteen participants offered comments, with seven of these explicitly making reference to the stress associated with the unit. Ten of the eighteen participants offered negative comments on their struggles with the unit: e.g. “In the final weeks of this unit, I have been having such a hard time understanding the material that when attempting the assignments, I've either been in tears or drinking liquor. I've never felt more stupid” and from a different student; “I had my exam yesterday and was an emotional wreck, I felt like I couldn't breath [sic], I only answered half the questions. I am seriously wondering if this degree is worth all the stress, with my family life suffering”.

Half of the participants commented on their difficulty in writing code or understanding how to navigate the environment, including: “I feel the most amount of mental effort was spent on figuring out how to write the code to solve the problem”, “it was mainly a process of ascertaining the right syntax within which to frame the required logic statements” and “have some problems to understand codes”.

Most participants took the opportunity to make general comments about their experience of the unit, although some did comment on the three areas of mental effort. For the purposes of analysis, participants were divided into two groups – 5 students who experienced a *total* mental effort score of 14 or below (average of less than 5 on all measures) and 13 students who scored 15 and above. The numbers of students that made negative comments, the numbers of students that had difficulty understanding

the problem, and the numbers of students that specifically stated they had problems with the environment and language were then analysed with the following results (Table 8-23):

Mental effort	Negative comments		Difficulty understanding the problem		Difficulty with environment and language	
	negative	Not negative	yes	no	yes	no
Lower (3-14)	1	4	1	4	1	4
Higher (15+)	9	4	3	10	8	5

Table 8-23 Mental Effort versus Comments

The data was analysed using Fisher exact tests, with the following results (see Table 8-24). There were no statistically significant differences between groups, although a trend (at the 0.1 level) was present for participants reporting higher levels of mental effort to also report more negative comments. This may represent a real effect.

	p	Statistical significance
Negative comments	0.0882	Not significant
Difficulty understanding the problem	0.7010	Not significant
Difficulty with environment and language	0.1471	Not significant

Table 8-24 Mental effort versus Comments Analysis

8.5 Further Discussion

8.5.1 Limitations of study

The participation rate in this survey was disappointing, with only 28 participants, (2 of which were excluded), and only 18 participants who completed both halves of the survey. Despite the small number of participants, a series of

analyses were performed. It should be noted that this has been treated as an “exploratory” study seeking broad insight into the validity (or not) of the information that had been provided by academics in the survey reported in Chapter 7 of this thesis. As such, consideration has been given to differences returning p values that are less than 0.1, but not reaching the set alpha level of 0.05, and have been treated as trends possibly indicating real effects. There have also been a range of statistical tests performed on some data sets where re-defining various fulcrum points for groupings have resulted in insightful information regarding the results. There has been a deliberate strategy of not dividing alpha rates by the number of statistical tests performed. All of these actions have been undertaken to provide a means of seeking a level of insight to the underlying patterns and effects that may be present. Utilising this approach the analyses have yielded some statistically significant results, and several trends. Most of these are consistent with the comments gathered from introductory programming academics in Chapter 7.

Note that a substantial proportion of students who enrolled in this unit either withdrew or did not participate in the unit (the “ghosts” of the previous chapter). Their reasons for withdrawing or “ghost”ing cannot be determined without further information. These students were not represented in the student sample, and yet may have included important insights about those students in the bottom 10% of performance, which was part of the focus of the survey reported in Chapter 7.

The set of the participants used in this study, which generally excludes the very low-performing fail students, has still yielded several differences that are consistent with the comments gathered from introductory programming instructors reported in Chapter 7.

8.5.2 Summary

Although this was a small study limited by the number of participants, it could well form the skeleton for a more substantial study into how the factors of prior programming experience, computer literacy, first language, gender, maturity, study mode and year of study may impact upon mental effort expended, and performance attained, in an introductory computer programming course.

The findings of this survey concur with the assessments provided by the first-year programming instructors of their students in the areas of performance, programming experience and computer literacy. Also, differences were observed in the pattern of results for non-native-English speakers. The results are summarised in Table 8-25 below, where the symbol “√” indicates a significant difference between groups, and “(T)” denotes a trend that may represent a real effect (as suggested by p values attaining the 0.1 level):

Dimension	Grade	Programming Experience	Computer literacy	English as second language	Gender
“Understanding and processing the problem statement”				√	√
“Navigating and using the environment and language”	(T)	(T)	√	√	
“learning from the problem/reinforcing previous concepts”	(T)	√			

Table 8-25 Significant Differences in Groups

Female students experienced higher mental effort than male students when trying to understand the problem statement. This warrants further investigation. The “IT Girls” Alice and Mindstorms programs described in Chapter 5 and 6 were designed for female students, and possibly the design of the materials *for girls* had an effect on the

successful outcomes of the program. Another study is needed that includes both females and males in order to explore this aspect of the “IT Girls” workshop materials. The results of such a further study are reported in Chapter 9.

The participants who had lower general computer literacy experienced higher mental effort in navigating and using the environment and language, and higher (extraneous) mental effort in this area tended to indicate poorer grades.

Participants who had lower levels of programming experience indicated that they were expending high to extreme mental effort on learning from the problem, *at the same time* as they were expending high mental effort on understanding and processing the problem statement, and using the environment. Given the limits of human working memory, this is not possible. These students may have been attempting to express that their ‘extreme’ mental effort on learning represented ‘I’m trying as hard as I can’.

In addition, “navigating and using the environment and language” was responsible for significant differences in the *changes* in mental effort for students over the course of the semester that is moderated by grade. The experience of the high school students (who were reporting moderate levels of mental effort across all measures) was very different to these university students. Given that the environment and language appears to be a significant factor in the mental effort expended by these students, the utilisation of ‘simpler’ programming environments, as used in the high school outreach programs discussed in Chapters 5 and 6, may lower total cognitive load, especially for those students who have lower computer literacy, less programming experience, and those students who are less able. This hypothesis is explored in Chapter 9.

Chapter Nine: Controlled Environment Interface Experiment

9.1 Introduction

There are apparent inconsistencies between the levels of mental effort experienced in solving introductory programming problems, as reported by high school students involved in outreach programs, by academics teaching introductory programming in universities, and by some university students experiencing first programming units. Cognitive Load Theory suggests that these differences may be attributable to both the level of experience of the learner due to the effect of schemas, and to the instructional design and presentation of learning materials due to impositions of cognitive load. One of the factors in instructional design when considering introductory programming is the use of programming environments designed for learners, as used in the high school programs, rather than the professional programming environments as typically used in university programming courses.

A controlled experiment was therefore designed to investigate what effect the complexity of programming environment interface has on:

- the learning of programming concepts such as sequence, looping and event-driven actions (measured by test performance);
- the perception of the difficulty of programming; and
- student self-efficacy in introductory programming.

The experiment was also designed to test whether there are any gender differences in performance, as the high school outreaches have previously targeted female students only.

The experiment was held in a Queensland school, and utilised the Mindstorms NXT robot programming interface previously used in school outreach programs.

9.2 School Background

9.2.1 Introduction to the School

Victory College is a small (316 students) non-government K-12 school situated in Gympie in the Cooloola region of Queensland (Australian Curriculum Assessment and Reporting Authority, 2011). Victory College is co-educational, and its philosophy is based on Christian principles and a biblical worldview (Victory College, 2011). The school agreed to participate in the programming education research detailed in this chapter as part of an IT Careers Week at the school, held during August 2011. Although Victory College is outside Southern Cross University's feeder region, the opportunity was taken to access a sizeable set of participants for this research.

9.2.2 Demographics

Victory College is located in Gympie, a regional area with a population of approximately 49 300 (Queensland Treasury, 2011). Gympie has a higher than State average proportion of children (under 14) and people aged over 50. This could imply that residents leave the area for work reasons (in the age group 15 to 49) and move to the area when they retire.

The college has a lower than average ICSEA (Index of Community Socio-Educational Advantage) score, with 54% of students falling into the bottom quartile. However the college has a reputation for relative educational excellence and its students score above average on national testing measures such as the NAPLAN

(National Assessment Program – Literacy And Numeracy) assessment (Australian Curriculum Assessment and Reporting Authority, 2011).

The school has a relatively homogenous population, with only 1% of students coming from a language background other than English. The College has 52% female and 48% male students, so the ratio of genders of participants was anticipated to be similar.

9.2.3 Technology

The college prides itself on being technology-aware and in using technology to its maximum potential. Students are encouraged to engage with technology from school-entry age onwards. Most classrooms are enabled with interactive smart-boards, and there are several purpose-built computer labs on the college grounds, which are used extensively. The Resource Centre includes an interactive computer-equipped learning laboratory.

9.3 IT Careers Day

9.3.1 Overview

The research described in this chapter was situated as a part of a wider “IT Careers Day” to be held as part of the Australian Computer Society’s ICT Careers Week 2011. All students from Year 7 at Victory College were invited to participate in this IT Careers Day via a printed flyer (see Appendix E.1) and an information pack that was sent home with each student by the college. The information pack invited students to participate in an IT Careers Information Session and Alice programming workshops, as well as optional participation in Mindstorms Robots research sessions (this research) and Touch and Voice Research sessions which were being conducted by another researcher. Information sheets on both research sessions and associated consent forms were included in the information pack.

The information sheets (see Appendix E.2 for Mindstorms information sheet sample) included:

- an introduction to the researchers and the project;
- a description of the research;
- what the research involved;
- how the privacy and confidentiality of the participants would be protected;
- the participants' responsibilities in the research (participation was voluntary, how long it would take to participate, what they would be asked to do, and that they should inform the researchers if they wished to leave the research);
- the likelihood and form of dissemination of the research results, including publication;
- how to give consent to participate in the research.

Consent forms for the Mindstorms Robots research also included a pre-questionnaire to be completed by the student (see Section 9.4.4.1 for more details about this questionnaire).

9.3.2 Timetable for IT Careers Day:

Period	Tutorial Room (Mindstorms Workshops)	Mindstorms Test room	Touch & Voice Workshops
1	Workshop 1		Session 1
2	Workshop 2	Workshop 1 test	Session 2
recess		Workshop 2 test	
3	Workshop 3		Session 3
4	Workshop 4	Workshop 3 test	Session 4
5		Workshop 4 test	Session 5
lunch			
6	IT Careers talk followed by Alice workshops (held in Language Learning Centre)		
7			

Figure 9-1 IT Careers Day Schedule

The participants were randomly assigned to one of two treatment groups, being careful to have equal amounts of each gender in each group. The sequence of activities is presented in Figure 9-1.

An individual student who had chosen to participate in all activities may experience the following sequence of activities:

Up to 2 weeks prior to the IT Careers Day, requested to:

- complete consent forms for Mindstorms Robots research and Touch & Voice Research;
- complete permission note for IT Careers Day (see Appendix E.3);
- complete pre-IT Careers Day Questionnaire (as part of permission note for IT Careers Day);

On actual IT Careers Day, requested to:

- participate in Mindstorms Robot programming workshop;
- complete Mindstorms Robot workshop test;
- pause for refreshments provided by researcher;
- participate in Touch & Voice research session;
- attend a school period of normal class-work;
- pause for lunch;
- participate in IT Careers Session;
- participate in Alice Programming Workshop;
- complete post-IT Careers Day Evaluation/Questionnaire (see Section 9.4.4.2 for more details about this questionnaire).

9.3.3 Mindstorms Workshops

The Mindstorms Robots workshops were conducted in a 40 minute class period. The workshops were held in a tutorial room section of the Resource Centre in

the college, with 5 computers temporarily provided for the purposes of the workshop (see Figure 9-2). A laptop was also provided that was connected to a smart-board, for demonstrating the Mindstorms NXT software. More details about the Mindstorms workshops content and two treatment groups can be found in Chapter 5 and Section 9.4 respectively. After each Mindstorms workshop, the students were taken to a secluded courtyard area and given the appropriate test (see Section 9.4.4.3 and Appendix E.5 and E.6, Figure 9-3), then refreshments, then released to return to their classrooms.



Figure 9-2 Mindstorms Workshop Space



Figure 9-3 Mindstorms Test Space

9.3.4 Touch and Voice Research

At the same time as the Mindstorms research (with the exception of the first workshop of the day) some other research was being conducted in another location in the school, on Touch and Voice interfaces. This research did not involve programming, and is not discussed further in this work.

9.3.5 Careers Sessions

After lunch, all participants in the IT Careers Day (including those who had decided not to participate in Mindstorms research or Touch & Voice research) joined together for an IT Careers Session in the Language Learning Centre. This session included an introduction to the staff present, followed by a video on IT career opportunities, and then a discussion about IT career options and study pathways. Participants were given the opportunity to ask questions and were encouraged to interact with staff.

9.3.6 Alice Workshops

The Careers Information Session was followed by an Alice workshop. This was conducted in a similar way to the Alice workshops on the IT Girls Days held on the Coffs Harbour campus, and the IT Girls program held in Lightning Ridge. Students were led through a series of exercises demonstrating worked examples, and then applied these examples to their own custom Alice ‘worlds’.

An instructor demonstrated each small activity using a smartboard projector at the front of the room (see Figure 9-4). The students then completed that activity individually on their own computer. The activities moved from simple to more

complex activities as the workshop progressed. At the end of the Alice workshop, a post-workshop questionnaire was given to all students (see Section 9.4.4.2).



Figure 9-4 Instructor demonstrating worked example in the Alice workshop

9.4 Methodology

9.4.1 Structure

There were two different treatments used in this study. Group Subset was presented with an interface for Mindstorms that was identical to that used in the previous deliveries of the IT Girls studies reported in Chapters 5 and 6. Group Complete was presented an interface for Mindstorms that was ‘expanded’ to provide a wider offering of programming blocks/tools.

The focus of this research was the hypothesised difference in both attitudes and performance that would result between treatment groups based upon their being given variants of the Mindstorms interface. Each student was asked about their knowledge and attitudes towards IT and programming before the IT Careers Day and at the end of the day, after the Alice workshop. Each student was also asked about their three

sources of mental effort experienced during the Mindstorm workshop, and was given a performance test directly after the Mindstorms workshop (see Figure 9-5).

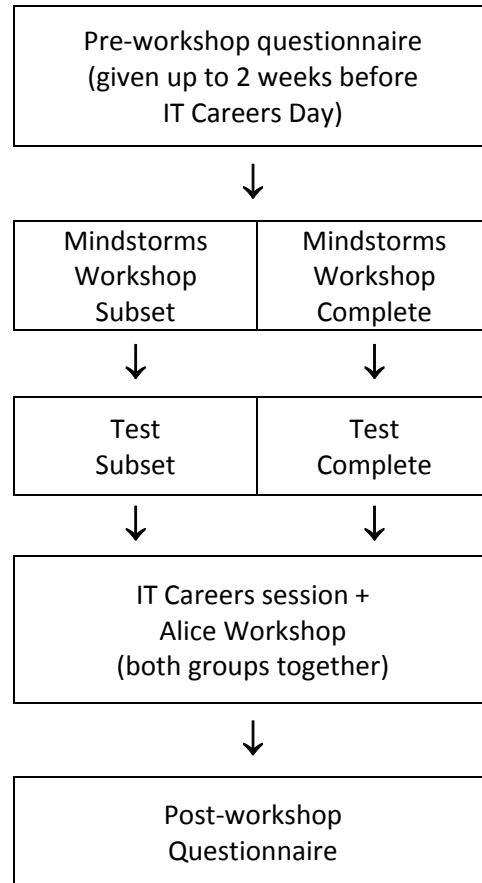


Figure 9-5 Experiment structure

9.4.2 Mindstorms Workshop Instructional Design

The Mindstorms workshops again followed the format set in the first IT Girls Days program. Students worked either singly or in pairs, with a Mindstorms Robot and a computer with the Mindstorms NXT software installed. Students were seen in groups of a maximum of 10 students in a workshop.

The design of the workshops again used a series of activities which each consisted of the author demonstrating a worked example using the Mindstorms software displayed on the smartboard, followed by students reproducing the demonstrated task on their own robot. The activities worked through programs built

using the structures of sequence and loops, as well as events (using sensors). For more details of the instructional design of the Mindstorms workshops and the underpinning relevant Cognitive Load Theory principles please refer to Chapter 5.

9.4.3 Treatment Groups

The two groups for this experiment (Group Subset and Group Complete) completed the same sequence of activities, were given the same amount of time to complete the activities, and used the same robots. The only difference between these two groups was the interface used for the Mindstorms NXT software.

As previously described, the Mindstorms NXT software uses programming blocks to build programs using drag and drop placement on a graphical timeline. The default palette of blocks includes the most commonly used programming blocks and is the first palette available when a new program is created for the first time (see Figure 9-6). Most programming blocks are available on the left of the screen, with one group of programming blocks situated in a slide-out sub-palette. In this chapter this interface containing the common programming blocks will be designated as the “Subset” interface.

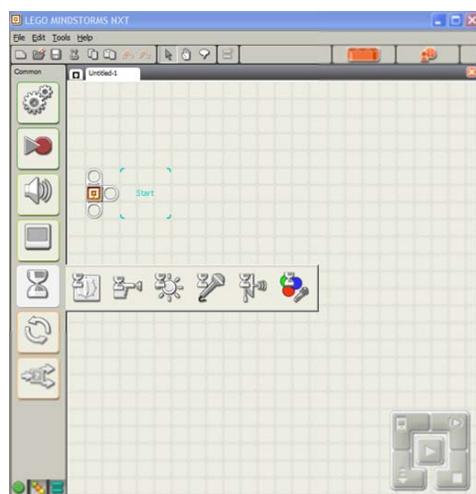


Figure 9-6 Common tools - "Subset" interface (expanded)

It is not immediately obvious that alternate palettes are available for use, as the tabbed interface for accessing other palettes is physically small and situated at the bottom left of the screen (see Figure 9-7). The author had noted previously in other workshops with school-aged students, as well as workshops with university students, that no participants had discovered the other palettes in sessions of up to 1½ hours duration.

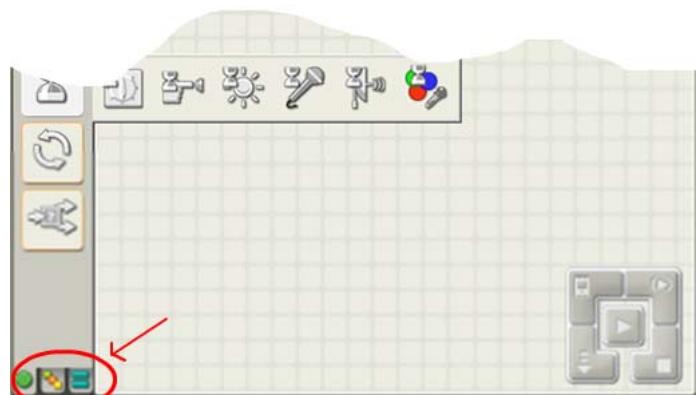


Figure 9-7 Palette tabs

The software can also be configured so that a more comprehensive set of programming blocks are available automatically when a new program is created. This more comprehensive palette groups most of the common blocks available in the Subset interface into one sub-palette, but also repeats these blocks and adds new blocks in sub-palettes accessed through side icons/buttons (for examples, see Figure 9-8 and Figure 9-9).

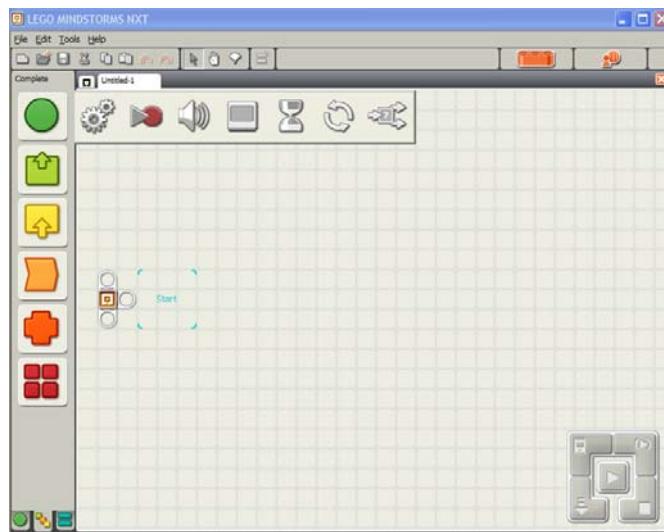


Figure 9-8 Complete interface with "common" sub-palette expanded

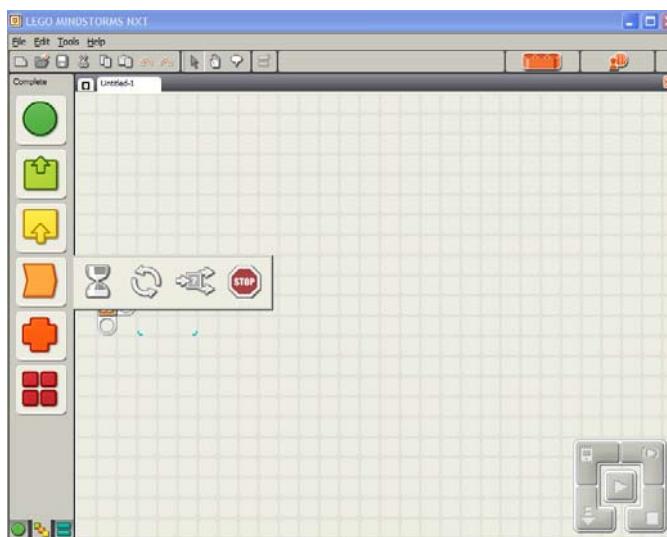


Figure 9-9 Complete interface showing sub-palette with loop and wait blocks

The user has a choice of more programming blocks in this general palette – from now on referred to as the “Complete” interface to distinguish it from the Subset interface.

Note that the buttons are the same size in each interface, and buttons that have the same functionality have the same appearance.

It was decided to use these two interfaces to test whether the relative complexity (in this case, the “completeness” of the number of elements available in the interface) of a programming environment has an effect on learning basic

programming concepts, for novices, even if the ‘additional’ features are not used or referenced.

The available participants were divided into two groups (Subset and Complete), with equal numbers of each gender in each group.

Group Subset used the Subset interface throughout their workshop.

Group Complete used the Complete interface throughout their workshop.

Both groups used the same programming blocks, and completed the same activities, in the same amount of time. Even though Group Complete had a greater number of blocks available via the palette onscreen, they were not directed to use any of these extra blocks, in any of the worked example activities or free programming time.

Group Complete used the same subset of blocks that Group Subset used in their workshop. The extra blocks were not referenced or explained in any way.

After each workshop, the participants were given an equivalent test dependent on their treatment group (see Section 9.4.4.3). The tests were timed.

9.4.4 Instruments

9.4.4.1 Pre-IT Careers Day questionnaire

As part of the IT Careers Day permission note (see E.4), each participant was asked some questions to ascertain their level of computer literacy and programming experience, their knowledge of IT, whether they were considering a career in IT, their perception of the difficulty of programming and confidence with programming (self-efficacy).

Nine-point Likert scales were used and language was chosen to which high-school students would relate. For example, students were asked to indicate how much

they agreed with the statement “I think programming generally is difficult”, where 1 was “no way” and 9 was “totally!”

Students were also asked for their name, age and school year, so that their pre-workshop questionnaire, post-workshop questionnaire and test results could be matched.

9.4.4.2 Post-IT Careers Day questionnaire

After the Alice workshop, at the end of the day, students were again asked about their level of knowledge of IT, whether they were considering a career in IT, their perception of the difficulty of programming and their confidence with programming (see Appendix E.7 for sample post-workshop questionnaire). Students were also asked about how interesting they found programming with Alice 3D worlds and robots, and how difficult they found programming with Alice and programming robots. Again, nine-point Likert scales were used to capture this data. Students were asked for their names and years, for matching purposes.

Participants were also given the two open questions: “what did you enjoy most about the day?” and “what could we have done better?”

9.4.4.3 Mindstorms Test

9.4.4.3.1 Design of Tests

The Mindstorms test was designed to test recall of the purpose of various programming blocks, the building of schema about the environment/interface used, near transfer of programming construct concepts as well as far transfer of concepts such as sequence, looping and events. The test was given in written format. The test instrument was also used to collect data about the level of mental effort used in each

of the three aspects of completing an activity in the Mindstorms workshop. The questions were:

- “How much mental effort did you need to use to understand what you had to do in each exercise?” [emphasis in original];
- “How much mental effort did you need to use to navigate and use the NXT programming software?”;
- “How much mental effort did you need to use to learn and understand concepts?”.

Students were then asked to describe briefly the purpose of each of three programming blocks when used in a Mindstorms program: a “wait until touch sensor is activated” block, a “play sound” block and a “loop” block.

To test the creation of schemas regarding the interface/environment, students were given a sheet with a screenshot of the Mindstorms interface, with sub-palettes expanded, and all blocks blanked-out. They were then given six programming blocks that had been used in the activities and asked to indicate where they should be placed on the screenshot, by writing the letter of the block on the screenshot. An example was given, using another programming block.

Students were then given one similar multiple choice problem and four near-transfer multiple choice problems which asked the student to choose between screenshot options for various programs. An example is given below in Figure 9-10.

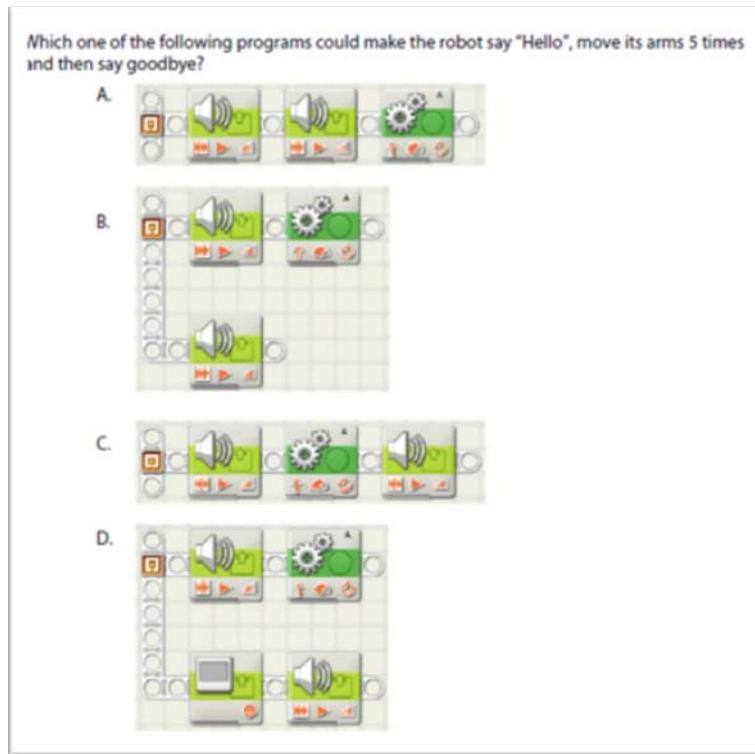


Figure 9-10 Mindstorms Test - Example multiple-choice question

Participants were also given a section of far-transfer questions. The participants were given a series of physical ‘program blocks’ showing dog tricks, such as “Bark”, “Fetch a ball”, “Lay Down” and “Point” (see Figure 9-11). Students were also given several “loop” constructs (Figure 9-12), several “wait” blocks (Figure 9-13) and two pieces of paper with timeline symbols (for example, see Figure 9-14), on which to build their ‘programs’. The loop, wait and timeline symbols were not the same symbols as had been used in the Mindstorms NXT interface.

These questions were designed to test the understanding of the participants of some programming constructs. However the surface cues of these questions were very different to the Mindstorms robots. The object being described (a dog) was different to the robots previously used, the questions involved everyday activities rather than technology-based activities, and the method of building the ‘program’ (physical manipulations rather than drag-and-drop on a computer with a mouse) was very

different. Ideal solutions to these tasks, however, carry aspects of these programming constructs such as sequence, events and iteration.

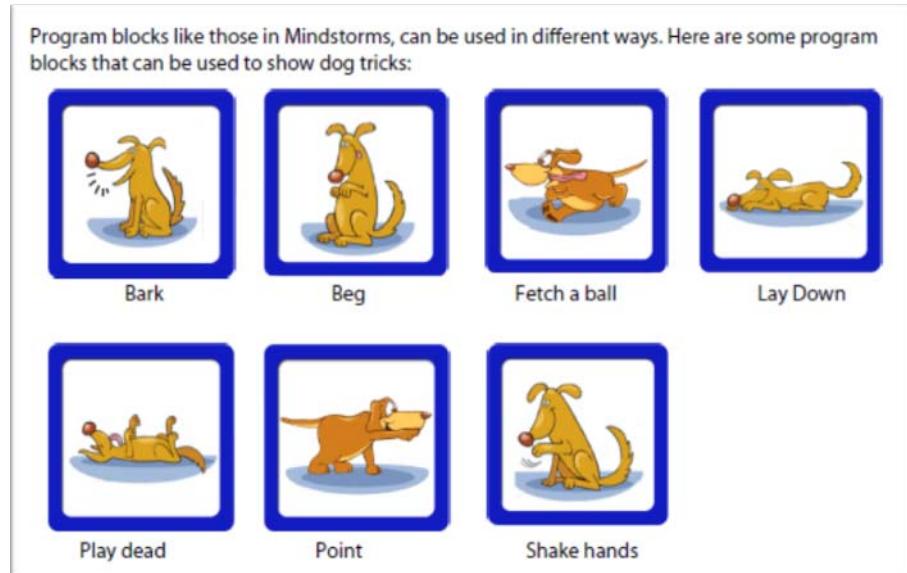


Figure 9-11 Dog trick 'program blocks'

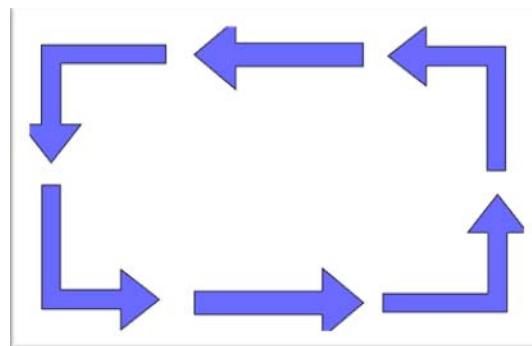


Figure 9-12 Dog trick "Loop" construct



Figure 9-13 Dog trick "Wait" construct



Figure 9-14 Dog trick "timeline"

The physical blocks were printed paper, covered in plastic contact. Participants were given ‘blu-tac’ removable adhesive and asked to stick the programming blocks onto the timeline to make the following dog tricks:

- “dog fetches a ball twice, points, and then begs” [sequence and optional loop];
- “dog shakes hands 5 times and then plays dead” [combination of sequence and loop preferred, longer sequence is an acceptable answer];
- “dog lays down and waits until told to fetch a ball, then barks 3 times” [combination of sequence, event and a loop preferred; longer sequence and event is an acceptable answer].

Participants were given a pen, the programming blocks including loops and waits, ‘blu-tac’, and the printed test - including sheets for completing the dog trick exercise – face-down on a desk (see Figure 9-15). Each participant was given enough blocks to complete all exercises with or without using loops, and was given extra blocks of each type in order to not influence the choice of solution.



Figure 9-15 Desktop ready for Mindstorms test

Participants were informed that they should do the test as quickly as possible, but also as accurately as possible, and raise their hand when they had completed the test. The instructor would then provide them with the current time, which they would record on the test paper.

9.4.4.3.2 Group Test Differences

Group Subset and Group Complete were given an identical test with the exception of the question regarding the rebuilding of the interface. Group Subset was given a blanked, expanded screenshot of the Subset interface (see Figure 9-16) and asked to identify the positions for the following blocks:

- wait for touch sensor

- loop
- play sound
- wait (timer)
- move motors
- wait for sound sensor.

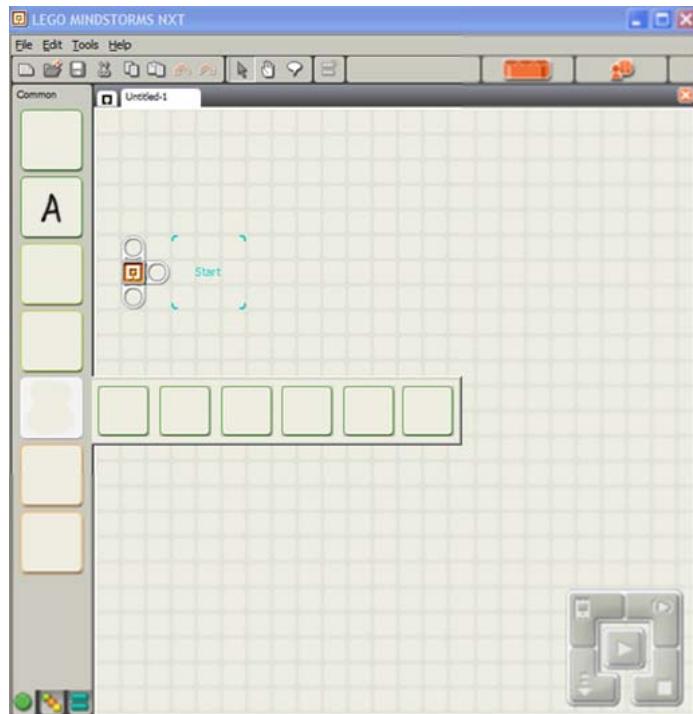


Figure 9-16 Subset blanked interface (with sample answer marked)

Group Complete was given a blanked, expanded screenshot of the Complete interface (see Figure 9-17) and asked to identify the positions for the same blocks as Group Subset. The possible answers for the Complete interface differed from the possible answers for the Subset interface in that some of the blocks were available in more than one location in the Complete interface only. Both groups had used each of the named programming blocks at least twice in the workshop activities, so were expected to be familiar with the location of the blocks.

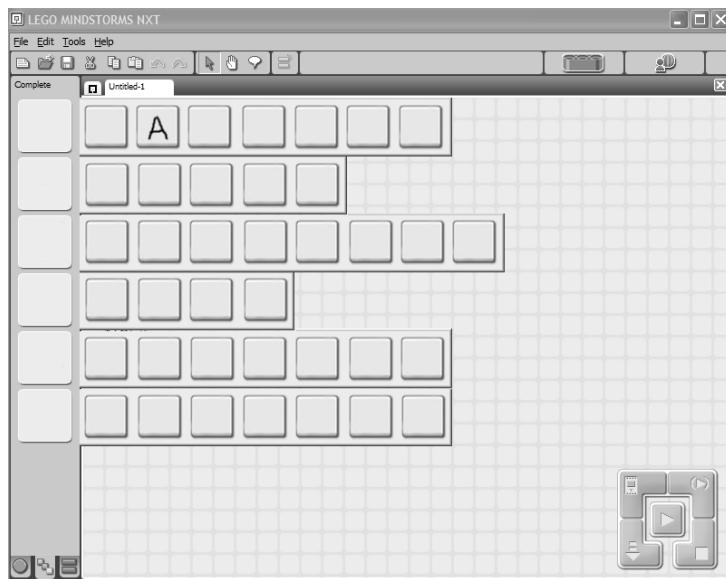


Figure 9-17 Complete blanked interface - expanded

Apart from this interface question, the two tests were identical.

9.5 Hypotheses

The hypotheses that were to be tested in this study are listed below:

- ❖ **H1 (Career Aspirations):** Participants will be more likely to consider a career in Information Technology after the workshop than before the workshop.
- ❖ **H2 (IT Knowledge):** Participants will report increased knowledge of Information Technology after the workshop than before the workshop.
- ❖ **H3 (Programming Difficulty):** Participants will perceive computer programming to be less difficult after the workshop than before the workshop.
- ❖ **H4 (Self-efficacy):** Participants will perceive their self-efficacy in computer programming to be higher after the workshop than before the workshop.
- ❖ **H5 (Test Completion Time):** Participants from Group Subset will take less time to complete the Mindstorms test than participants from Group Complete.
- ❖ **H6 (Test Score):** Participants from Group Subset will have a higher overall score for the Mindstorms test than participants from Group Complete.

- ❖ **H7 (Interface Schema Acquisition):** Participants from Group Subset will have a higher score for rebuilding the interface test question than participants from Group Complete.
- ❖ **H8 (Knowledge Acquisition):** Participants from Group Subset will have a higher score for the knowledge acquisition part of the test than participants from Group Complete.
- ❖ **H9 (Difficulty of Mindstorms Programming):** Participants from Group Subset will have a lower perceived difficulty of Mindstorms Robots programming than participants from Group Complete.
- ❖ **H10 (Difficulty of Alice Programming):** Participants from Group Subset will have a lower perceived difficulty of Alice programming than participants from Group Complete.
- ❖ **H11 (Cognitive Load):** Participants from Group Subset will report a different pattern of mental effort scores to participants in Group Complete.
- ❖ **H12 (Gender Difference - Performance):** There will be no difference between female and male participant performance in the Mindstorms test.
- ❖ **H13 (Gender Difference - Attitudes):** There will be no difference between female and male participants in the change in career aspirations and IT knowledge from before the workshop to after the workshop.

Recall that for all statistical analysis, a significance level of $p = 0.05$ is used throughout this thesis.

9.6 Results and Discussion

9.6.1 Age, Computer Literacy and Programming Experience

The participants ranged in age from 11 to 13 with the majority aged 12 years (mean=11.8, $s = 0.634$). All indicated that they had an average level of computer knowledge (median = 5 on a 9 point Likert scale), and all but two students reported that they had little to no programming experience (median = 2 on a 9 point Likert scale where 1 = no programming and 9 = ‘lots’).

9.6.2 Homogeneity of Groups

There was no significant difference in participant age, computer literacy, programming experience, knowledge of IT and intent to pursue a career in IT between Group Subset and Group Complex, before the workshop. A t-test on age returned no significant difference ($\text{Mean}(\text{Subset}) - \text{Mean}(\text{Complete}) = -0.08$; $t = -0.3$; $df = 24$; $p = 0.7667$), and Mann-Whitney U tests on computing skill, programming experience, knowledge of IT and intent to pursue a career in IT all returned no significant difference between Group Subset and Group Complete, based upon Subset ranks minus Complete ranks (see Table 9-1).

	U_A	z	p
Computer Literacy	87	0.1	0.46
Prog. Experience	97	-0.62	0.27
IT Knowledge	79	0.26	0.4
Career Intent	83.5	0.03	0.49

Table 9-1 Homogeneity between groups

9.6.3 Hypothesis 1 (Career Aspirations)

H_A1 : Participants will be more likely to consider a career in Information Technology after the workshop than before the workshop.

H_01 : (Null Hypothesis) Participants will show no change in their likelihood to consider a career in Information Technology after the workshop than before the workshop.

Analysis

In the pre-IT Careers Day questionnaire, participants were asked to indicate how much they agreed with the statement "I am considering a career in IT" on a 9 point Likert scale (where 1 = 'no way' and 9 = 'totally!'). After completing both of the programming workshops, the participants were asked the same question as part of the post-questionnaire. It was anticipated that – in accordance with the previous IT Girls Day remote school workshops (see Chapter 6) – all participants would be more likely to consider a career in IT after the workshops, as a result of the positive experiences, and career information given to them.

Although there were 17 participants in Group Subset and 15 participants in Group Complete for the actual Mindstorms workshops and test, not all participants completed both the pre- and post- questionnaires. The pre- and post- answers of the 12 participants in each group who did complete both questionnaires were analysed using the Wilcoxon Signed Ranks test.

There was a significant difference in the participants' intents to consider a career in IT after the workshop than before the workshop [Wilcoxon Signed Rank test: $n = 24$, $W = 128$, $N_{s/r} = 19$, $z = 2.57$, $p = 0.0051$].

From these results, the null hypothesis H_01 can be rejected in favour of the alternative hypothesis, that participants did report higher likelihood to pursue a career in Information Technology after the workshop than before the workshop.

Note that both groups had significantly higher intents to consider a career in IT after the workshops than before the workshops (Table 9-2).

Career Intent	n	W	N _{s/r}	z	p
Group Subset	12	29	9	--	p<0.05
Group Complete	12	38	10	1.91	0.0281

Table 9-2 Wilcoxon Signed Rank - Career Intent Day 1

This is in agreement with the results for IT Girls Day workshops at Lightning Ridge, which were conducted using the Subset interface and with female-only participants.

9.6.4 Hypothesis 2 (IT knowledge)

H_{A2}: Participants will report increased knowledge of Information Technology after the workshop than before the workshop.

H₀₂: (Null Hypothesis) Participants will report no change in their knowledge of Information Technology after the workshop than before the workshop.

Analysis

In the pre-IT Careers Day questionnaire, participants were asked to indicate how much they agreed with the statement "I know a lot about Information Technology" on a 9 point Likert scale (where 1 = 'definitely not' and 9 = 'I know heaps!'). After completing both of the programming workshops and the IT careers session, the participants were asked the same question as part of the post-questionnaire. It was anticipated that – in accordance with the previous IT Girls Day remote school workshops (see Chapter 6) – all participants would report higher knowledge of Information Technology after the Careers Day when compared to before the Careers Day, despite the emphasis on technology within the college.

The pre- and post- answers of the 12 participants in each group who did complete both questionnaires were analysed using the Wilcoxon Signed Ranks test.

There was a significant difference in the perceived IT knowledge after the workshop than before the workshop [Wilcoxon Signed Rank test: $n = 24$, $W = 179$, $N_{s/r} = 21$, $z = 3.1$, $p = 0.001$].

From these results, the null hypothesis H_02 can be rejected in favour of the alternative hypothesis, that participants did report higher knowledge of Information Technology after the workshop than before the workshop.

In addition, both groups reported significant higher knowledge about IT after the workshops than before the workshops (see Table 9-3).

IT Knowledge	n	W	$N_{s/r}$	z	p
Group Subset	12	39	10	1.96	0.025
Group Complete	12	55	11	2.42	0.0078

Table 9-3 Wilcoxon Signed Rank - IT Knowledge Day

Again, this is in agreement with the results for IT Girls Day workshops at Lightning Ridge, which were conducted using the Subset interface and with female-only participants.

9.6.5 Hypothesis 3 (Programming Difficulty)

H_A3 : Participants will perceive computer programming to be less difficult after the workshop than before the workshop.

H_03 : (Null Hypothesis) Participants will have no change in their perception of the difficulty of computer programming after the workshop than before the workshop.

Analysis

In the pre-IT Careers Day questionnaire, participants were asked to indicate how much they agreed with the statement "I think programming generally is difficult" on a 9 point Likert scale (where 1 = 'no way' and 9 = 'totally!'). After completing

both of the programming workshops, the participants were asked the same question as part of the post-questionnaire.

From the previous experience of both the on-campus IT Girls Day workshop and the IT Girls Day remote school workshops it was anticipated that all participants would report a perception of programming as less difficult after the Careers Day when compared to before the Careers Day.

Answers from all participants were analysed using a Wilcoxon Signed Rank test, and there was found to be a significant difference in participants' perception of the difficulty of programming towards "less difficult", from before the workshops to after the workshops [$n = 24$, $W = 128$, $N_{s/r} = 19$, $z = 2.57$, $p = 0.0051$].

From these results, the null hypothesis H_0 can be rejected in favour of the alternative hypothesis, that participants did perceive computer programming to be less difficult after the workshop than before the workshop.

It was anticipated that Group Subset may have a greater shift in perception of difficulty in the direction of "easier" than Group Complete, who were presented with the complete interface in the Mindstorms workshops.

Pre-workshop and post-workshop answers from the 12 participants in Group Subset and 12 participants in Group Complete who completed the Mindstorms workshops and Alice workshops and completed both questionnaires were then analysed individually using Wilcoxon Signed Rank tests. The results are shown in Table 9-4 below:

	n	W	$N_{s/r}$	z	p
Group Subset	12	-55	11	-2.42	0.0078
Group Complete	12	-29	10	-1.45	0.0735

Table 9-4 Wilcoxon Signed Rank - Difficulty of Programming

Group Subset had a significantly different perception of the difficulty of programming after the workshop compared to before the workshop, in the direction of ‘easier’.

Although Group Complete displayed shifts downwards, this group did not experience a significant change in their perception of the difficulty of programming.

9.6.6 Hypothesis 4 (Self-efficacy)

H_A4 : Participants will perceive their self-efficacy in computer programming to be higher after the workshop than before the workshop.

H_04 : (Null Hypothesis) Participants will have no change in their self-efficacy in computer programming after the workshop from before the workshop.

Analysis

In the pre-IT Careers Day questionnaire, participants were asked to indicate how much they agreed with the statement "I feel confident with programming" on a 9 point Likert scale (where 1 = ‘no way’ and 9 = ‘totally!’). After completing both of the programming workshops, the participants were asked the same question as part of the post-questionnaire.

The researcher’s experience of both the on-campus IT Girls Day workshop and the IT Girls Day remote school workshops indicated that it was likely that participants would report increased self-efficacy in computer programming as a result of the IT Careers Day. However it was anticipated that Group Subset may have a greater increase in self-efficacy in programming than Group Complete.

Answers from both groups were analysed together using Wilcoxon Signed Rank tests and participant’s self-efficacy was significantly higher after the workshops than before the workshops [$n = 24$, $W = 119$, $N_{s/r} = 18$, $z = 2.58$, $p = 0.0049$]. The null

hypothesis H_04 can be rejected in favour of the alternative hypothesis, that participants did have higher self-efficacy after the workshop than before the workshop.

Pre- and post-workshop answers from the 12 participants in Group Subset and 12 participants in Group Complete who completed the Mindstorms workshops and completed both questionnaires were then analysed separately using Wilcoxon Signed Rank tests and the results are shown in Table 9-5 below:

	n	W	N_{s/r}	p
Group A	12	42	9	p < 0.01
Group B	12	18	9	p > 0.05

Table 9-5 Wilcoxon Signed Rank - Self-efficacy

Group Subset had a significantly higher self-efficacy in programming after the workshop compared to before the workshop. Although Group Complete displayed some shifts upwards in self-efficacy, this group did not experience a significant change.

9.6.7 Hypothesis 5 (Test Completion Time)

H_A5 : Participants from Group Subset will take less time to complete the

Mindstorms test than participants from Group Complete.

H_05 (Null Hypothesis) There will be no difference in the time taken to complete the Mindstorms test between participants in Group Subset and Group Complete.

Analysis

The supervisor of the test recorded the time that all participants in a group test began the test phase. Participants were asked to raise their hand when they had completed the test, and the supervisor would then provide them with the current time,

which they would record on the test paper. The start time was subtracted from this recorded time to get the elapsed time [in seconds] for the test for each participant.

It was anticipated that participants from Group Subset would take less time to complete the test than participants in Group Complete. The times from the 17 participants in Group Subset and 15 participants in Group Complete were compared using a t- test, and participants from Group Subset were found to take significantly less time to complete the test than Group Complete [$\text{Mean}_{\text{Subset}} - \text{Mean}_{\text{Complete}} = -150$ seconds; $t = -1.91$, $df = 30$, $p = 0.0349$].

From these results, the null hypothesis H_05 can be rejected in favour of the alternative hypothesis, that participants from Group Subset did take less time to complete the Mindstorms test than participants from Group Complete.

9.6.8 Hypothesis 6 (Test Score)

H_A6 : Participants from Group Subset will have a higher overall score for the Mindstorms test than participants from Group Complete.

H_06 : (Null Hypothesis) There will be no difference in the overall score for the Mindstorms test between Group Subset and Group Complete.

Analysis

There was a total maximum possible mark of 17 for the Mindstorms test. The marks were distributed as follows:

- 1 mark each for correct description of the purpose of programming blocks [total 3 marks];
- 1 mark each for correct placement of programming blocks on blanked interface [total 6 marks];
- 1 mark each for correct multiple choice answers [total 5 marks];

- 1 mark each for dog trick (far transfer) answers [total 3 marks].

Each student's test was marked and a total mark out of 17 recorded. The two groups' (Subset and Complete) mean marks, standard deviation, minimum and maximum are shown in Table 9-6.

	mean	st dev	min	max
Group Subset	11.03	2.60	6.5	15
Group Complete	8.83	1.88	6	14

Table 9-6 Test Score Totals

The total test scores from the 17 participants in Group Subset and 15 participants in Group Complete were compared using a t- test, and participants from Group Subset were found to have a significantly higher test score than Group Complete [$\text{Mean}_{\text{Subset}} - \text{Mean}_{\text{Complete}} = 2.20$; $t = 2.71$, $df = 30$, $p = 0.0055$].

From these results, the null hypothesis H_06 can be rejected in favour of the alternative hypothesis, that participants from Group Subset did have a higher overall score for the Mindstorms test than participants from Group Complete.

9.6.9 Hypothesis 7 (Interface Schema Acquisition)

H_A7 : Participants from Group Subset will have a higher score for rebuilding the interface test question than participants from Group Complete.

H_07 : (Null Hypothesis) There will be no difference in the scores for rebuilding the interface test question between participants from Group Subset and Group Complete.

Analysis

Students were asked to indicate where six programming blocks were placed on a blanked version of the Mindstorms interface. Each of these six programming blocks had been used in the Mindstorms workshop activities at least twice. It was anticipated

that students in Group Subset would more effectively build a schema for the positioning of the blocks in the interface over students in Group Complete, who would have more of their working memory capacity used by dealing with the distractors of the availability of extra blocks – even though those blocks were not being used or referenced in any of the activities.

The maximum score available on this question was 6, if the student placed all of the blocks in the correct positions. The Complete Interface makes several programming blocks available in more than one location, so if a student in Group Complete placed that particular block in any of the correct positions, they received a point for that block.

The two groups' (Subset and Complete) mean marks, standard deviation, minimum and maximum are shown in Table 9-7 below.

	mean	st dev	min	max
Group Subset	2.76	1.71	0	5
Group Complete	1.33	1.18	0	4

Table 9-7 Test Scores Interface Schema Acquisition

Participants from Group Subset scored significantly higher on rebuilding the interface than participants from Group Complete [t-test: $\text{Mean}_{\text{Subset}} - \text{Mean}_{\text{Complete}} = 1.43$, $t = 2.72$, $df = 30$, $p = 0.0054$].

From these results, the null hypothesis H_07 can be rejected in favour of the alternative hypothesis, that participants from Group Subset did have a higher score for rebuilding the interface test question than participants from Group Complete.

9.6.10 Hypothesis 8 (Knowledge Acquisition)

H_A8 : Participants from Group Subset will have a higher score for the knowledge acquisition part of the test than participants from Group Complete.

H_08 : (Null Hypothesis) There will be no difference in the score for the knowledge acquisition part of the test between Group Subset and Group Complete.

Analysis

The maximum score available for the Mindstorms test, excluding rebuilding the interface, was 11. The two group scores for this knowledge acquisition part of the test were compared using a t-test and although these results were not statistically significant [$\text{MeanSubset} - \text{MeanComplete} = 0.7647$, $t = +1.36$, $df = 30$, $p = 0.09$], the results trended in the expected direction.

Closer inspection of the test scores for Knowledge Acquisition showed that 9 of the 17 participants in Group Subset scored 9 or higher (from a possible 11), while in Group Complete, only 2 of the 15 participants scored 9 or above. This result is statistically significant ($p = 0.02$) using Chi-Square analysis. This indicates that more participants in Group Subset scored highly (at 9 or more marks out of a possible 11 marks) for Knowledge Acquisition than those participants in Group Complete.

From these results, the null hypothesis H_08 can be rejected in favour of the alternative hypothesis, that participants from Group Subset did have a higher score for the knowledge acquisition part of the test than participants from Group Complete. Specifically, more participants from Group Subset achieved a “high” level of knowledge (mastery), as defined by scoring at least 9 out of the possible 11 points available, than did participants in Group Complete.

9.6.11 Hypothesis 9 (Difficulty of Mindstorms Programming)

H_A9 : Participants from Group Subset will have a lower perceived difficulty of Mindstorms Robots programming than participants from Group Complete.

H₀9: (Null Hypothesis) There will be no difference in the perceived difficulty of Mindstorms Robots programming between participants from Group Subset and Group Complete.

Analysis

In the post-IT Careers Day questionnaire, participants were asked to complete the statement "Programming robots to do things is ... " on a 9 point Likert scale (where 1 = 'really easy' and 9 = 'really difficult'. It was expected that if having the extra (unused) programming blocks available in the complete interface were having an increasing effect on load placed on working memory while using the interface, then Group Complete would perceive programming robots as more difficult than Group Subset, even though both groups completed the same activities, with the same programming blocks.

The responses of the 15 participants in Group Subset and 14 participants in Group Complete who completed the Mindstorms workshops and completed the post-questionnaire were analysed for differences using a Mann-Whitney U Test. Participants in Group Complete found programming with the Mindstorms Robots significantly more difficult than participants in Group Subset [$U_A = 144$, $z = 1.66$, $p = 0.049$].

From these results, the null hypothesis H₀9 can be rejected in favour of the alternative hypothesis, that participants from Group Subset did have a lower perceived difficulty of Mindstorms Robots programming than participants from Group Complete.

This was an expected result, if the extra (unused) programming blocks were placing an additional load on working memory for the students in Group Complete.

9.6.12 Hypothesis 10 (Difficulty of Alice Programming)

H_A10 : Participants from Group Subset will have a lower perceived difficulty of Alice programming than participants from Group Complete.

H_010 : (Null Hypothesis) There will be no difference in the perceived difficulty of Alice programming between participants from Group Subset and Group Complete.

Analysis

In the post-IT Careers Day questionnaire, participants were also asked to complete a similar statement about the difficulty of programming with Alice - "Programming with Alice 3D worlds is .. " on a 9 point Likert scale (where 1 = 'really easy' and 9 = 'really difficult').

Note that both Group Subset and Group Complete were mixed in a common Alice programming session in the afternoon, after completing the Mindstorms workshops (see Figure 9-18).

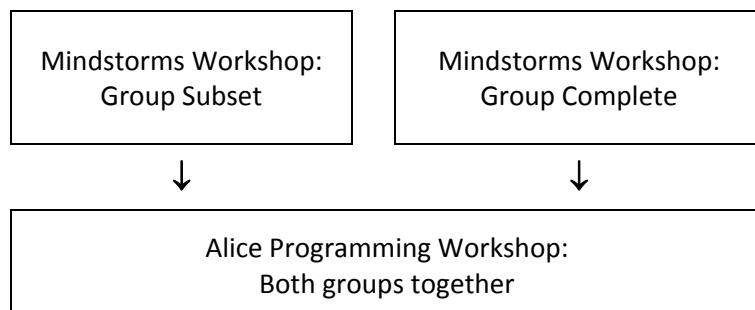


Figure 9-18 Mindstorms and Alice workshop relationship

Both groups completed the same Alice workshop activities, with the same Alice interface, and were asked afterwards about the difficulty of programming in Alice. As all participants were in the same Alice programming workshop, the only variable was the Mindstorms group in which each participant had participated. It was hypothesised that the difference in difficulty experienced in the Mindstorms

workshops as a result of having the complete interface compared to the subset interface would have a transfer effect to the perceived difficulty of Alice programming, due to the differences in base knowledge (schemas) acquired about programming concepts whilst participating in the Mindstorms activities.

The responses of the 15 participants in Group Subset and 14 participants in Group Complex who completed the Mindstorms workshops and completed the post-questionnaire after the Alice workshop were analysed for differences using a Mann-Whitney U Test. The novice programming participants in Group Complete found programming with *Alice* significantly more difficult than novice programming participants in Group Subset [$U_A = 158$, $z = 2.29$, $p = 0.011$].

From these results, the null hypothesis H_010 can be rejected in favour of the alternative hypothesis, that participants from Group Subset did have a lower perceived difficulty of Alice programming than participants from Group Complete.

This result shows a *transfer* effect from the Mindstorms workshop to the Alice workshop moderated by whether participants were presented the subset or complete interface for Mindstorms. This is discussed more in Section 9.7.

9.6.13 Hypothesis 11 (Cognitive Load)

H_A11 : Participants from Group Subset will report a different pattern of mental effort scores to participants in Group Complete.

H_011 : (Null Hypothesis) Participants from Group Subset will report a similar pattern of mental effort scores to participants in Group Complete.

Analysis

At the beginning of the Mindstorms test, participants in both groups were asked to indicate the mental effort which they had expended on each of three aspects of completing the Mindstorms Workshop activities. These were:

- understanding what they had to do in each exercise;
- navigating and using the Mindstorms NXT software;
- learning and understanding concepts;

on a 9-point Likert scale, where 1 = no effort and 9 = extreme effort.

It was anticipated that there would be difference on these reported measures between Group Subset and Group Complete, particularly around the mental effort expended in “navigating and using the Mindstorms NXT software”. However, there was no significant difference found between the groups for each of these three measures of mental effort (see Table 9-8), using Mann-Whitney U Tests:

Measure	U_A	z	p
Understanding the problem	127.5	0.02	0.492
Navigating and using the interface	142.5	-0.55	0.2912
Learning and understanding concepts	147	-0.72	0.2358

Table 9-8 Comparison between Groups for mental effort measures

Graphs of the mental effort measure distributions may show trends. For the measure of mental effort involved in “understanding the problem”, most students gave an ‘average’ score in both groups, with few students reporting high levels of mental effort (see Figure 9-19). No students in either group reported “extreme” mental effort was required to understand the problem, and only one student (from Group Complete) reported a level of 8 on the 9 point Likert scale.

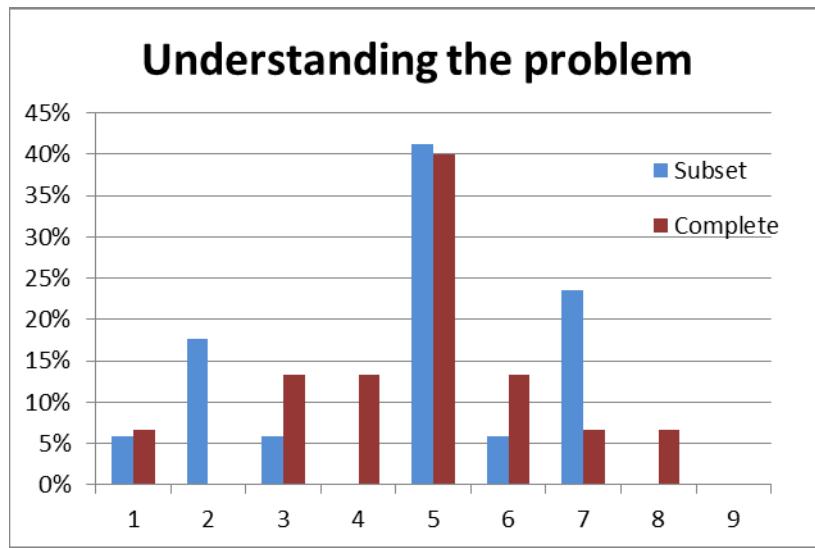


Figure 9-19 Mental Effort Measure distribution - Understanding the problem

In contrast, the mental effort scores given by the participants to “navigate and use the Mindstorms NXT programming software” have a wider spread (see Figure 9-20 below). Group Subset scores range from 1 (no mental effort) to 7 and Group Complete scores include two students who indicated that they needed very high to extreme levels (8 and 9) of mental effort to navigate and use the environment.

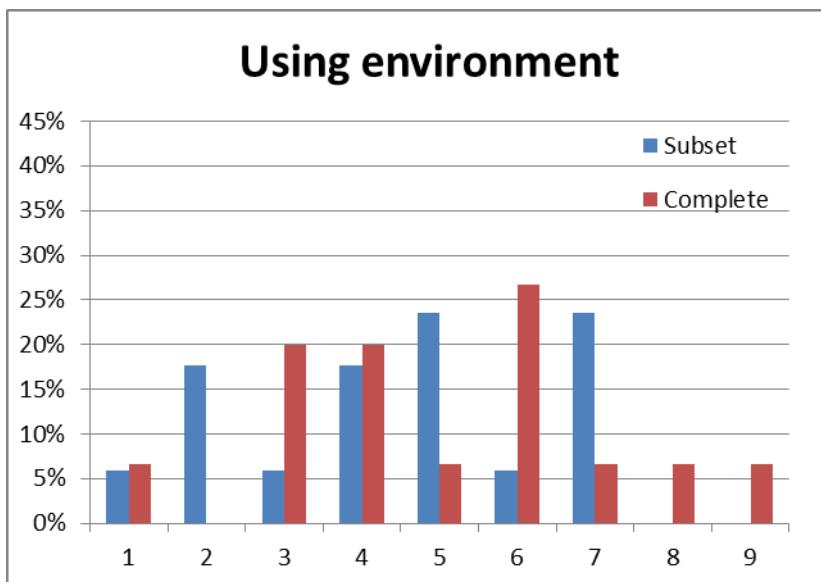


Figure 9-20 Mental Effort Measure distribution - Using the environment

Recall that the only difference between Group Subset and Group Complete is the number of programming blocks available in the interface. The extra programming

blocks in the interface used by Group Complete were not used in the exercises, which were common to both groups.

The distribution chart for the third measure “learning and understanding concepts” (Figure 9-21 below) shows that some students in Group Complete identified as expending ‘extreme mental effort’ on learning from the task. However these students were the same students who identified themselves as expending very high levels of mental effort on understanding the problem, and navigating and using the environment, making it unlikely that they had the short-term memory resources remaining to learn from the task. This highlights a difficulty with these types of self-reported mental effort measures, particularly with school children, some of which may not have the intellectual maturity to perform this kind of meta-analysis on their thought processes.

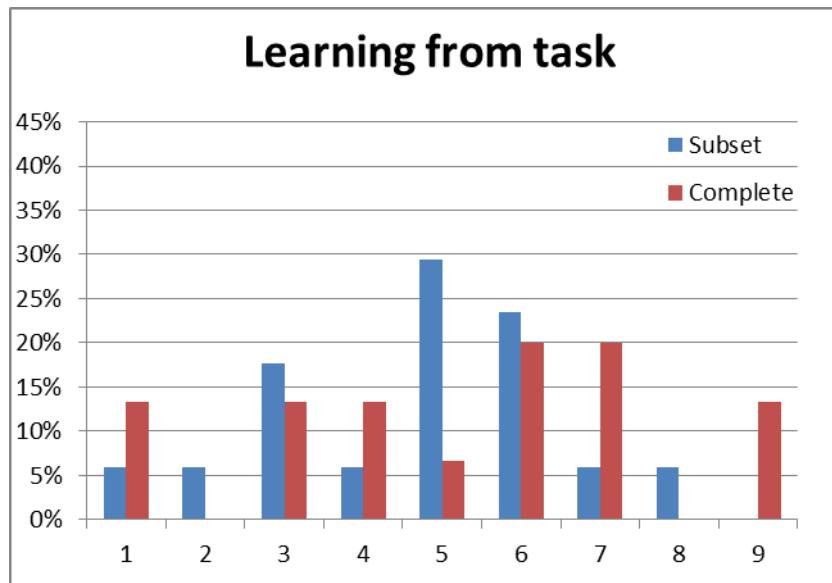


Figure 9-21 Mental Effort measure distribution - Learning from the task

Although the analysis of the raw scores for Group Subset and Group Complex show no significant differences between Group Subset and Group Complex on these three mental effort scores, it is informative to look at the extremes of the reported mental

effort scores. Some of the participants reported that they used ‘no’ (1 on the 9 point scale) or negligible (2 on the 9 point scale) mental effort on either understanding the problem or navigating the environment; or conversely, very high (8) or extreme (9 on the 9 point scale) mental effort on these two measures. The numbers of participants reporting *at least one* measure on these extremes is shown in Table 9-9 below.

At least one measure...	Subset	Complete
< 3 (no or negligible mental effort)	35%	7%
> 7 (very high to extreme mental effort)	0%	13%

Table 9-9 Very easy / Very hard mental effort proportions

Although Group Complete participants tended towards higher mental effort than Group Subset on two measures (navigating and using the interface, and learning and understanding concepts), there was not a statistically significant difference [Fisher Exact Probability Test: $p = 0.083$]. Although there were trends towards lower reported mental effort scores from participants in Group Subset than Group Complete, from these results, the null hypothesis H_011 cannot be rejected in favour of the alternative hypothesis.

9.6.14 Hypothesis 12 (Gender Difference - Performance)

H_A12 : There will be no difference between female and male participant performance in the Mindstorms test.

H_012 : (Null Hypothesis) There will be a significant difference between female and male participant performance in the Mindstorms test.

Analysis

The IT Careers Day was based on initiatives taken to encourage female students into technology. For this reason, female instructors were used, and participants had male-only participants and female-only participants in the Mindstorms workshops. Although all participants received instruction from female instructors, the workshop cohorts were all single gender. Recall that there was a significant difference between the test scores for Group Subset and Group Complete. The instructional design of the workshops was based on cognitive load principles, and so if the beneficial effects of the workshop (originally designed for females) are due primarily to the instructional design - rather than merely the use of female instructors - then both females and males should experience a performance difference in the Mindstorms test. Alternatively, if the beneficial effects are a result of the female-only instructor aspects of the workshops, there will be performance differences between males and females. In effect, there are 4 groups, arranged in a 2 x 2 structure (Figure 9-22 below).

	Group Subset	Group Complete
Male	Group M-S	Group M-C
Female	Group F-S	Group F-C

Figure 9-22 Gender versus Group

The test results from these groups were analysed using 2 x 2 ANOVA, and there was found to be no significant difference between male and female test scores ($F = 0.1$, $df = 1$, $p = 0.7542$), and no significant interaction ($F = 2.47$, $df=1$, $p = 0.1273$).

From these results, the null hypothesis H_012 can be rejected in favour of the alternative hypothesis, that there was no difference between female and male participant performance in the Mindstorms test.

9.6.15 Hypothesis 13 (Gender Difference – Attitude)

H_A12 : There will be no difference between girls and boys in the change in career aspirations and IT knowledge from before the workshop to after the workshop.

H_012 : (Null Hypothesis) There will be a difference between girls and boys in the change in career aspirations and IT knowledge from before the workshop to after the workshop.

Analysis

As these workshops were originally designed for female students, the use of female-only instructors and female-only workshops may have had a greater effect on the female students' attitudes towards their self-efficacy in computer programming, and their perceived knowledge of IT over the *change* in male students' attitudes.

Again, a 2×2 ANOVA (Group x Gender) was used to analyse the differences in self-efficacy and IT knowledge pre- and post-workshop between genders, groups and the interaction between these factors.

There was found to be no significant difference between male and female self-efficacy changes ($F = 0.55$, $df = 1$, $p = 0.4669$), and no significant interaction group x gender for self-efficacy ($F = 0.04$, $df=1$, $p = 0.8435$). There was also found to be no significant difference in IT knowledge changes between males and females ($F = 0.74$, $df = 1$, $p = 0.3999$), and no significant interaction group x gender for perceived IT knowledge ($F = 0.01$, $df = 1$, $p = 0.9213$).

From these results, the null hypothesis H_012 can be rejected in favour of the alternative hypothesis, that there was no difference between female and male

participants in the change in career aspirations and IT knowledge from before the workshop to after the workshop.

9.7 General Discussion

The results can be summarised as follows in Table 9-10:

Hypothesis	Accepted	Notes
General Benefits		
H1: Career Aspirations	Yes	Both groups had higher aspirations for a career in IT after workshops.
H2: IT Knowledge	Yes	Both groups had higher knowledge of IT after workshops
H3: Programming Difficulty	Yes	Perception of programming as easier after the workshops. Group Subset significant change, Group Complete no significant change.
H4: Self-efficacy	Yes	Significantly higher after workshops. Significant difference in Group Subset, no significant difference in Group Complete.
Performance Measures		
H5: Test Completion Time (Subset < Complete)	Yes	Group Subset took significantly less time to complete the test than Group Complete.
H6: Test Score (Subset > Complete)	Yes	Group Subset had significantly higher score than Group Complete.
H7: Schema Acquisition (Subset > Complete)	Yes	Group Subset scored significantly higher on rebuilding the interface than Group Complete.
H8: Knowledge Acquisition (Subset > Complete)	Yes	Group Subset scored higher than Group Complete.
Programming with Environments (Difficulty)		
H9: Mindstorms Difficulty (Subset < Complete)	Yes	Group Subset found Mindstorms programming significantly easier than Group Complete.
H10: Alice Difficulty (Subset < Complete)	Yes	Group Subset found Alice programming significantly easier than Group Complete.
Cognitive Load		
H11: Cognitive Load differences	No	Trends were present in the direction of lower cognitive load in Group Subset, but no significant differences.

(table continues over page)

Gender Differences		
H12: Gender Performance Difference (Male = Female)	Yes	No significant difference in performance between males and females, and no interaction with treatment group effect.
H13: Gender Attitude Difference (Male = Female)	Yes	No significant difference in self-efficacy or IT knowledge between males and females, and no interaction with treatment group.

Table 9-10 Summary of results

Positive benefits

The benefits of this type of intervention for both girls and boys who were novices to programming were clear, with positive shifts in IT career aspirations, IT knowledge, and self-efficacy in programming, and lowered perception of the difficulty of programming, from before the workshops to after the workshops. This is in agreement with the results from the Lightning Ridge intervention with girls only (Chapter 6), and also supports the positive results of the local IT Girls Days reported in Chapter 5.

Participants from Group Subset tended to have greater shifts in measures of self-efficacy and programming difficulty than Group Complete, however the differences between groups were not statistically significant.

Performance Measures

The effect of the treatments became evident on analysing the performance difference in the Mindstorms test. On each of the four chosen measures (Test Completion Time, Test Score, Schema Acquisition Score, and Knowledge Acquisition), Group Subset outperformed Group Complete. There was a demonstrated advantage for the students who were given the “simpler” subset interface which

presented fewer options, even though the extra options for Group Complete were not used.

Programming with Environments (Difficulty)

It was expected that if the students in Group Complete were experiencing higher cognitive load on working memory during the Mindstorms workshop than Group Subset as a result of the extra options on screen, then they would identify the task of programming in the Mindstorms environment as more difficult, compared to Group Subset. Group Complete did find programming in the Mindstorms environment significantly more difficult than Group Subset, which supports this hypothesis.

Both treatment groups then participated in the same Alice workshop, at the same time, with the same materials and instruction. There was a significant transfer effect, with those participants in Mindstorms Group Subset finding Alice programming significantly easier than participants who had been in Group Complete.

Cognitive Load

It was expected that participants in Group Subset would report lower mental effort (cognitive load) measures in “navigating and using the Mindstorms NXT software” and their overall patterns of mental effort would differ from participants in Group Complete. The performance measures indicate that learning was more effective for Group Subset, and so by Cognitive Load Theory, mental effort should be lower for Group Subset. Although there were trends present in the direction of lower cognitive load in Group Subset, there were no significant differences between groups. This result is discussed more fully in the discussion at the end of this chapter.

Gender Differences

Previous interventions were targeted at girls only, and part of the design of the program involved placing students in single-gender (female) workshops, and having female instructors. However, it was hypothesised that the instructional material used would be effective for both boys and girls, and that performance and attitude differences would be seen in both genders.

The results indicate that the performance and attitude differences between group Subset and group Complete were seen in both genders, and that (at least in the case of single-gender workshop design) this approach was equally valid for both genders.

9.8 Conclusion

Only one student, over the whole program, asked about any other programming blocks in the interface. The attention of participants was on completing activities and navigating those parts of the interface that they needed to use for the activities, rather than exploring other options. Despite this apparent disregard for the other options available in the interface, the tacit distraction provided from merely having those options available on screen, has impeded learning, decreased self-efficacy in ability to program, and affected perception of the difficulty of programming, for these participants who were novices to programming. Using the more complete interface has also had a flow-on (transfer) effect to affect students' perception of the difficulty of another programming environment.

The results using participants who are programming novices have implications for the often-debated question about whether it is better to introduce introductory programming students to an IDE that is used in industry first, or to introduce students

to programming using a teaching environment first, and then move on to an industry standard IDE later. Instructors that are in favour of using an industry standard environment for introductory programming courses often point to the extra effort of teaching two environments (see Chapter 7). However, the result of this research indicates that for novices to programming, having extra options available in the environment - *even if they are not used or referenced* - hinders their learning (reduces performance) and causes the students to perceive programming in both that environment and subsequent environments as more difficult.

There were no significant differences observed between groups on reported cognitive load measures, although the trends were in accordance with Cognitive Load Theory. This may be due to lack of sensitivity in the Likert test questions used with this participant pool, or may represent another dynamic associated with tacit distraction. This issue warrants further exploration to describe the distinction between these measures.

Chapter Ten: General Discussion

10.1 The Difficulty of Programming

Programming is widely accepted as being difficult to learn (Buck and Stucki, 2000; du Boulay, 1986; Jenkins, 2002; Marton and SäLjö, 1976; Schulte and Bennedsen, 2006), resulting in high drop-out rates (Denning and McGetrick, 2005; Ford and Venema, 2010), lack of progression into further programming courses (Bloch, 2000) and poor learning and poor performance in students (Bennedsen and Caspersen, 2007; Lister et al., 2004; Ma et al., 2007; McCracken et al., 2001).

To make programming more achievable by students, there has been much discussion about what should be taught in programming courses and how it should be taught. Various teaching tools and methodologies have been developed, including learning languages and environments. These learning environments have often resulted in reduced student attrition, increased student motivation and increased performance. Of primary interest, however, is the specific aspects and dynamics of such environments which may be beneficial to novices learning introductory programming.

10.2 Cognitive Load Theory

A theory that may offer insight into aspects of these environments from the perspective of human information processing is Cognitive Load Theory (Sweller, 1999). Cognitive Load Theory is an instructional theory based on knowledge of human cognitive architecture, which focuses on its limitations and the need to manage load placed on working memory. Cognitive Load Theory has given rise to many

demonstrably useful instructional effects, such as the Worked Example Effect, the Redundancy Effect and the Modality Effect.

Cognitive Load Theory is most useful in the context of the teaching and learning of complex information, that is, content that is high in element interactivity, especially for novices. When to-be-learnt information is complex, it imposes a high load on limited working memory and this may result in insufficient resources being available to enable successful learning (development, elaboration and automation of schemas) from the instructional material.

Whether an area is ‘complex’ depends on the person using the materials. If the learner has well-developed schemas in the area then it may be ‘simple’, however a novice to the area will likely find the same material ‘complex’. Teachers (in general) are experts in their area, and their students are not. This is also true, perhaps particularly true, for instructors in programming. Such instructors often - perhaps invariably - may not be aware of the complexity, interrelatedness and size of their schemas regarding programming. It may, therefore, be quite difficult for instructors to relate their instructional materials to the level of novices, who have small disconnected schemas, if any.

10.3 Females and Programming

Low numbers of females studying IT, and in particular, programming, at a university level, was the impetus for developing outreach programming workshops for high school girls. It was recognised that such workshops needed to be presented before senior years of high school, because by then, students have already committed themselves to vocational subject areas.

Instructional materials for the workshops were designed, developed and delivered in accordance with instructional guidelines from Cognitive Load Theory, and learning environments were chosen that adhered to these guidelines. The purpose of the workshops was to build participants' self-efficacy in introductory programming, make programming achievable, and to make the pathway to a career in Information Technology attractive and 'easy', rather than to place "roadblocks" or "hurdles" in the way, that seem too often to be present due to the relative complexity of materials and activities.

The workshops were also presented in such a way as to be mindful of gender effects, with all-female presenters and female-only workshops. The result of these workshops, reported in Chapter 5, was increased self-efficacy in programming and reduced perceptions of the difficulty of programming (to medium levels). While the workshops were built to be gender-specific, and on cognitive load theory design principles, the effect could have been logically attributable to participant motivation, as the girls had self-selected for these workshops.

The workshops were repeated for non-self-selecting girls (all of Year 7 to 12 at a remote school) and the same effects were obtained, as reported in Chapter 6. Self-selection and motivation cannot explain these positive results.

These high school girls were also asked about the mental effort that they expended while completing the activities. All groups reported moderate levels of mental effort on all measures, for intrinsic, extraneous and germane activities, which indicated that the instructional design had been effective, and was a likely cause for the successful outcomes.

10.4 The State of Play in Universities

There is a disconnect between the positive experience that these programming novices at high school reported and the difficult, even fear-filled, experiences of failure of university students as reported in literature. To determine the current activities, languages and environments that were being used in Australian Universities, as well as to explore the perceived mental effort experienced by instructors and university students while solving novice programming problems, a survey of introductory programming instructors was conducted as reported in Chapter 7.

The results of the mental effort questions were entirely consistent with Cognitive Load Theory. Instructors reported low levels of mental effort for themselves, moderate to high levels for the ‘average’ novice student and extreme to ‘off the scale’ levels for the less-able students, across all measures. Comments offered by the instructors indicated that they were sympathetic to students who were putting in their utmost effort to solve programming problems and to learn from them. The limited capacity of working memory, however, means that these struggling students could not possibly be using the maximum effort on everything at once. For these less-able, novice programming students, asking them to learn programming within some current courses may be akin to asking children to learn to swim by throwing them into the deep end of the pool and yelling instructions. As with swimming, the less able drown, and they do so conscious of their fate and full of fear.

Students in a typical introductory university programming course were then surveyed at the beginning and end of a semester, to determine demographic factors that may have an impact on mental effort expended when learning introductory programming, as reported in Chapter 8. The mental effort expended was also mapped

against the final grade achieved by participants in the survey. The participants in this study were broadly in agreement with the instructors of first programming courses. The students also expressed their experience of trying their utmost on all aspects – although this is impossible given the limitations of working memory. They were simply not recognising that it was impossible to learn (build and elaborate schemas) if their intrinsic and extraneous load is too high.

Recall that, on the whole, participants in the school workshops were not reporting extreme mental effort on any measure. Given the differences between the experience of the school students and the university students, it begs the question; what was it about the school workshops that made them successful?

Many aspects were factored into the design of the school workshops. They were female-friendly for a female only audience, they utilised worked examples, provided instant feedback in the form of visual responses, utilised the modality effect, and avoided redundancy. In addition, they used learning environments that were designed to reduce complexity. The positive effects experienced by participants could be attributed to any of these factors, or a synergy and interaction between some, or all, of the factors.

The final study, reported in Chapter 9, involved both male and female participants, with the intent of testing whether factors based on Cognitive Load Theory were likely responsible for the success of the workshops, rather than gender factors. A controlled experiment was performed that focused on one factor – the simplicity of the programming interface – to determine if this was one of the factors likely to have contributed to the success of the workshops.

10.5 Environment Simplicity

The results of the experiment reported in Chapter 9 indicated positive shifts in IT career aspirations, IT knowledge, self-efficacy in programming and lowered the perception of the difficulty of programming, for *both female and male* participants.

The experiment returned significant differences between participant groups given a complete programming learning environment interface, and a subset version of the interface. The Subset Group outperformed the Complete Group on performance score and performance time, and perceived programming in the environment to be less difficult than did the Complete Group. There was also a significant transfer effect, with participants in the Subset Group finding programming within a second environment significantly easier than those in the Complete Group.

The effect of having extra icons or options on screen, even when they are not referenced in any way, is related to Cognitive Load Theory's Split Attention and Redundancy Effects, but is not purely explained by these. Both the Split Attention Effect and Redundancy Effect refer to distractions that are caused when using relevant information that is explicitly part of the to-be-learnt content. The Split Attention Effect occurs when two mutually referring information sources are physically separated and must be integrated to be understood. In the interface experiment of Chapter 9, however,, the extra icons presented to the Complete Group are physically separated but are not used to perform the task. The Redundancy Effect occurs when two sources of information overlap and the effort required to integrate these sources adds to the total mental effort required for the problem. Again, the two sources of information are both used to perform the task. In the interface experiment of Chapter 9, however, extra information (in the form of extra programming blocks) is available on screen, but is not related to the task to be performed. Indeed, the extra blocks are

never referenced. The extra icons form a **tacit distraction** by their mere physical presence on screen and have had a negative effect on performance and self-efficacy.

Tacit distraction may represent a new form of cognitive load effect. The mere presence of irrelevant information, in this case, of irrelevant icon blocks, has been demonstrated to be sufficient to impede learning.

10.6 Limitations and Further Work

There are some limitations of the studies in this volume, which may give rise to further work.

- The positive changes in self-efficacy, decreased perceived difficulty of programming and increased IT career aspirations were reported directly after the intervention workshops described in Chapter 5, 6 and 9. Longitudinal studies are needed to follow-up the changes in perception for these students and to determine if this resolves into actual impacts upon career paths.
- The workshops in these studies were completed using high school students, not university students. They were also one-day workshops, rather than 13-week introductory university courses. The workshops may, however, have relevance to the processes of learning introductory programming for novices in university courses. Specifically, more research is needed into the aspects of languages, environments and activities that will help to scaffold and support novice students into the foundational concepts of programming, before they are exposed to professional environments.
- In the study of introductory university programming courses, instructors reported on the reasons for the choice of particular programming languages and environments. Although at times more than one reason was offered, these reasons were not weighted as to importance. A further study could be

developed that would drill further into the issues of industry-relevance reasons versus pedagogically-related reasons for choice.

- The study assumes that university students can distinguish between the mental effort being expended on the understanding of the problem statement, expended on using the environment, and expended on learning from the problem. Future work could include a study that explores whether students can accurately self-report in this way.
- In the survey of university students undertaking an introductory programming course there was a low participation rate and a strategic decision was made to give note to effects that were significant at the 0.1 level. A more comprehensive study of this nature using both a larger and more representative sample would enable greater confidence in the validity of results.
- In the controlled experiment in Chapter 9, tests were performed as paper-based tests, and the time taken for each participant was manually recorded for the whole-test. Computer-based presentation and management of the tests would allow tracking of the performance of individuals on each question.
- Although the Complete Group participants reported that they found the Alice workshops more difficult than did the Subset Group, performance of the two groups with Alice was not tested. Further study is needed to determine if using a less complex (subset) environment has performance transfer effects when using another environment.

Another area for further work involves using Cognitive Load Theory instructional design guidelines to identify other aspects of learning environments which may reduce cognitive load. For example, the number of windows used in an environment may have a bearing on cognitive load experienced by the learner, by the Split Attention Effect.

Also using Cognitive Load Theory, test problems used by novice programming courses could be examined for element interactivity as a means to determine their relative difficulty for novice programmers.. This may aid instructors to better tailor their assessments to ensure not disadvantaging struggling students, or even as a way to diagnose those students who have not yet acquired fully coherent and/or automated schemas.

10.7 A Final Word

The purpose of courses at school and university that teach programming is to assist in producing graduates who are ready for the professional world. The results of the studies reported in this thesis suggest that the best way to produce these graduates may be to nurture them through their first steps as a novice, using appropriate simplified subset environments, helping them to build schemas of fundamental programming concepts, before moving to a more professional setting.

More broadly, most people in our highly technology-focussed world use computers – not for programming, but for word-processing, using the internet, reading emails, and for training in non-computer-related fields. Tacit distraction effects may well apply to any computing context, and this has the implication that an over-supply of icons and other options and functionality in *any computer environment* are likely to impede the performance and learning by novices.

References

- Ahmadvad, M., Elliman, D., Higgins, C., 2005. An analysis of patterns of debugging among novice computer science students. ACM SIGCSE Bulletin 37, 84-88.
- Aimonetti, M., 2009. CouchDb Perform like a pr0n star. Golden Gate Ruby Conference. <http://www.slideshare.net/mattetti/couchdb-perform-like-a-pr0n-star?type=presentation> [Accessed 09-12-2009]
- Ambrose, S.A., 2010. How Learning Works: Seven Research-Based Principles for Smart Teaching. Jossey-Bass.
- Anastasiade, J.V., 2009. Instructional strategies for developing problem-solving skills: the worked-example effect using ill-structured visual pattern recognition problems. Doctoral Dissertation, Capella University <http://dl.acm.org/citation.cfm?id=1835131&coll=DL&dl=GUIDE&CFID=133949218&CFTOKEN=17527474> [Accessed 7-10-2012]
- Anderson, A.A., 1996. Predictors of computer anxiety and performance in information systems. Computers in Human Behavior 12, 61–77.
- Anderson, N., 2009. Equity and Information Communication Technology (ICT) in Education. Peter Lang, New York.
- Association for Computing Machinery, 2008. Computing Curricula - Information Technology 2008. Curriculum Guidelines for Undergraduate Degree Programs in Information Technology. URL: http://www.acm.org/education/curricula/IT2008_Curriculum.pdf [Accessed 29-09-2011].
- Atkinson, R.K., Derry, S.J., Renkl, A., Wortham, D., 2000. Learning from examples: instructional principles from the worked examples research. Review of Educational Research 70, 181–214.
- Atwood, J., 2007. Why Can't Programmers ... Program? [WWW Document]. URL <http://www.codinghorror.com/blog/archives/000781.html> [Accessed 9-3-2011]
- Australian Bureau of Statistics, 2006. Census Results by Location [WWW Document]. URL <http://www.censusdata.abs.gov.au> [accessed 12-15-2010].
- Australian Bureau of Statistics, 2010. Remoteness Structure [WWW Document]. URL <http://www.abs.gov.au/websitedbs/D3310114.nsf/home/remoteness+structure> [Accessed 24-01-2011].
- Australian Computer Society, 1999. ACS Submission to Review of Equal Opportunity for Women in the Workplace Act 1999 [WWW Document]. URL [http://www.acs.org.au/index.cfm?action=show&con ID=200911051329058011](http://www.acs.org.au/index.cfm?action=show&conID=200911051329058011) [Accessed 7-12-2009].
- Australian Computer Society, 2010. Australian ICT Statistical Compendium 2010 [WWW Document]. URL <http://www.acs.org.au/2010compendium> [Accessed 18-3-2011].
- Australian Computer Society, 2011. Australian ICT Statistical Compendium 2011 [WWW Document]. URL <http://www.acs.org.au/2011compendium/> [Accessed 16-02-2012].
- Australian Curriculum Assessment and Reporting Authority, 2010. My School: Lightning Ridge Central School, Lightning Ridge NSW [WWW Document].

- http://www.myschool.edu.au/Main.aspx?PageId=0&SDRSchoolId=NSWG00539_2395&DEEWRIId=16309&CalendarYear=2009 [Accessed 01-12-2010].
- Australian Curriculum Assessment and Reporting Authority, 2011. MySchool: Victory College, Gympie, QLD. [WWW Document]. MySchool. URL <http://myschool.edu.au/MainPages/SchoolProfileRep.aspx?SDRSchoolId=33000005869&DEEWRIId=2545&CalendarYear=2010&RefId=G9nrHja4Fb1BX2uVUjC5kJFmgC5cTR29> [Accessed 21-10-2011].
- Ausubel, D.P., 1968. Educational Psychology: A Cognitive View. Holt, Rinehart & Winston, New York, NY.
- Avancena, A.T., Nishihara, A., 2010. Automated tests for measuring cognition and aptitude in introductory programming, in: Proceedings of Global Learn Asia Pacific 2010. Presented at the Global Learn Asia Pacific (Global Learn) 2010, AACE, Penang, Malaysia, pp. 3371-3379.
- Baddeley, A., 1992. Working memory. *Science* 255, 556–559.
- Balci, O., Gilley, W.S., Adams, R.J., Tunar, E., Barnette, N.D., 2001. Animations to assist learning some key computer science topics. *Journal on Educational Resources in Computing* Vol 1, 2 Article 5
- Bandura, A., 1977. Self-efficacy: toward a unifying theory of behavioral change. *Psychological Review* 84, 191–215.
- Bandura, A., 1982. Self-efficacy mechanism in human agency. *American Psychologist* 37, 122–147.
- Barker, L., Aspray, W., 2000. The state of research on girls and IT, in: Cohoon, J.M., Aspray, W. (Eds.), *Women and Information Technology: Research on the Reasons for Under-Representation*. MIT Press, Cambridge, MA, 3–54.
- Barker, L., Snow, E., Weston, T., Garvin-Doxas, K., 2006. Recruiting middle school girls into IT: data on girls' perceptions and experiences from a mixed demographic group, in: Aspray, B., Cohoon, J. (Eds.), *Women and Information Technology: Research on Underrepresentation*. MIT Press, Cambridge, MA, 115–136.
- Barnes, D.J., 2002. Teaching introductory Java through LEGO MINDSTORMS models. *ACM SIGCSE Bulletin* 34, 147-151.
- Bateman, C.R., 1973. Predicting performance in a basic computer course, in: *Proceedings of the Fifth Annual Meeting of the American Institute for Decision Sciences*. Boston, Mass. 130-133.
- Beck, K., Cunningham, W., 1987. Using pattern languages for object-oriented programs. Presented at the OOPSLA-87 Workshop on the Specification and Design for Object-Oriented Programming. Available at <http://c2.com/doc/oopsla87.html> [Accessed 20-07-2011]
- Beise, C., VanBrackle, L., Myers, M., Chevli-Saroq, N., 2003. An examination of age, race and sex as predictors of success in the first programming course. *Journal of Informatics Education and Research* 5, 51-64.
- Bennedsen, J., Caspersen, M.E., 2006. Abstraction ability as an indicator of success for learning object-oriented programming? *ACM SIGCSE Bulletin* 38, 39–43.
- Bennedsen, J., Caspersen, M.E., 2007. Failure rates in introductory programming. *SIGCSE Bulletin* 39, 32–36.
- Bergin, S., Reilly, R., 2005. Programming: factors that influence success, in: *SIGCSE '05 Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*. ACM Press, 411–415.

- Beyer, S., DeKeuster, M., 2006. Women in Computer Science or Management Information Systems courses: a comparative analysis, in: Cohoon, J.M., Aspray, W. (Eds.), *Women and Information Technology: Research on the Reasons for Under-Representation*. MIT Press, Cambridge, MA, 3–54.
- Bishop-Clark, C., Courte, J., Evans, D., Howard, E., 2007. A quantitative and qualitative investigation of using Alice programming to improve confidence, enjoyment and achievement among non-majors. *Journal of Educational Computing Research* 37, 193–207.
- Blank, G.D., Pottenger, W.M., Sahasrabudhe, S., Li, S., Wei, F., Odi, H., 2003. Multimedia for Computer Science: from CS0 to grades 7-12, in: Proceedings of the ED-Media 03, World Conference on Educational Multimedia, Hypermedia & Telecommunications. EdMedia, Honolulu, Hawaii, USA. Available at <http://www.cse.lehigh.edu/~cimel/papers/EdMedia03.pdf> [Accessed: 9-03-2011]
- Bloch, S.A., 2000. Scheme and Java in the first year. *Journal of Computing Sciences in Colleges* 15, 157–165.
- Bloom, B.S., Englehart, M.D., Furst, E.J., Hill, W.H., Krathwohl, D.R., 1969. *Taxonomy of educational objectives : the classification of educational goals: handbook II: affective domain*. Longman Group, United Kingdom.
- Bogalin, V., 2011. forum post. LinkedIn Higher Education Teaching and Learning group. Available at http://www.linkedin.com/groupItem?view=&gid=2774663&type=member&item=69930810&qid=b1b4fa03-ae03-4495-8455-7b37bae93cc7&trk=group_items_see_more-0-b-ttl [Accessed 1-10-2011].
- Boisvert, C., 1995. A learning environment for Natural Language Processing, in: *Proceedings of the CS-NLP'95 Conference on the Cognitive Science of Natural Language Processing*. Dublin.
- Boisvert, C., 2006. Web animation to communicate iterative development. *ACM SIGCSE Bulletin* 38, 173–177.
- Borge, R., Fjuk, A., Groven, A.K., 2004. Using Karel J collaboratively to facilitate object-oriented learning, in: *IEEE International Conference on Advanced Learning Technologies (ICALT'04)*. IEEE Computer Society, 580–584.
- Bornat, R., Dehnadi, S., Simon, 2008. Mental models, consistency and programming aptitude, in: *ACE '08 Proceedings of the Tenth Conference on Australasian Computing Education*. Australian Computer Society, Inc., 53–61.
- Bradley, J., Millspaugh, A., 2011. *Programming in Visual Basic 2010*. McGraw-Hill, New York NY.
- Brainerd, C., Howe, M., 1978. The origins of all-or-none learning. *Child Development* 49, 1028–1034.
- Braught, G., 2005. Teaching empirical skills and concepts in computer science using random walks [WWW Document]. Technical Symposium on Computer Science Education. URL <http://doi.acm.org/10.1145/1047344.1047373>
- Brown, P.H., 2008. Some field experience with Alice. *Journal of Computing Sciences in Colleges* 24, 213–219.
- Bruce, K.B., 2004. Controversy on how to teach CS 1: a discussion on the SIGCSE-members mailing list. *Working group reports from ITiCSE on Innovation and technology in computer science education* 29–34.
- Bruner, J.S., 1961. The act of discovery. *Harvard Educational Review* 31, 21–32.

- Brunstein, A., Betts, S., Anderson, J.R., 2009. Practice enables successful learning under minimal guidance. *Journal of Educational Psychology* 101, 790–802.
- Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A., Miller, P., 1997. Mini-languages: a way to learn programming principles. *Education and Information Technologies* 2, 65–83.
- Brusilovsky, P., Weber, G., 1996. Collaborative example selection in an intelligent example-based programming environment, in: *Proceedings of the 1996 International Conference on Learning Sciences*. International Society of the Learning Sciences, pp. 357–362.
- Buck, D., Stucki, D.J., 2000. Design early considered harmful: graduated exposure to complexity and structure based on levels of cognitive development [WWW Document]. *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*. URL <http://doi.acm.org/10.1145/330908.331817>
- Buck, D., Stucki, D.J., 2001. JKarelRobot: a case study in supporting levels of cognitive development in the computer science curriculum, in: *The 32nd SIGCSE Technical Symposium on Computer Science Education*. ACM Press, Charlotte, North Carolina, USA, 16–20.
- Burton, P.J., Bruhn, R.E., 2003. Teaching programming in the OOP era. *SIGCSE Bulletin* 35, 111–114.
- Byrne, M.D., Catrambone, R., Stasko, J.T., 1999. Evaluating animations as student aids in learning computer algorithms. *Computers & Education* 33, 253–278.
- Byrne, P., Lyons, G., 2001. The effect of student attributes on success in programming. *ACM SIGCSE Bulletin* 33, 49–52.
- Carbone, A., Hurst, J., Mitchell, I., Gunstone, D., 2009. An exploration of internal factors influencing student learning of programming, in: *ACE '09 Proceedings of the Eleventh Australasian Conference on Computing Education*. Australian Computer Society, Inc., Darlinghurst, Australia, 25–34.
- Carnegie Mellon University, 2006. What is Alice and what is it good for? [WWW Document]. URL http://www.alice.org/index.php?page=what_is_alice/what_is_alice [Accessed 25-11-2006).
- Caron, F., 2009. Microsoft trains next-gen coders with XNA's Kodu. *Ars Technica*. Available at <http://arstechnica.com/gaming/news/2009/01/microsoft-trains-next-gen-coders-with-xnas-kodu.ars> [Accessed 18-02-2009] .
- Carter, J., Bouvier, D., Cardell-Oliver, R., Hamilton, M., Kurkovsky, S., Markham, S., McClung, O.W., McDermott, R., Riedesel, C., Shi, J. & White, S. 2011. Motivating all our students? in *Proceedings of the 16th annual conference reports on Innovation and technology in computer science education - working group reports* ACM Press, pp. 1-18.
- Carter, J., White, S., Fraser, K., Kurkovsky, S., McCreesh, C., Wieck, M., 2010. ITiCSE 2010 working group report motivating our top students in *Proceedings of the 2010 ITiCSE working group reports* ACM Press, pp 29-47.
- Caspersen, M.E., Bennedsen, J., 2007. Instructional design of a programming course: a learning theoretic approach, in: *ICER '07 Proceedings of the Third International Workshop on Computing Education Research*. ACM Press, pp. 111–122.
- Caspersen, M.E., Larsen, K.D., Bennedsen, J., 2007. Mental models and programming aptitude. in *ITiCSE '07 Proceedings of the 12th annual SIGCSE*

- conference on Innovation and technology in computer science education. ACM Press, pp. 206-210.
- Catrambone, R., 1998. The subgoal learning model: creating better examples so that students can solve novel problems. *Journal of Experimental Psychology: General* 127, 355–376.
- Chalk, B., Fraser, K., 2006. A survey on the teaching of introductory programming in Higher Education, in: Proceedings of the 10th Java & the Internet in the Computing Curriculum Conference (JICC10). Thomson Publishers, London, United Kingdom, pp. 1–6.
- Chalmers, C., Price, P., 2000. Promoting gender equity in the information technology classroom. *Australian Educational Computing* 15, 13–16.
- Chandler, P., Sweller, J., 1991. Cognitive Load Theory and the format of instruction. *Cognition and Instruction* 8, 293–332.
- Charney, D.H., Reder, L.M., Kusbit, G.W., 1990. Goal setting and procedure selection in acquiring computer skills: a comparison of tutorials, problem-solving, and learner exploration. *Cognition and Instruction* 7, 323–342.
- Chase, W.G., Simon, H.A., 1973. Perception in chess. *Cognitive Psychology* 4, 55–81.
- Chi, M., Glaser, R., Rees, E., 1982. Expertise in problem solving, in: Advances in the Psychology of Human Intelligence. Erlbaum, Hillsdale, NJ, pp. 7–75.
- Chmura, G.A., 1998. What abilities are necessary for success in computer science. *ACM SIGCSE Bulletin* 30, 55–58.
- Commonwealth of Australia, 2010. Other Grants Guidelines (Education) [WWW Document]. URL
<http://www.deewr.gov.au/HigherEducation/Programs/Equity/Pages/HEPPProgram.aspx> [Accessed 27-01-2011].
- Cooper, G., 1998. Research into Cognitive Load Theory and instructional design at UNSW [WWW Document]. URL
<http://dwb4.unl.edu/Diss/Cooper/UNSW.htm> [Accessed 12-10-2000].
- Cooper, G., Sweller, J., 1987. Effects of schema acquisition and rule automation on mathematical problem-solving transfer. *Journal of Educational Psychology* 79, 347–362.
- Cooper, G., Tindall-Ford, S., Chandler, P., Sweller, J., 2001. Learning by imagining. *Journal of Experimental Psychology: Applied* 7, 68–82.
- Cooper, S., 2010. The design of Alice. *ACM Transactions on Computing Education* 10, pp. 1–16.
- Cooper, S., Dann, W., Pausch, R., 2000. Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges* 15, 107–116.
- Cooper, S., Dann, W., Pausch, R., 2003. Teaching objects-first in introductory computer science. 34th SIGCSE technical symposium on Computer science education SIGCSE '03 35, 191–195.
- Corney, M., Teague, D., Ahadi, A., Lister, R., 2012. Some Empirical Results for Neo-Piagetian Reasoning in Novice Programmers and the Relationship to Code Explanation Questions, in: Proc. Australasian Computing Education Conference (ACE2012). Australian Computer Society, Inc., Melbourne, Australia, pp. 77–86.

- Craig, M., Horton, D., 2009. Gr8 designs for gr8 girls: a middle-school program and its evaluation. in SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education. ACM Press, pp. 221–225.
- Crosby, M., Stelovsky, J., 1995. From multimedia instruction to multimedia evaluation. *Journal of Educational Multimedia and Hypermedia* 4, 147–162.
- Csikszentmihalyi, M., 1997. *Finding Flow: The Psychology of Engagement With Everyday Life*. Basic Books.
- D'Souza, D., Hamilton, M., Harland, J., Muir, P., Thevathayan, C., Walker, C., 2008. Transforming learning of programming: a mentoring project, in: Proceedings Tenth Australasian Computing Education Conference (ACE 2008). The Australian Computer Society, Australia. pp. 75–84.
- Dale, N., 2005. Content and emphasis in CS1. *ACM SIGCSE Bulletin* 37, 69–73.
- Daly, T., 2011. Minimizing to maximize: an initial attempt at teaching introductory programming using Alice. *Journal of Computing Sciences in Colleges* 26, 23–30.
- Dann, W., Cooper, S., Slater, D., 2006. An overview of Alice [WWW Document]. Alice Programming. URL <http://www.aliceprogramming.net/overview/overview.html> [Accessed 04-02-2012].
- Dann, W., Cosgrove, D., Slater, D., Culyba, D., Cooper, S., 2012. Mediated transfer: Alice 3 to Java, in: Proceedings of the 43rd ACM Technical Symposium on Computer Science Education. ACM Press, pp. 141–146.
- Darabi, A.A., Nelson, D.W., Palanki, S., 2007. Acquisition of troubleshooting skills in a computer simulation: worked example vs. conventional problem solving instructional strategies. *Computers in Human Behavior* 23, 1809–1819.
- Davy, J., Jenkins, T., 1999. Research-led innovation in teaching and learning programming. In: ITiCSE '99: Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education. ACM Press, pp. 5–8.
- De Groot, A., 1965. Thought and choice in chess. Mouton, The Hague, Netherlands.
- De Raadt, M., Watson, R., Toleman, M., 2002. Language trends in introductory programming courses [WWW Document]. Informing Science + IT Education Conference. URL http://proceedings.informingscience.org/IS2002Proceedings/papers/deRaa136_Langu.pdf [Accessed 2-04-2008]
- De Raadt, M., Watson, R., Toleman, M., 2003. Introductory programming languages at Australian universities at the beginning of the twenty-first century. *Journal of Research and Practice in Information Technology* 35, 163–167.
- De Raadt, M., Watson, R., Toleman, M., 2004. Introductory programming: what's happening today and will there be any students to teach tomorrow?, in: ACE'04 Proceedings of the Sixth Conference on Australasian Computing Education. Australian Computer Society, Inc., pp. 277–282.
- Deci, E.L., 1975. *Intrinsic Motivation*. Plenum, New York, USA.
- Decker, A., 2003. A tale of two paradigms. *Journal of Computing Sciences in Colleges* 19, 238–246.
- Decker, R., Hirshfield, S., 1994. The top 10 reasons why object-oriented programming can't be taught in CS 1. *ACM SIGCSE Bulletin* 26, 51–55.

- DEEWR, 2011. Australian Jobs 2011 [WWW Document]. URL
<http://www.deewr.gov.au/Employment/ResearchStatistics/Pages/AustralianJobs.aspx> [Accessed 22-08-2011].
- Dehnadi, S., 2007. Testing Programming Aptitude [WWW Document]. URL
http://www.cs.mdx.ac.uk/research/PhDArea/saeed/S_Dehnadi_ppij-2006_2.pdf [Accessed 9-03-2011]
- Dehnadi, S., Bornat, R., 2007. The camel has two humps (working title) [WWW Document]. URL <http://cs.mdx.ac.uk/research/PhDArea/saeed/paper1.pdf> [Accessed 01-07-2008]
- Denning, P., McGetrick, A., 2005. Recentering Computer Science. Communications of the ACM 48, 15–19.
- Denning, P.J., 2003. Great principles of computing. Communications of the ACM 46, 15-20.
- Dept of Business and Innovation, 2012. 2011 ICT Skills Snapshot - The state of ICT skills in Victoria. Dept of Business and Innovation, Victoria, Australia.
- Dept of Employment Education and Workplace Relations, 2011. Digital Education Revolution [WWW Document]. URL
<http://www.deewr.gov.au/Schooling/DigitalEducationRevolution/Pages/default.aspx> [Accessed 15-08-2011].
- Dept of Health and Aging, 2011. DoctorConnect Remoteness Map [WWW Document]. URL
<http://www.doctorconnect.gov.au/internet/otd/Publishing.nsf/Content/locator> [Accessed 24-01-2011].
- Dept of Innovation Industry and Regional Development, 2009. Attitudes to ICT careers and study among 14 to 19 year old Victorians. Dept of Innovation Industry and Regional Development, Melbourne, Australia.
- deWit, K., Heerwegh, D., Verhoeven, J.C., 2012. Do ICT competencies support educational attainment at University? Journal of Information Technology Education: Research 11. Available at
<http://www.jite.org/documents/Vol11/JITEv11p001-025DeWit1037.pdf> [Accessed 22-05-2012]
- Dijkstra, E.W., 1972. Notes on structured programming, in: Structured Programming. Academic Press, New York, NY, pp. 1–82.
- Dorn, B., 2008. Jeroo Teaching Notes [WWW Document]. Jeroo. URL
<http://home.cc.gatech.edu/dorn/49> [Accessed 12-08-2009].
- Dorn, B., Sanders, D., 2003. Using Jeroo to introduce object-oriented programming. In: Proceedings of 33rd ASEE/IEEE Frontiers in Education Conference pp 22-27.
- Double, W., 1998. Multimedia delivery of computer programming subjects: basing structure on instructional design. Proceedings of the 3rd Australiasian conference on Computer Science Education 85–93.
- Du Boulay, B., 1986. Some difficulties of learning to program. Journal of Educational Computing Research 2, 57–73.
- Eckerdal, A., McCartney, R., Mostrom, J.E., Ratcliffe, M., Zander, C., 2006. Can graduating students design software systems? In: Proceedings of the 37th SIGCSE technical symposium on Computer science education. pp. 403-407.
- Edwards, J., Kay, J., 2001. A sorry tale - a study of women's participation in IT Higher Education in Australia. Journal of Research and Practice in Information Technology 33, 329–335.

- Ehlert, A., Schulte, C., 2009. Empirical comparison of objects-first and objects-later. In: Proceedings of the fifth international workshop on Computing education research workshop. pp. 15–26.
- Ellis, J., 2010. Styles of logic and thinking - implications of Nisbett's Geography of Thought for Teaching and Assessment in the multi-cultural classroom. ACM Inroads 1, 34–42.
- Ensmenger, N., 2011. Programmer Aptitude? [WWW Document]. The Computer Boys Take Over - Computers, Programmers and the Politics of Technical Expertise. URL <http://thecomputerboys.com/?tag=aptitude> [A 28-05-2012].
- Entwistle, N., 1998. Motivation and approaches to learning: motivation and conceptions of teaching, in: Brown, S., Armstrong, S., Thompson, G. (Eds.), Motivating Students. Kogan Page, London, United Kingdom.
- Evans, G.E., Simkin, M.G., 1989. What best predicts computer proficiency? Communications of the ACM 32, 1322–1327.
- Farrell, M., 2007. What are Students' Attitudes toward Technology? Presentation available at <http://www.cips.ca/?q=webcasts> - <http://www.cips.ca/?q=systems/files/MicrosoftCIPSPresentation2007Live.ppt> [Accessed 25-01-2011].
- Fay, A.L., Mayer, R.E., 1994. Benefits of teaching design skills before teaching LOGO computer programming: evidence for syntax-independent learning. Journal of Educational Computing Research 11, 187–210.
- Feldman, D.H., 2004. Piaget's stages: the unfinished symphony of cognitive development. New Ideas in Psychology 175–231.
- Felleisen, M., Fisler, K., Krishnamurthi, S., 1999. The technology that Computer Science education overlooked, in: Proceedings of International Conference on Mathematics / Science Education and Technology 1999., pp. 9 – 13.
- Felleisen, M., Flatt, M., Findler, R.B., 2008. PTL Scheme [WWW Document]. URL <http://plt-scheme.org> [Accessed: 21-10-2009]
- Findler, R.B., Clements, J., Flanagan, C., Flatt, M., Krishnamurthi, S., Steckler, P., Felleisen, M., 2002. DrScheme: a programming environment for Scheme. Journal of Functional Programming 12.
- Fisher, A., Margolis, J., 2002. Unlocking the clubhouse: the Carnegie Mellon experience. SIGCSE Bulletin 34, 79–83.
- Fisher, J., Lang, C., Forgasz, H.J., Craig, A., 2009. Digital Divas: working to change students' perceptions about ICT courses and careers. Curriculum Leadership 7.
- Ford, M., Venema, S., 2010. Assessing the success of an introductory programming course. Journal of Information Technology Education 9.
- Foster, A.L., 2005. Student interest in Computer Science plummets [WWW Document]. The Chronicle of Higher Education. URL <http://chronicle.com/article/Student-Interest-in-Computer/10912> [Accessed 08-09-2011].
- Fowler, M., 2009. SmutOnRails. Martin Fowler's Blog. URL: <http://martinfowler.com/bliki/SmutOnRails.html> [Accessed 30-04-2009].
- Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design Patterns: elements of reusable object-oriented software. Addison-Wesley, Reading, Massachusetts.
- Garlick, R., Cankaya, E.C., 2010. Using Alice in CS1: a quantitative experiment, in: Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education. ACM Press, pp. 165–168.

- Garner, S., 2002. Reducing the cognitive load on novice programmers, in: Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2002. AACE, Chesapeake, VA, pp. 578–583.
- Gomes, A.J., Santos, A.N., Mendes, A.J., 2012. A study on students' behaviours and attitudes towards learning to program. In: ITiCSE '12 Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education. ACM Press, p. 132-137.
- Gomez-Albaran, M., 2005. The teaching and learning of programming: a survey of supporting software tools. *The Computer Journal* 48, 130–144.
- Graduate Careers Australia, 2012. Computer Science - Bachelor Graduates (All) [WWW Document]. Graduate Careers Online. URL <http://www.graduatecareers.com.au/Research/GradJobsDollars/BachelorAll/ComputerScience/index.htm> [Accessed 03-05-2012].
- Graham, S., Latulipe, C., 2003. CS girls rock. *ACM SIGCSE Bulletin* 35, 322.
- Grant, N., Hsiao, S., Turich, T., 2010. A research study on the usage of implementing Alice in an introductory Java computer programming course, in: Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2010. pp. 1958-1961.
- Gray, S., Clair, C.S., James, R., Mead, J., 2007. Suggestions for graduated exposure to programming concepts using fading worked examples In: Proceedings of the third international workshop on Computing education research. pp. 99-110.
- Gross, D., 2012. In tech, some bemoan the rise of “brogrammer” culture [WWW Document]. CNN. URL <http://edition.cnn.com/2012/05/07/tech/web/brogrammers/index.html> [Accessed 31-07-2012].
- Gruba, P., Moffat, A., Sondergaard, H., Zobel, J., 2007. What drives curriculum change?. In: Proceedings of the sixth conference on Australasian computing education (ACE'06). pp 109-117.
- Guibert, N., Girard, P., 2003. Programming by example and computer-aided teaching of algorithmics. ACM Press, pp. 248–251.
- Guibert, N., Girard, P., Guittet, L., 2004. Example-based programming: a pertinent visual approach for learning to program. In: AVI '04 Proceedings of the working conference on Advanced visual interfaces. ACM Press, p. 358-361.
- Guthrie, J.T., 1967. Expository instruction versus a discovery method. *Journal of Educational Psychology* 45–49.
- Guzdial, M., 2001. Using squeak for teaching user interface software, in: Proceedings of the Thirty-second Technical Symposium on Computer Science Education. ACM Press, Charlotte, North Carolina, USA, pp. 219–223.
- Guzdial, M., 2010. The Complicated Issues of Computing Education in Qatar. Communications of the ACM blog. URL: <http://cacm.acm.org/blogs/blog-cacm/91580-the-complicated-issues-of-computing-education-in-qatar/fulltext>. [Accessed 5-01-2011]
- Hardy, C., Heeler, P., Brooks, D., 2006. Are high school graduates technologically ready for post-secondary education? *Journal of Computing Sciences in Colleges* 21, 52–60.
- Harriger, A., 2009. Could Alice equalize student learning?, in: Proceedings of the 2009 Alice Symposium. ACM Press, Article No. 8.

- Hauser, R., Paul, R., Bradley, J., 2012. Computer self-efficacy, anxiety and learning in online versus face to face medium. *Journal of Information Technology Education: Research* 11.
- Hawkridge, D., 1983. *New Information Technologies in Education*. Croom Helm, London.
- Hayes, S., 2012. That's College, 1st ed. New College, UNSW, Sydney.
- Head, B., 2012. Univers-IT entry standards plummet. *IT Wire*. URL <http://www.itwire.com/it-people-news/training/54369-univers-it-challenge> [Accessed 30-04-2012].
- Helme, S., Clarke, D., 2001. Identifying cognitive engagement in the mathematics classroom. *Mathematics Education Research Journal* 13, 133–153.
- Henriksen, P., Kölbing, M., 2004. greenfoot: combining object visualisation with interaction. In: Proceedings of the 19th Annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (OOPSLA). URL <http://www.greenfoot.org/doc/papers.html>
- Hovis, R.A., 2005. Managing the complexity in first year programming. *ACM SIGCSE Bulletin* 37, 394.
- Howles, T., 2007. Preliminary results of a longitudinal study of computer science student trends, behaviors and preferences. *Journal of Computing Sciences in Colleges* 22, 18–27.
- Hu, H.H., 2008. A summer programming workshop for middle school girls. *Journal of Computing Sciences in Colleges* 23, 194–202.
- Hundhausen, C.D., Douglas, S., Stasko, J.T., 2002. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing* 13, 259–290.
- Hundhausen, C.D., Farley, S.F., Brown, J.L., 2009. Can direct manipulation lower the barriers to computer programming and promote transfer of training? *ACM Transactions on Computer-Human Interaction* 16, 1–40.
- Hunt, M., 1982. *The Universe Within*. The Harvester Press, Great Britain.
- Ingalls, D., Kaehler, T., Maloney, J., Wallace, S., Kay, A., 1997. Back to the future, in: OOPSLA '97 Proceedings of the 12th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. ACM Press, pp. 318–326.
- Ingalls, D., Kay, A., Kaehler, T., Wallace, S., 2007. About Squeak [WWW Document]. URL <http://www.squeak.org/About> [Accessed: 18-05-2007]
- Jarusek, P., Pelánek, R., 2012. A web-based problem solving tool for introductory computer science. In: ITiCSE '12 Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education . ACM Press, p. 371.
- Jenkins, T., 2002. On the difficulty of learning to program, in: Proceedings of the 3rd Annual Conference of the LTSN-ICS. Loughborough, Ireland, pp. 53–58.
- Kahle, J., Schmidt, G., 2004. Reasons women pursue a computer science career: perspectives of women from a mid-sized institution. *Journal of Computing in Small Colleges* 19, 78–89.
- Kalyuga, S., Ayres, P., Chandler, P., Sweller, J., 2003. Expertise reversal effect. *Educational Psychologist* 38, 23–33.
- Kalyuga, S., Chandler, P., Tuovinen, J., Sweller, J., 2001. When problem solving is superior to studying worked examples. *Journal of Educational Psychology* 93, 579–588.

- Kasmarik, K., Thurbon, J., 2003. Experimental evaluation of a program visualisation tool for use in computer science education, in: APVis '02 Proceedings of the Asia-Pacific Symposium on Information Visualisation. Australian Computer Society, Inc., pp. 111–116.
- Kay, A., 2008. Sketching and Scripting with Squeak [WWW Document]. Squeakland - Home of Squeak Etoys. URL
<http://www.squeakland.org/about/intro/article.jsp?id=2317> [Accessed 06-06-2012].
- Kegel, D., 2008. How to get hired -- what CS students need to know [WWW Document]. URL <http://www.kegel.com/academy/getting-hired.html> [Accessed 04-08-2010].
- Kelleher, C., Cosgrove, D., Culyba, D., Forlines, C., Pratt, J., Pausch, R., 2002. Alice2: programming without syntax errors. Presented at the 2002 Conference on User Interface Software and Technology, Carnegie Mellon University.
- 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., 2007. Using storytelling to motivate programming. Communications of the ACM 50, 58.
- Kelleher, C., Pausch, R., Kiesler, S., 2007. Storytelling Alice motivates middle school girls to learn computer programming. in: CHI '07 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM Press, p. 1455-1464.
- Kirschner, P.A., 2002. Cognitive load theory: implications of cognitive load theory on the design of learning. Learning and Instruction 12, 1–10.
- Klahr, D., Nigram, M., 2004. The equivalence of learning paths in early science instruction: effects of direct instruction and discovery learning. Psychological Science 15, 661–667.
- Klassner, F., 2002. A case study of LEGO Mindstorms' suitability for artificial intelligence and robotics courses at the college level. ACM SIGCSE Bulletin 34, 8–12.
- Klassner, F., Anderson, S.D., 2003. LEGO mindstorms: Not just for K-12 anymore. IEEE Robotics & Automation Magazine 10, 12–18.
- Kölling, M., 1999a. The problem of teaching object-oriented programming, part 2: Environments. Journal of Object-Oriented Programming 11, 6–12.
- Kölling, M., 1999b. The Design of an Object-Oriented Environment and Language for Teaching (PhD Thesis). University of Sydney, Sydney.
- Kölling, M., 2006a. I Object! [WWW Document]. URL
<http://www.bluej.org/mrt/docs/objection.pdf> [Accessed 07-12-2007]
- Kölling, M., 2006b. The problem of teaching object-oriented programming, part 1: Languages. Journal of Object-Oriented Programming 11, 8–15.
- Kölling, M., 2008. Greenfoot: a highly graphical ide for learning object-oriented programming. ACM SIGCSE Bulletin 40, 327.
- Kölling, M., 2010. The Greenfoot Programming Environment. ACM Transactions on Computing Education 10, 1–21.
- Kölling, M., 2012. BlueJ - Users List [WWW Document]. BlueJ - The interactive Java environment. URL <http://www.bluej.org/about/users.html> [Accessed 22-10-2012].

- Kölling, M., Quig, B., Patterson, A., Rosenberg, J., 2003. The BlueJ system and its pedagogy. *Journal of Computer Science Education, Special Issue on Learning and Teaching Object Technology* 13.
- Kölling, M., Utting, I., McCall, D., Brown, N., Stevens, P., Berry, M., 2012. Greenfoot [WWW Document]. Greenfoot. URL <http://www.greenfoot.org/home> [Accessed 12-02-2012].
- Kotovsky, K., Hayes, J.R., Simon, H.A., 1985. Why are some problems hard? Evidence from Tower of Hanoi. *Cognitive Psychology* 17, 248–294.
- Krishnamurthi, S., Felleisen, M., Duba, B.F., 2000. From Macros to Reusable Generative Programming, in: Czarnecki, K., Eisenecker, U.W. (Eds.), *Generative and Component-Based Software Engineering, Lecture Notes in Computer Science*. Springer, pp. 105–120.
- Kruck, S.E., Lending, D., 2003. Predicting academic performance in an introductory college-level IS course. *Information Technology, Learning and Performance Journal* 21, 9–15.
- Kuittinne, M., Sajaniemi, J., 2004. Teaching roles of variables in elementary programming courses. *ACM SIGCSE Bulletin* 36, 57–61.
- Leahy, W., Chandler, P., Sweller, J., 2003. When auditory presentations should and should not be a component of multimedia instruction. *Applied Cognitive Psychology* 17, 401–418.
- Learning Space, 2008. Go Go Gidgits [WWW Document]. URL <http://www.learningplace.com.au/defaulteqa2.asp?orgid=48&suborgid=535> [Accessed 08-08-2011].
- Lee, M., Thompson, A., 1997. Guided instruction in LOGO programming and the development of cognitive monitoring strategies among college students. *Journal of Educational Computing Research* 16, 125–144.
- LeMay, R., 2010. Microsoft apologizes for Meter Maid bikinis at Aussie Tech.Ed [WWW Document]. ZDNet. URL <http://www.zdnet.com/news/microsoft-apologizes-for-meter-maid-bikinis-at-aussie-tech-ed/460085> [Accessed 07-10-2010].
- Lepper, M.R., 1988. Motivational considerations in the study of instruction. *Cognition and Instruction* 5, 289–309.
- Lewis, M.W., Anderson, J.R., 1985. Discrimination of operator schemata in problem-solving: procedural learning from examples. *Cognitive Psychology* 17, 26–65.
- Liao, Y., Bright, G., 1991. Effects of computer programming on cognitive outcomes: A meta-analysis. *Journal of Educational Computing Research* 7, 251–268.
- Lifelong Kindergarten Group, 2007a. Creating with Scratch [WWW Document]. URL <http://llk.media.mit.edu/projects/scratch/papers/Creating-with-Scratch1.pdf> [Accessed 9-03-2011].
- Lifelong Kindergarten Group, 2007b. Programming with Scratch [WWW Document]. URL <http://llk.media.mit.edu/projects/scratch/papers/Programming-with-Scratch.pdf> [Accessed 9-03-2011].
- Lister, R., 2000. On blooming first year programming, and its blooming assessment, in: ACSE '00 Proceedings of the Australasian Conference on Computing Education. ACM Press, pp. 158–162.
- Lister, R., 2008. After the gold rush: toward sustainable scholarship in computing, in: ACE '08 Proceedings of the Tenth Conference on Australasian Computing Education. Australian Computer Society, Inc., pp. 3–17.

- Lister, R., 2010. Computing Education Research: Geek genes and bimodal grades. *ACM Inroads* 1, 16–17.
- Lister, R., Adams, E., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Mostrom, J.E., Sanders, K., Seppala, O., Simon, B., Thomas, L., 2004. A Multi-National Study of Reading and Tracing Skills in Novice Programmers. *ACM SIGCSE Bulletin* 36, 119–150.
- Lister, R., Berglund, A., Clear, T., Bergin, J., Garvin-Doxas, K., Hanks, B., Hitchner, L., Luxton-Reilly, A., Sanders, K., Schulte, C., Whalley, J.L., 2006. Research perspectives on the objects-early debate. *ACM SIGCSE Bulletin* 38, 146–165.
- Lister, R., Leaney, J., 2003. Introductory programming, criterion-referencing, and bloom. ACM Press, p. 143.
- Lomako, G., 2007. Learning computer programming and role of errors in design. *ACM SIGCSE Bulletin* 39, 142.
- Lung, J., Aranda, J., Easterbrook, S.M., Wilson, G.V., 2008. On the difficulty of replicating human subjects studies in software engineering, in: ICSE '08 Proceedings of the 30th International Conference on Software Engineering. ACM Press, p. 191.
- Lynn, K.-M., Raphael, C., Olefsky, K., Bachen, C.M., 2003. Bridging the gender gap in computing: an integrative approach to content design for girls. *Journal of Educational Computing Research* 28, 143–162.
- Ma, L., Ferguson, J., Roper, M., Wood, M., 2007. Investigating the viability of mental models held by novice programmers, in: Proceedings of the 38th Technical Symposium on Computer Science Education. ACM Press, pp. 499–503.
- MacLaurin, M., 2009. Kodu: end-user programming and design for games, in: Proceedings of the 4th International Conference on Foundations of Digital Games. ACM Press, Article No. 2.
- MacMillan, D., 2012. The Rise of the “Brogrammer” [WWW Document]. Businessweek. URL <http://www.businessweek.com/articles/2012-03-01/the-rise-of-the-brogrammer> [Accessed 31-07-2012].
- Maheshwari, P., 1997. Teaching programming paradigms and languages for qualitative learning, in: 2nd Australasian Conference on Computer Science Education ACSE '97. pp. 32–39.
- Maloney, J., Burd, L., Kafai, Y.B., Rusk, N., Silverman, B., Resnick, M., 2004. Scratch: a sneak preview, in: Second International Conference on Creating, Connecting, and Collaborating Through Computing. Kyoto, Japan, pp. 104–109.
- Margolis, J., 2002. *Unlocking the clubhouse : women in computing*. MIT Press, Cambridge Mass. .
- Margulieux, L.E., Guzdial, M., Catrambone, R., 2012. Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications, in: ICER '12 Proceedings of the Ninth Annual International Conference on International Computing Education Research. ACM Press, Auckland, New Zealand, pp. 71–78.
- Marton, F., Säljö, R., 1976. On qualitative differences in learning: I - outcome and process. *British Journal of Educational Psychology* 46, 4–11.
- Mason, R., Cooper, G., 2012. Why the bottom 10% just can't do it - mental effort measures and implication for introductory programming courses, in: Proceedings of the Fourteenth Australasian Computing Education Conference

- (ACE2012). Australian Computer Society, Inc., Melbourne, Australia, pp. 187–196.
- Matlen, B.J., Klahr, D., 2010. Sequential effects of high and low guidance on children's early science learning, in: Gomez, K., Lyons, L., Radinsky, J. (Eds.), *Proceedings of the 9th International Conference on Learning Science*. International Society of Learning Science, Chicago, pp. 1016–1023.
- Mayer, R.E., 2001. *Multimedia Learning*. Cambridge University Press, New York, NY.
- Mayer, R.E., 2005. Cognitive theory of multimedia learning, in: Mayer, R.E. (Ed.), *The Cambridge Handbook of Multimedia Learning*. Cambridge University Press, New York, NY, pp. 31–48.
- Mayer, R.E., Anderson, R.B., 1991. Animations need narrations: an experimental test of a dual-coding hypothesis. *Journal of Educational Psychology* 83, 484–490.
- Mayer, R.E., Chandler, P., 2001. When learning is just a click away: Does simple user interaction foster deeper understanding of multimedia messages? *Journal of Educational Psychology* 93, 390–397.
- Mayer, R.E., Dow, G., Mayer, S., 2003. Multimedia learning in an interactive self-explaining environment: What works in the design of agent-based microworlds? *Journal of Educational Psychology* 95, 806–813.
- Mayer, R.E., Gallini, J.K., 1990. When is an illustration worth ten thousand words? *Journal of Educational Psychology* 82, 715–726.
- Mayer, R.E., Heiser, J., Lonn, S., 2001. Cognitive constraints on multimedia learning: When presenting more material results in less understanding. *Journal of Educational Psychology* 93, 187–198.
- Mayer, R.E., Mathias, A., Wetzel, K., 2002. Fostering understanding of multimedia messages through pre-training: evidence for a two-stage theory of mental model construction. *Journal of Experimental Psychology: Applied* 8, 147–154.
- Mayer, R.E., Moreno, R., 1998. A Cognitive Theory of Multimedia Learning: Implications for Design Principles URL: www.unm.edu/~moreno/PDFS/chi.pdf [Accessed: 20-07-2012].
- Mayer, R.E., Moreno, R., 2003. Nine ways to reduce cognitive load in multimedia learning. *Educational Psychologist* 38, 43–52.
- Mazlack, L.J., 1980. Identifying potential to acquire programming skill. *Communications of the ACM* 23, 14–17.
- McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B.-D., Laxer, C., Thomas, L., Utting, I., Wilusz, T., 2001. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin* 33, 125–180.
- McDaniel, M.A., Schlager, M.S., 1990. Discovery learning and transfer of problem-solving skills. *Cognition and Instruction* 7, 129–159.
- McDowell, C., 2002. Java meets Karel the Robot [WWW Document]. URL <http://www.cse.usc.edu/~charlie/pubs/jarel2000.pdf> [Accessed: 09-03-2011].
- McGill, M.M., 2012. Learning to program with personal robots: influences on student motivation. *ACM Transactions on Computing Education* 12, 1–32.
- McLaren, B.M., Isotani, S., 2011. When is it best to learn with all worked examples?, in: *AIED'11 Proceedings of the 15th International Conference on Artificial Intelligence in Education*. Springer-Verlag Berlin, pp. 222–229.

- McWhorter, W.I., O'Connor, B.C., 2009. Do LEGO® Mindstorms® motivate students in CS1?, in: SIGCSE '09 Proceedings of the 40th ACM Technical Symposium on Computer Science Education. ACM Press, p. 438-442.
- Mead, J., Gray, S., Hamer, J., James, R., St. Clair, C., Sorva, J., Thomas, L., 2006. A cognitive approach to identifying measurable milestones for programming skill acquisition. ACM SIGCSE Bulletin 38, 182–194.
- Merriënboer, J.J.G., Clark, R.E., Croock, M.B.M., 2002. Blueprints for complex learning: The 4C/ID-model. Educational Technology Research and Development 50, 39–61.
- Meyer, J.H.F., Land, R., 2003. Threshold Concepts and Troublesome Knowledge – Linkages to Ways of Thinking and Practising, in: Rust, C. (Ed.), Improving Student Learning – Ten Years On. OCSLD, Oxford.
- Middleton, D., 2010. Initial experiences teaching problem solving to computing freshmen (using robots), in: ACM SE '10 Proceedings of the 48th Annual Southeast Regional Conference. ACM Press, Article No. 105.
- Miller, G., 1956. The magical number seven, plus or minus two: some limits on our capacity for processing information. The Psychological Review 63, 81–97.
- MIT, 2010. Programming Concepts and Skills supported in Scratch. URL: <http://info.scratch.mit.edu/sites/infoscratch.media.mit.edu/files/file/ScratchProgrammingConcepts-v14.pdf> [Accessed: 10-07-2012]
- MIT, 2012a. About the Women's Technology Program [WWW Document]. Women's Technology Program - Massachusetts Institute of Technology. URL <http://wtp.mit.edu/> [Accessed 10-08-2012].
- MIT, 2012b. Scratch Gallery [WWW Document]. Scratch. URL <http://scratch.mit.edu/galleries/browse/newest> [Accessed 28-10-2012].
- MIT, C. for M.L., 2012c. What is App Inventor ? [WWW Document]. MIT App Inventor. URL <http://appinventor.mit.edu/explore/content/what-app-inventor.html> [Accessed 12-07-2012].
- MMV, 2004. Attitudes to ICT Careers. Report available from <http://www.mmv.vic.gov.au/Assets/537/1/AttitudesetoICTcareers.pdf> [Accessed 15-05-2012]
- Monash University, 2011. Social inclusion: diversifying the IT and information professions [WWW Document]. URL <http://www.thehindu.com/news/states/karnataka/article3291208.ece> [Accessed 10-08-2011].
- Moreno, R., Reisslein, M., Delgoda, G., 2006. Toward a fundamental understanding of worked example instruction: impact of means-ends practice, backward/forward fading, and adaptivity, in: Proceedings of the 36th Annual Frontiers in Education Conference. IEEE, San Diego, CA, pp. 5–10.
- Morling, S., McDonald, T., 2011. The Australian economy and the global downturn Part 2: The key quarters. The Australian Treasury. Report available from <http://www.treasury.gov.au/PublicationsAndMedia/Publications/2011/Economic-Roundup-Issue-2/Report/The-Australian-economy-and-the-global-downturn-Part-2-The-key-quarters> [Accessed 14-05-2012].
- Moskal, B., Lurie, D., Cooper, S., 2004. Evaluating the effectiveness of a new instructional approach, in: 35th SIGCSE Technical Symposium on Computer Science Education. ACM Press, Norfolk, Virginia, USA, pp. 75–79.

- Moura, I., 2009. Teaching a CS introductory course: an active Aapproach, in: Proceedings of Society for Information Technology & Teacher Education International Conference 2009. pp. 2308 – 2317.
- Mousavi, S.Y., Low, R., Sweller, J., 1995. Reducing cognitive load by mixing auditory and visual presentation modes. *Journal of Educational Psychology* 87, 319–334.
- Mullins, P., Whitfield, D., Conlon, M., 2009. Using Alice 2.0 as a first language. *Journal of Computing Sciences in Colleges* 24, 136–143.
- Murphy, L., Lewandowski, G., McCauley, R., Simon, B., Thomas, L., Zander, C., 2008. Debugging: the good, the bad, and the quirky -- a qualitative analysis of novices' strategies in. *Proceedings of the 39th SIGCSE technical symposium on Computer science education*. 163-167
- Mwangi, W., Sweller, J., 1998. Learning to solve ompare word problems: the effect of example format and generating self-explanations. *Cognition and Instruction* 16, 173–199.
- Neal, L.R., 1989. A system for example-based programming. *ACM SIGCHI Bulletin* 20, 63–68.
- Nestler, S., Egloff, B., Küfner, A.C.P., Back, M.D., 2012. An integrative lens model approach to bias and accuracy in human inferences: hindsight effects and knowledge updating in personality judgments. *Journal of Personality and Social Psychology* 103, 689–717.
- Ng, C., Das, S., Ching, L.T., Abdullah, M.C., 1998. Telework and gender in the Information Age: new opportunities for the developing world, in: *Proceedings of the Regional Conference on Gender and Technology in Asia*. Bangkok, Thailand.
- ICTA, 2007. National crisis: ICT skills shortage. *Science Alert Australia & New Zealand*. Available at <http://www.sciencealert.com.au/opinions/20070706-15970.html> [Accessed 14-01-2012]
- Nikula, U., Sajaniemi, J., Tedre, M., Wray, S., 2007. Python and roles of variables in introductory programming: experiences from three educational institutions. *Journal of Information Technology Education* 6, 199–214.
- Nisbett, R.E., 2003. *The geography of thought : how Asians and Westerners think differently-- and why*. Free Press, New York.
- North Sydney Institute of TAFE, 2011. Digi-girls [WWW Document]. URL <http://www.nsi.tafensw.edu.au/digigirls/index.htm> [Accessed 08-08-2011].
- Nourie, D., 2002. Teaching Java Technology with BlueJ [WWW Document]. URL <http://java.sun.com/features/2002/07/bluej.html>. [Accessed 7-12-2006]
- NSI TAFE, 2011. Digi-girls [WWW Document]. Digi-Girls. URL <http://www.nsi.tafensw.edu.au/digigirls/index.htm> [Accessed 08-08-2011].
- O’Grady, M.J., 2012. Practical problem-based learning in computing education. *ACM Transactions on Computing Education* 12, 1–16.
- Paas, F., Renkl, A., Sweller, J., 2003a. Cognitive Load Theory and instructional design: recent developments. *Educational Psychologist* 38, 1–4.
- Paas, F., Renkl, A., Sweller, J., 2004. Cognitive Load Theory: instructional implications of the interaction between information structures and cognitive architecture. *Instructional Science* 32, 1–8.
- Paas, F., Tuovinen, J.E., Tabbers, H., Van Gerven, P.W.M., 2003b. Cognitive Lload measurement as a means to advance Cognitive Load Theory. *Educational Psychologist* 38, 63–71.

- Paas, F., Van Merriënboer, J.J.G., 1993. The efficiency of instructional conditions: an approach to combine mental effort and performance measures. *Human Factors* 35, 737–743.
- Paas, F.G.W.C., Van Merriënboer, J.J.G., 1994. Variability of worked examples and transfer of geometrical problem-solving skills: A cognitive-load approach. *Journal of Educational Psychology* 86, 122–133.
- Papert, S., 1980. Mindstorms, children, computers and powerful ideas. Basic Books, New York.
- Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., Paterson, J., 2007. A survey of literature on the teaching of introductory programming. *ACM SIGCSE Bulletin* 39, 204–223.
- Peluso, E.M., Mauch, E., 2009. Incorporating Alice into a summer math and science outreach program, in: Proceedings of the 2009 Alice Symposium. ACM Press, Article No. 1.
- Peterson, L., Peterson, M.J., 1959. Short-term retention of individual verbal items. *Journal of Experimental Psychology* 58, 193–198.
- Piaget, J., 1971a. The theory of stages in cognitive development, in: Green, D.R., Ford, M.P., Flamer, G.B. (Eds.), *Measurement and Piaget*. McGraw-Hill, New York NY, pp. 1–11.
- Piaget, J., 1971b. Developmental stages and developmental processes, in: Green, D.R., Ford, M.P., Flamer, G.B. (Eds.), *Measurement and Piaget*. McGraw-Hill, New York NY, pp. 172–188.
- Piaget, J., 1973. *To Understand is to Invent*. Grossman, New York NY.
- Pillay, N., Jugoo, V.R., 2005. An investigation into student characteristics affecting novice programming performance. *Inroads - The SIGCSE Bulletin* 37, 107–110.
- Porter, R., 2006. Design Patterns in Learning to Program (PhD thesis). Flinders University, South Australia.
- Proulx, V.K., 2000. Programming patterns and design patterns in the introductory computer science course, in: Proceedings of the thirty-first SIGCSE technical symposium on Computer science education. pp 80-84.
- Psychometrics, 2009. Berger Aptitude for Programming Test [WWW Document]. Psychometrics I.T. Tests. URL <http://www.psy-test.com/Baptd.html> [Accessed 28-05-2012].
- Queensland Treasury, 2011. Population and Housing Profile - Gympie Regional Council. Office of Economic and Statistical Research, Queensland Treasury. Available from <http://www.oesr.qld.gov.au/products/profiles/pop-housing-profiles-lga/pop-housing-profile-gympie.pdf> [Accessed 19-10-2011].
- Raja, R., 2012. “Gangbang Interviews” and “Bikini Shots”: Silicon Valley’s Brogrammer Problem [WWW Document]. Mother Jones. URL <http://www.motherjones.com/media/2012/04/silicon-valley-brogrammer-culture-sexist-sxsw> [Accessed 31-07-2012].
- Rajaravivarma, R., 2005. A games-based approach for teaching the introductory programming course. *ACM SIGCSE Bulletin* 37, 98-102.
- Ramalingam, V., Wiedenbeck, S., 1997. An empirical study of novice program comprehension in the imperative and object-oriented styles. in: Proceeding: ESP '97 Papers presented at the seventh workshop on Empirical studies of programmers. 124-139.

- Randolph, J., 2007. Computer science education research at the crossroads: A methodological review of the computer science education research: 2000–2005. PhD Dissertation. Utah State University.
- Raouf, A., 2011. Programming Skills. LinkedIn Higher Education Teaching and Learning. Blog Post. Available at http://www.linkedin.com/groupItem?view=&gid=2774663&type=member&item=69930810&qid=b1b4fa03-ae03-4495-8455-7b37bae93cc7&trk=group_items_see_more-0-b-ttl [Accessed 01-10-2011]
- Reges, S., 2006. Back to basics in CS1 and CS2. ACM SIGCSE Bulletin 38, 293–297.
- Renkl, A., 2005. The worked-out example principle in multimedia learning, in: Mayer, R.E. (Ed.), *The Cambridge Handbook of Multimedia Learning*. Cambridge University Press, New York, NY, pp. 229–245.
- Repenning, A., Perrone, C., 2000. Programming by example: programming by analogous examples. Communications of the ACM 43, 90–97.
- Resnick, M., Kafai, Y.B., Maeda, J., 2003. A networked, media-rich programming environment to enhance technological fluency at after-school centers in economically-disadvantaged communities. Report from: National Science Foundation (Information Technology Research). Available from <http://web.media.mit.edu/~Emres/papers/scratch-proposal.pdf> [Accessed 9-03-2008].
- Restle, F., 1965. Significance of all-or-none learning. Psychological Bulletin 64, 313–325.
- Rittle-Johnson, B., Siegler, R.S., Alibali, M.W., 2001. Developing conceptual understanding and procedural skill in mathematics: an iterative process. Journal of Educational Psychology 93, 346–362.
- Robins, A., 2010. Learning edge momentum: a new account of outcomes in CS1. Computer Science Education 20, 37–71.
- Robogals, 2011. Robogals Rural & Regional (RRR) 2011 [WWW Document]. URL <http://www.robogals.org/about/what-we-do/rrr> [Accessed 10-07-2011].
- Rock, A., 2011. MaSH (Making Stuff Happen) [WWW Document]. URL <http://www.ict.griffith.edu.au/arock/MaSH/> [Accessed 23-08-2011].
- Rodger, S.H., Slater, D., Hayes, J., Lezin, G., Qin, H., Nelson, D., Tucker, R., Lopez, M., Cooper, S., Dann, W., 2009. Engaging middle school teachers and students with Alice in a diverse set of subjects. ACM SIGCSE Bulletin 41, 271–275.
- Rogerson, C., Scott, E., 2010. The fear factor: how it affects students learning to program in a tertiary environment. Journal of Information Technology Education 9, 147 – 171.
- Rößling, G., Freisleben, B., 2000. Experiences in using animations in introductory computer science lectures, in: SIGCSE '00 Proceedings of the Thirty-first SIGCSE Technical Symposium on Computer Science Education. ACM Press, pp. 134–138.
- Roumani, H., 2006. Practice what you preach, in: Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education. ACM Press, pp. 491–494.
- Ryan, R.M., Deci, E.L., 2000a. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. American Psychologist 55, 68–78.

- Ryan, R.M., Deci, E.L., 2000b. Intrinsic and extrinsic motivations: classic definitions and new directions. *Contemporary Educational Psychologist* 25, 54–67.
- Saha, A., 2012. Android programming with app inventor. *Linux Journal* 2012.
- Sajaniemi, J., 2002. An empirical analysis of roles of variables in novice-level procedural programs, in: *Proceedings of the IEEE 2002 Symposia on Human Centric Computer Languages and Environments*. IEEE Computer Society, pp. 37–39.
- Sanders, D., Dorn, B., 2003a. Jeroo: a tool for introducing object-oriented programming. in: *Proceedings of the 34th SIGCSE technical symposium on Computer science education*. 201-204.
- Sanders, D., Dorn, B., 2003b. Classroom experience with Jeroo. *Journal of Computing Sciences in Colleges* 18, 308–316.
- Sanders, D., Dorn, B., 2008. Introduction to computer programming with JEROO. *Journal of Computing Sciences in Colleges* 24, 132–133.
- Sanders, D., Dorn, B., 2011. Welcome to Jeroo! [WWW Document]. Jeroo. URL <http://home.cc.gatech.edu/dorn/jeroo>[Accessed 09-03-2009].
- Schmitt, N., Keeney, J., Oswald, F.L., Pleskac, T.J., Billington, A.Q., Sinha, R., Zorzie, M., 2009. Prediction of 4-year college student performance using cognitive and noncognitive predictors and the impact on demographic status of admitted students. *Journal of Applied Psychology* 94, 1479–1497.
- Schulte, C., Bennedsen, J., 2006. Teachers & learners: which to study? What do teachers teach in introductory programming? in: *Proceedings of the 2006 international workshop on Computing education research ICER '06*. 17-28.
- Schwartz, J., Stagner, J., Morrison, W., 2006. Kid's Programming Language (KPL), in: *International Conference on Computer Graphics and Interactive Techniques*. ACM Press, Boston, Massachusetts.
- Schwonke, R., Renkl, A., Krieg, C., Wittwer, J., Aleven, V., Salden, R., 2009. The worked-example effect: not an artefact of lousy control conditions. *Computers in Human Behavior* 25, 258–266.
- Seidman, R.H., 2009. Alice first: 3D interactive game programming. *ACM SIGCSE Bulletin* 41, 345–345.
- Selig, A., 2011. forum post. LinkedIn Higher Education Teaching and Learning. Available at http://www.linkedin.com/groupItem?view=&gid=2774663&type=member&item=69930810&qid=b1b4fa03-ae03-4495-8455-7b37bae93cc7&trk=group_items_see_more-0-b-ttl [Accessed 1-10-2011]
- Shaffer, C., Cooper, M.L., Alon, A.J.D., Akbar, M., Stewart, M., Ponce, S., Edwards, S.H., 2010. Algorithm visualization: the state of the field. *ACM Transactions on Computing Education* 10, 1–22.
- Shah, C., Burke, G., North, Sue, 2008. University IT Graduates for ICT occupations in Victoria 2008 to 2022. Multimedia Victoria, Victoria, Australia. Report available at http://www.mmv.vic.gov.au/Assets/2114/1/University_IT_graduates_for_ICT_occupations_in_Victoria_2008_to_2022.pdf [Accessed: 15-05-2012]
- Sheard, J., Hagan, D., 1998. Our failing students: a study of a repeat group. *ACM SIGCSE Bulletin* 30, 223–227.

- Sheard, J., Simon, S., Hamilton, M., Lönnberg, J., 2009a. Analysis of research into the teaching and learning of programming. ACM Press, p. 93.
- Sheard, J., Simon, S., Hamilton, M., Lönnberg, J., 2009b. Analysis of research into the teaching and learning of programming, in: ICER '09 Proceedings of the Fifth International Workshop on Computing Education Research Workshop. ACM Press, pp. 93–104.
- Shiffrin, R., Schneider, W., 1977. Controlled and automatic human information processing II. Perceptual learning, automatic attending and a general theory. *Psychological Review* 84, 127–190.
- Simon, 2007. A classification of recent Australasian computing education publications. *Computer Science Education* 17, 155–169.
- Simon, H., 1979. Information processing models of cognition. *Annual Review of Psychology* 30, 363–396.
- Simon, H.A., Gilmartin, K., 1973. A simulation of memory for chess positions. *Cognitive Psychology* 5, 29–46.
- Sorva, J., Karavirta, V., Korhonen, A., 2007. Roles of variables in teaching. *Journal of Information Technology Education* 6, 407–423.
- Southern Cross University, 2010. CSC00235 Applications Development Unit Information Guide 2010.
- Southern Cross University, 2011. 2010 Southern Cross University Annual Report. Southern Cross University, Lismore.
- Southern Cross University, 2012. ISY00243 Principles of Programming Unit Information Guide 2010.
- Stasko, J.T., Kraemer, E., 1993. A methodology for building application-specific visualizations of parallel programs. *Journal of Parallel and Distributed Computing* 18, 258–264.
- State Government Victoria, 2001. Reality Bytes: an in-depth analysis of attitudes about technology and career skills. [WWW Document]. URL <http://www.mmv.vic.gov.au/Assets/611/1/RealityBytes.pdf> [Accessed 20-07-2011].
- Strand-Cary, M., Klahr, D., 2008. Developing elementary science skills: instructional effectiveness and path independence. *Cognitive Development* 23, 488–511.
- Sweller, J., 1988. Cognitive load during problem solving: effects on learning. *Cognitive Science* 12, 257–285.
- Sweller, J., 1994. Cognitive load theory, learning difficulty, and instructional design. *Learning and Instruction* 4, 295–312.
- Sweller, J., 1999. Instructional Design in Technical Areas. The Australian Council for Educational Research Ltd, Camberwell, VIC.
- Sweller, J., 2003. Evolution of human cognitive architecture, in: *Psychology of Learning and Motivation*. Elsevier, pp. 215–266.
- Sweller, J., 2005. Implications of Cognitive Load Theory for Multimedia Learning, in: Mayer, R.E. (Ed.), *The Cambridge Handbook of Multimedia Learning*. Cambridge University Press, New York, NY, pp. 19–30.
- Sweller, J., 2010. Element interactivity and intrinsic, extraneous, and germane cognitive load. *Educational Psychology Review* 22, 123–138.
- Sweller, J., Cooper, G., 1985. The use of worked examples as a substitute for problem solving in learning algebra. *Cognition and Instruction* 2, 59–89.

- Sweller, J., Levine, M., 1982. Effects of goal specificity on means-ends analysis and learning. *Journal of Experimental Psychology: Learning, Memory and Cognition* 8, 463–474.
- Tasneem, S., 2012. Critical thinking in an introductory programming course. *Journal of Computing Sciences in Colleges* 27, 81–83.
- The Joint Task Force for Computing Curricula, 2005. *Computing Curricula 2005: The Overview Report*. A cooperative project of ACM, AIS and IEEE-CS.
- The LEGO Group, 2009. MINDSTORMS [WWW Document]. URL <http://mindstorms.lego.com/en-us/Default.aspx> [Accessed 28-08-2009].
- Theeuwes, J., Kramer, A.E., Hahn, S., Irwin, D.E., 1998. Our eyes do not always go where we want them to go: Capture of the eyes by new objects. *Psychological Science* 9, 379–385.
- Thomas, L., Ratcliffe, M., Robertson, A., 2003. Code warriors and code-a-phobes: a study in attitude and pair programming in: *Proceedings of the 34th Technical Symposium on Computer Science Education*. 363–367.
- Thomas, L., Ratcliffe, M., Thomasson, B., 2004. Scaffolding with object diagrams in first year programming classes. *ACM SIGCSE Bulletin* 36, 250–254.
- Tindall-Ford, S., Chandler, P., Sweller, J., 1997. When two sensory modes are better than one. *Journal of Experimental Psychology: Applied* 3, 257–287.
- TIOBE Software, 2011. TIOBE Programming Community Index - Long Term Trends [WWW Document]. URL <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html> [Accessed 22-08-2011].
- TIOBE Software, 2012. TIOBE Software General Information [WWW Document]. URL <http://www.tiobe.com/index.php/content/company/GeneralInfo.html> [Accessed 12-10-2012].
- Trapani, G., 2012. In war for talent, “brogrammers” will be losers [WWW Document]. CNN. URL <http://edition.cnn.com/2012/05/10/opinion/trapani-brogrammer-culture/index.html> [Accessed 31-07-2012].
- Tudoreanu, M.E., Kraemer, E., 2008. Balanced cognitive load significantly improves the effectiveness of algorithm animation as a problem-solving tool. *Journal of Visual Languages & Computing* 19, 598–616.
- Tuovinen, J.E., 2000. Optimising student cognitive load in computer education. *Proceedings of the Australiasian conference on Computing Education* 8, 235–241.
- Tuovinen, J.E., Sweller, J., 1999. Comparison of cognitive load associated with discovery learning and worked examples. *Journal of Educational Psychology* 91, 334–341.
- Turner, S., Bernt, P., Pecora, N., 2002. Why Women Choose Information Technology Careers: Educational, Social and Familial Influences. Presented at the Annual Meeting of the American Educational Research Association.
- UNDP, 2001. Human Development Report 2001 - Making New Technologies Work for Human Development. United Natons Development Programme, Brussels, Belgium.
- Universities Admissions Centre, 2010. UAC Guide.
- Uutting, I., Brown, N., Kölling, M., McCall, D., Stevens, P., 2012. Web-scale data gathering with BlueJ, in: *Proceedings of the Ninth Annual International Conference on International Computing Education Research*. ACM Press, Auckland, New Zealand, pp. 1–4.

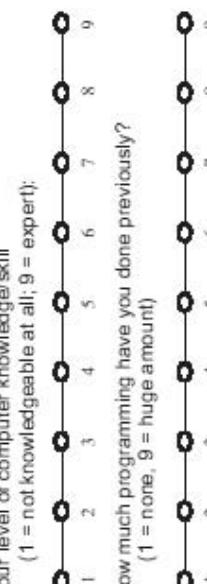
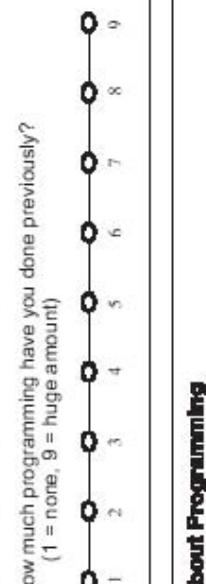
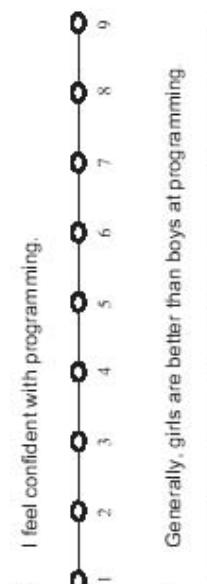
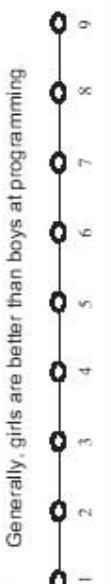
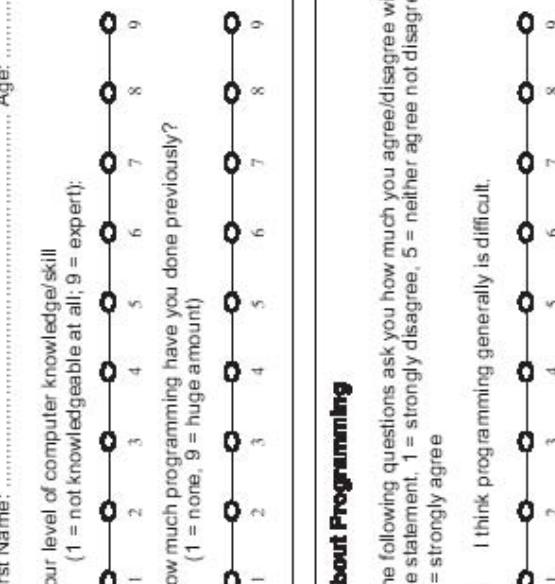
- Utting, I., Cooper, S., Kölking, M., Maloney, J., Resnick, M., 2010. Alice, Greenfoot, and Scratch -- a discussion. *ACM Transactions on Computing Education* 10, 1–11.
- Van Haaster, K., Hagan, D., 2004. Teaching and learning with BlueJ: an evaluation of a pedagogical tool, in: Proceedings of the Information Science & Information Technology Education Joint Conference.
- Van Merriënboer, J.J.G., Kester, L., 2005. The Four-Component Instructional Design Model: Multimedia Principles in Environments for Complex Learning, in: Mayer, R.E. (Ed.), *The Cambridge Handbook of Multimedia Learning*. Cambridge University Press, New York, NY, pp. 71–93.
- Van Merriënboer, J.J.G., Kester, L., Paas, F., 2006. Teaching complex rather than simple tasks: balancing intrinsic and germane load to enhance transfer of learning. *Applied Cognitive Psychology* 20, 343–352.
- Van Merriënboer, J.J.G., Sweller, J., 2005. Cognitive Load Theory and complex learning: recent developments and future directions. *Educational Psychology Review* 17, 147–177.
- Ventura, P.R., 2005. Identifying predictors of success for an objects-first CS1. *Computer Science Education* 15, 223–243.
- Victory College, 2011. Victory College - Mission [WWW Document]. URL <http://www.victorycollege.com.au/index.php?p=mission> [Accessed 21-20-2011].
- Walsh, T., 2008. USC to host Science, Engineering and Technology Expo [WWW Document]. URL <http://www.usc.edu.au/University/NewsEvents/News/2008News/SETExpo.htm> [Accessed 19-09-2011].
- White, G., Sivitanides, M., 2005. Cognitive differences between procedural programming and object oriented programming. *Information Technology and Management* 6, 333–350.
- White, G.L., Sivitanides, M.P., 2002. A theory of the relationships between cognitive requirements of computer programming languages and programmers' cognitive characteristics. *Journal of Information Systems Education* 13, 59–66.
- Whitehouse, G., Diamond, C., 2005. 'Hybrids' and the gendering of computing jobs in Australia. *Australasian Journal of Information Systems* 12, 79–89.
- Wilson, B.C., Shrock, S., 2001. Contributing to success in an introductory computer science course, in: Proceedings of the Thirty-second SIGCSE Technical Symposium on Computer Science Education - SIGCSE '01. pp. 184–188.
- Winslow, L.E., 1996. Programming pedagogy - A psychological overview. *SIGCSE Bulletin* 28, 17–22.
- Wolber, D., 2011. App inventor and real-world motivation, in: Proceedings of the 42nd ACM Technical Symposium on Computer Science Education. ACM Press, Dallas Texas USA, pp. 601–606.
- Xinogalos, S., Satratzemi, M., Dagdilelis, V., 2007. A comparison of two object-oriented programming environments for novices, in: CATE '07 Proceedings of the 10th IASTED International Conference on Computers and Advanced Technology in Education. ACTA Press Anaheim, CA, pp. 49–54.
- Zhang, X., 2010. Assessing students' structured programming skills with Java: the "Blue, Berry, and Blueberry" assignment. *Journal of Information Technology Education: Innovations in Practice* 9, 227 – 235.

- Zhu, H., Zhou, M., 2003. Methodology first and language second: a way to teach object-oriented programming. in Proceedings of the Conference on Object Oriented Programming Systems Languages and Applications. 140-147.
- Zhu, X., Simon, H.A., 1987. Learning mathematics from examples and by doing. *Cognition and Instruction* 4, 137–166.

APPENDIX A: SELF-SELECTING GIRLS

Appendix A.1	Pre-questionnaire – first IT Girls Day
Appendix A.2	Post-questionnaire – first IT Girls Day
Appendix A.3	Pre-questionnaire – second IT Girls Day
Appendix A.4	Post-questionnaire – second IT Girls Day
Appendix A.5	Mindstorms workshop script
Appendix A.6	Participant comments

A.1. Pre-questionnaire – first IT Girls Day

<p>Alice 3D Worlds:</p> <p>The following questions ask you how much you agree/disagree with the statement. 1 = strongly disagree, 5 = neither agree/not disagree 9 = strongly agree</p> <p>About You:</p> <p>First Name: Age:</p> <p>Your level of computer knowledge/skill (1 = not knowledgeable at all; 9 = expert):</p>  <p>How much programming have you done previously? (1 = none, 9 = huge amount)</p> 	<p>Robots:</p> <p>The following questions ask you how much you agree/disagree with the statement. 1 = strongly disagree, 5 = neither agree/not disagree 9 = strongly agree</p> <p>6. Programming robots sounds interesting to me.</p>  <p>7. Programming robots to do things is probably difficult.</p> 
<p>About Programming:</p> <p>The following questions ask you how much you agree/disagree with the statement. 1 = strongly disagree, 5 = neither agree/not disagree 9 = strongly agree</p> <ol style="list-style-type: none"> I think programming generally is difficult. I feel confident with programming. Generally, girls are better than boys at programming. 	

A.2. Post-questionnaire – first IT Girls Day

Alice 3D Worlds:	
<p>The following questions ask you how much you agree/disagree with the statement. 1 = strongly disagree, 5 = neither agree nor disagree 9 = strongly agree</p>	
<p>4. Programming with Alice is interesting to me.</p>	
<p>5. Programming with Alice to do things is difficult.</p>	
<p>6. I found the Alice software difficult to use</p>	
Mental Effort:	
<p>The next questions ask you about how much mental effort you estimated you used during the Alice workshop on a scale of 1 to 9, where 1 = no mental effort, and 9 = extreme mental effort.</p>	
<p>1. How much mental effort do you estimate you used on understanding and processing what you needed to do on each exercise?</p>	
<p>2. How much mental effort do you estimate you used on navigating and using the Alice programming software?</p>	
<p>3. How much mental effort do you estimate you used on learning from the program and reinforcing previous concepts?</p>	

<h3>IT Girls Day—Feedback Survey</h3>	
About You: First Name: Age:	
About Programming <p>The following questions ask you how much you agree/disagree with the statement. 1 = strongly disagree, 5 = neither agree nor disagree 9 = strongly agree</p>	
<p>1. I think programming generally is difficult.</p>	
<p>2. I feel confident with programming.</p>	
<p>3. Generally, girls are better than boys at programming.</p>	
<p>4. I enjoyed the Mindstorms NXT workshop</p>	
<p>5. I enjoyed the Alice 3D Worlds workshop</p>	

Mindstorms Robots:

The following questions ask you how much you agree/disagree with the statement, 1 = strongly disagree, 5 = neither agree nor disagree, 9 = strongly agree

4. Programming the robots is interesting to me.



5. Programming robots to do things is difficult.



6. I found the NXT software difficult to use

**Mental Effort:**

The next questions ask you about how much mental effort you estimated you used during the *Mindstorms* workshop on a scale of 1 to 9, where 1 = **no mental effort**, and 9 = **extreme mental effort**.

1. How much mental effort do you estimate you used on understanding and processing what you needed to do on each exercise?



2. How much mental effort do you estimate you used on navigating and using the NXT programming software?



3. How much mental effort do you estimate you used on learning from the program and reinforcing previous concepts?

**Your comments:**

What did you enjoy most about the workshops?

What could we have done better?

Any other comments?

A.3. Pre-questionnaire – Second IT Girls Day

IT Girls Day – Starting Survey									
Alice 3D Worlds									
<p>The following questions ask you how much you agree/disagree with each statement. 1 = strongly DISAGREE, 5 = neutral, 9 = strongly AGREE.</p>									
<p>A1. Your level of computer knowledge/skill: (1 = not knowledgeable at all, 9 = expert)</p>					<p>A6. Programming with Alice sounds interesting to me.</p>				
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
1	2	3	4	5	6	7	8	9	
<p>A2. How much programming have you done previously? (1 = none, 9 = huge amount)</p>					<p>A7. Programming with Alice sounds difficult.</p>				
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
1	2	3	4	5	6	7	8	9	
About You:									
First Name: _____ Age: _____									
<p>A8. Programming with Robots sounds interesting to me.</p>									
<p>A9. Programming with Robots to do things is probably difficult.</p>									
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
1	2	3	4	5	6	7	8	9	
About Programming									
<p>The following questions ask you how much you agree/disagree with each statement. 1 = strongly DISAGREE, 5 = neutral, 9 = strongly AGREE.</p>									
<p>A3. I think programming generally is difficult</p>					<p>A8. Programming with Alice sounds interesting to me.</p>				
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
1	2	3	4	5	6	7	8	9	
<p>A4. I feel confident with programming.</p>					<p>A9. Programming with Robots to do things is probably difficult.</p>				
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
1	2	3	4	5	6	7	8	9	
<p>A5. Generally, boys are better than girls at programming.</p>									
<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
1	2	3	4	5	6	7	8	9	
<p>There will also be a short questionnaire after the workshops. Your answers will help us to improve the program for other groups, so thank you!</p>									

A.4. Post-questionnaire – Second IT Girls Day

General:

B16. What, if anything, about your activities today, affected your perceptions of how easy or difficult programming can be?

The following questions ask you how much you agree/disagree with each statement. 1 = strongly DISAGREE, 5 = neutral, 9 = strongly AGREE.

B11. Programming the robots is interesting to me.

<input type="radio"/>	1	2	3	4	5	6	7	8	9
-----------------------	---	---	---	---	---	---	---	---	---

B12. Programming robots to do things is difficult.

<input type="radio"/>	1	2	3	4	5	6	7	8	9
-----------------------	---	---	---	---	---	---	---	---	---

B13. I found the NXT software difficult to use.

<input type="radio"/>	1	2	3	4	5	6	7	8	9
-----------------------	---	---	---	---	---	---	---	---	---

B14. Rank order these in terms of the mental effort you used when completing the activities in the *Mindstorms NXT* workshop. You should rank 1 for the activity that took the MOST mental effort, 2 for the next and 3 for the activity that took the LEAST mental effort.

- understanding/processing what you had to do on each exercise.
 - Navigating and using the Alice programming software
 - Learning from the program and reinforcing previous concepts.
- B15. Rank order these in order of importance to you
- understanding/processing what you had to do on each exercise.
 - Navigating and using the Alice programming software
 - Learning from the program and reinforcing previous concepts.

Mindstorms Robots.

B16. What, if anything, about your activities today, affected your perceptions of how easy or difficult programming can be?

--	--	--	--	--	--	--	--	--	--

B17. What did you enjoy most about the workshops?

--	--	--	--	--	--	--	--	--	--

B18. What could we have done better?

--	--	--	--	--	--	--	--	--	--

B19. Any other comments?

--	--	--	--	--	--	--	--	--	--

A.5. Pre-defined Mindstorms script

Mindstorms Robots:

Introduction to the Robots.

Mindstorms robots have two parts: Hardware and software.

1. Hardware: demonstrate
 - a. "NXT" brick – the brain of the robot
 - b. Touch sensor – enables the robot to 'feel'
 - c. Sound sensor – enables the robot to react to noise
 - d. Light sensor – enables the robot to detect light and colour
 - e. Ultrasonic sensor – enables the robot to measure distance to an object and react to movement
 - f. Servo motors – enables the robot to move
 - g. Introduce the concept of cables plugging into named ports on the NXT brick (A, B, C etc).
2. Software: Introduce the Mindstorms NXT software.
3. Process:
 First: build a robot (already done for you)
 Second: Create a program (what we are going to do) on the computer
 Third: Download it to the NXT brick
 Fourth: Use the program (run it).

Exercises

1. Use the robot to run a pre-loaded program. Use the touch sensor to interact with the program.
2. Build a program to make the robot "talk". Introduction to the timeline, programming blocks, properties and the compilation/downloading process.
3. Add a second sound – concept of sequence.
4. Add a display block. Introduction of concurrent timeline, and consolidation of choice of properties.
5. Create 'beating heart' animation.
 Introduction of "wait" time block to show different displays.
 Introduction of loop to repeat sections of blocks.
 Discussion about different types of loops (time, count, infinite)
6. Introduction of touch sensor event handling – specialized type of 'wait' wait until touch sensor pressed and then say something. Talk about selection of ports, and options (pressed, released, bumped)

7. Movement of arms using one servo motor. Introduction of ports for motors and other properties of the motor: brake/coast, direction, power and types of durations (rotations, time etc).
8. Make the robot walk: re-use of move block, with other options/using other motors.
9. Add sound sensor block to wait for sound before walking.

Give free time to use for own creations.

A.6. Comments from Participants

IT Girls Day 1 Responses

“What did you enjoy most about the workshops?”

1. Everything
2. Besides free food, enjoyed Alice ☺
3. There was no bossiness, and the things we did were fun
4. When robots randomly danced ☺
5. The Alice program its fun and creative
6. I love how its all hands-on. It's a great day ☺
7. All of it! I especially enjoyed the Alice program
8. Mindstorms robots
9. The ability to have hands-on experience with real-life programs and robots
10. It was fun and easy, and also interesting.

“What could we have done better?”

1. You couldn't have. (as in it was already good)
2. Explained more slowly of how to work the robots
3. Nothing, it was great
4. I don't know
5. I personally believe that everything was fine and did not need to be improved.
6. MORE ROBOTS!
7. More time with everything, especially Alice
8. Nothing
9. NOTHING!! ENJOYED ALL OF IT!! ☺
10. You guys did amazing fun, interesting, free gifts ☺ Nothing needs to be changed.

“Any other comments?”

1. I liked this day a lot
2. THANK YOU! ☺ <Picture of robot drawn>
3. Thank you for allowing us to participate on today and come along
4. IT WAS FUN! THANK YOU!
5. Alice isn't as smooth and natural when moving . But was a good day. ☺
6. THANK YOU FOR A GREAT DAY ☺
7. Thanks for the opportunity to use all the different programs.
8. No, I loved it
9. Thanks for having us [School Name redacted] girls here! It was fun, and interesting and a great day.
10. I had a rad day, thanks ☺

IT Girls Day 2 Responses

“What, if anything, about your activities today affected your perceptions of how easy or difficult programming can be?”

- 1 it was easier than I thought
- 2 easier than I thought it would be
- 3 now that I have done a little bit of programming, its easier than I thought
- 4 I thought it would be hard but it was fun and fairly easy
- 5 I thought they would be a lot harder
- 6 it took less than 7 minutes to be instructed what to do :) excellent
- 7 The simplicity, I thought it was far more complicated
- 8 I thought it would be a lot more difficult. But they make it so much more simple
- 9 Using the software and learning about it affected my perception
- 10 The easy use of both programs was great

“What did you enjoy most about the workshops?”

- 1 making my own 3D world (Alice)
- 2 playing with the robots and building a world in Alice
- 3 all of it
- 4 creating an Alice world
- 5 all of it!!
- 6 Alice!! Loving her xx
- 7 Alice!
- 8 I can't choose. I love them! =D
- 9 Using the Alice program
- 10 The robotics and the robotics program

“What could we have done better?”

- 1 ummm - no
- 2 UMMM
- 3 longer on each task ... And less explanation
- 4 nothing it was good :)
- 5 nothing it was awesome
- 6 Nothing. You can't make it better than it already is
- 7 N/A
- 8 NOTHING
- 9 Nothing, I liked it all
- 10 Nothing, it was wonderful

“Any other comments?”

2 no
3
4 thank you
5 Cya next year! =D ^_^ :p
6 Thank you for the experience. We are the IT Girls :)
7 Thank you :)
8 Thank you so much =) xx
9 I really loved the Alice program :)
10 Thank you so much for hosting this day.

APPENDIX B: IT FOR ALL GIRLS

Appendix B.1	Pre-workshop questionnaire 1
Appendix B.2	Pre-workshop questionnaire 2
Appendix B.3	Questionnaire given after Alice/Photoshop session
Appendix B.4	Questionnaire given after Mindstorms session
Appendix B.5	Post-workshop questionnaire 1
Appendix B.6	Post-workshop questionnaire 2

B.1. Pre-workshop questionnaire 1

NAME: _____ Year _____ Age: _____

ABOUT YOU			
Your level of computer knowledge or skill :	Little or none	average	expert
How much programming have you done?	none	lots!	
ABOUT CAREERS IN INFORMATION TECHNOLOGY (I.T.)			
I know a lot about careers in I.T.	Strongly disagree	neither	Strongly agree
I am considering a career in I.T.	Strongly disagree	neither	Strongly agree
ABOUT PROGRAMMING			
I think programming generally is difficult	Strongly disagree	neither	Strongly agree
I feel confident with programming	Strongly disagree	neither	Strongly agree
Generally, girls are better than boys at programming	Strongly disagree	neither	Strongly agree
Programming with Alice (3D Worlds) sounds interesting to me	Strongly disagree	neither	Strongly agree
Programming with Alice (3D Worlds) sounds difficult to me	Strongly disagree	neither	Strongly agree
Programming with robots sounds interesting to me	Strongly disagree	neither	Strongly agree
Programming robots to do things is probably difficult	Strongly disagree	neither	Strongly agree

B.2. Pre-workshop questionnaire 2

NAME: _____ Year: _____ Age: _____

ABOUT YOU				
Your level of computer knowledge or skill :	<input type="checkbox"/> Little or none	<input type="checkbox"/> average	<input type="checkbox"/> expert	
How much programming have you done?	<input type="checkbox"/> none	<input type="checkbox"/> Lots!		
ABOUT CAREERS IN INFORMATION TECHNOLOGY (I.T.)				
I know a lot about careers in I.T.	<input type="checkbox"/> Strongly disagree	<input type="checkbox"/> neither	<input type="checkbox"/> Strongly agree	
I am considering a career in I.T.	<input type="checkbox"/> Strongly disagree	<input type="checkbox"/> neither	<input type="checkbox"/> Strongly agree	
ABOUT PROGRAMMING				
I think programming generally is difficult	<input type="checkbox"/> Strongly disagree	<input type="checkbox"/> neither	<input type="checkbox"/> Strongly agree	
I feel confident with programming	<input type="checkbox"/> Strongly disagree	<input type="checkbox"/> neither	<input type="checkbox"/> Strongly agree	
Generally, boys are better than girls at programming	<input type="checkbox"/> Strongly disagree	<input type="checkbox"/> neither	<input type="checkbox"/> Strongly agree	
Programming with Alice (3D Worlds) sounds interesting to me	<input type="checkbox"/> Strongly disagree	<input type="checkbox"/> neither	<input type="checkbox"/> Strongly agree	
Programming with Alice (3D Worlds) sounds difficult to me	<input type="checkbox"/> Strongly disagree	<input type="checkbox"/> neither	<input type="checkbox"/> Strongly agree	
Programming with robots sounds interesting to me	<input type="checkbox"/> Strongly disagree	<input type="checkbox"/> neither	<input type="checkbox"/> Strongly agree	
Programming robots to do things is probably difficult	<input type="checkbox"/> Strongly disagree	<input type="checkbox"/> neither	<input type="checkbox"/> Strongly agree	

B.3. Questionnaire given after Alice/Photoshop session

NAME: _____ Year _____ Age: _____

WORKSHOP	Strongly disagree	neither	Strongly agree				
I enjoyed the Alice Workshop:	<input type="checkbox"/>						

ABOUT PROGRAMMING	Strongly disagree	neither	Strongly agree				
Programming with Alice is interesting to me	<input type="checkbox"/>						
Programming Alice to do things is difficult	<input type="checkbox"/>						
I found the Alice software difficult to use	<input type="checkbox"/>						
I had to think hard to understand what I had to do in each exercise	<input type="checkbox"/>						
I had to think hard to navigate and use the Alice software	<input type="checkbox"/>						
I had to think hard to learn from the program and understand concepts	<input type="checkbox"/>						
I think programming generally is difficult	<input type="checkbox"/>						
I feel confident with programming	<input type="checkbox"/>						

B.4. Questionnaire given after Mindstorms session

NAME: _____ Year _____ Age: _____

WORKSHOP							
	Strongly disagree	neither	Strongly agree				
I enjoyed the Mindstorms Workshop:	<input type="checkbox"/>						
	Strongly disagree	neither	Strongly agree				
I enjoyed the Photoshop Workshop:	<input type="checkbox"/>						

ABOUT PROGRAMMING							
	Strongly disagree	neither	Strongly agree				
Programming the robots is interesting to me	<input type="checkbox"/>						
	Strongly disagree	neither	Strongly agree				
Programming robots to do things is difficult	<input type="checkbox"/>						
	Strongly disagree	neither	Strongly agree				
I found the NXT software difficult to use	<input type="checkbox"/>						
	Strongly disagree	neither	Strongly agree				
I had to think hard to understand what I had to do in each exercise	<input type="checkbox"/>						
	Strongly disagree	neither	Strongly agree				
I had to think hard to navigate and use the NXT programming software	<input type="checkbox"/>						
	Strongly disagree	neither	Strongly agree				
I had to think hard to learn from the program and understand concepts	<input type="checkbox"/>						
	Strongly disagree	neither	Strongly agree				
I think programming generally is difficult	<input type="checkbox"/>						
	Strongly disagree	neither	Strongly agree				
I feel confident with programming	<input type="checkbox"/>						

B.5. Post-workshop questionnaire 1

NAME: _____ Year: _____ Age: _____

ABOUT CAREERS IN INFORMATION TECHNOLOGY			
	Strongly disagree	neither	Strongly agree
I know a lot about careers in I T	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	Strongly disagree	neither	Strongly agree
I would like to have a career in I T	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

ABOUT PROGRAMMING			
	Strongly disagree	neither	Strongly agree
I think programming generally is difficult	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	Strongly disagree	neither	Strongly agree
I feel confident with programming	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	Strongly disagree	neither	Strongly agree
Generally, girls are better than boys at programming	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

YOUR COMMENTS		
What did you enjoy most about the workshops?		
What could we have done better?		
Any other comments?		

B.6. Post-workshop questionnaire 2

NAME: _____ Year: _____ Age: _____

ABOUT CAREERS IN INFORMATION TECHNOLOGY			
	Strongly disagree	neither	Strongly agree
I know a lot about careers in I T	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	Strongly disagree	neither	Strongly agree
I would like to have a career in I T	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

ABOUT PROGRAMMING			
	Strongly disagree	neither	Strongly agree
I think programming generally is difficult	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	Strongly disagree	neither	Strongly agree
I feel confident with programming	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	Strongly disagree	neither	Strongly agree
Generally, boys are better than girls at programming	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

YOUR COMMENTS		
What did you enjoy most about the workshops?		
What could we have done better?		
Any other comments?		

APPENDIX C: AUSTRALIAN UNIVERSITIES SURVEY

C.1. Census document 2001

Trends in Novice Programming Languages

Trends in Novice Programming Languages

Introduction to Participants

"My name is Michael de Raadt. I am a PhD student from the University of Southern Queensland, Department of Mathematics and Computing Science, studying under Dr Mark Tolerman. I am hoping you can spare about 5 minutes of your time to answer questions. I am undertaking a census of programming languages used in first programming courses at all universities around Australia that offer courses accredited by the Australian Computer Society. The study's primary focus is the teaching of Non-Commercial languages, as opposed to teaching Wide Spread Commercial languages, in first programming courses. Hopefully, with your participation a clearer picture of trends in this area can be created to help those involved in teaching. The results of this study are not intended for commercial benefit. Your responses to questions will not be published, and will remain secure, private and confidential. Any findings of this study will be made available to you. I will ask you if you would a copy of the results of this study at the conclusion of the interview. You may withdraw from the study at any time".

Objectives

1. It is hoped that from this Census, a picture detailing the current use of Programming Languages in First Programming courses can be established.
2. Specifically of interest is the use of Non-Commercial languages and the proposed trend away from their use and towards languages that are used predominantly in Commercial settings.
3. It is also hoped that trends towards the use of specific languages may be revealed.

Method of Data Collection

1. This census is to be addressed to the Coordinator/Lecturers of First Programming courses at universities around Australia.
 2. The questions are to be asked in person over the phone.
 3. This survey is referred to as a census as it is hoped that Coordinator/Lecturers of all First Programming courses can be contacted and asked to participate.
-

Notes	Participant Name _____
	Participant E-Mail _____
	Participant Phone _____
	University _____
	School/Faculty _____
	Course Code _____
	Course Name _____

Census Questions

1. What type of student is your first programming course designed for (possibly multiple types)?

- Computer Science / IT
- Engineering
- Business
- Other _____

2. How many students are currently undertaking this course? _____

3. What programming language is being used in your first programming course?

- C
- C++
- Java
- VB
- Basic
- Pascal
- Non-Commercial Language _____

- Other _____

4. If Non-Commercial language used

Why was this language chosen (as opposed to a commercial language)?

If Wide Spread Commercial language used

Why was this language chosen (as opposed to a non-commercial language)?

Trends in Novice Programming Languages

5. Are there plans to change the first language?

- No
- Yes

If Yes, what programming language will be used?

- C
- C++
- Java
- VB
- Basic
- Pascal
- Non-Commercial Language _____
- Other _____

If Yes, why is this planned?

If Yes, when is this to be undertaken?

6. What language was taught previously in the course (most recent)?

- C
- C++
- Java
- VB
- Basic
- Pascal
- Non-Commercial Language _____
- Other _____

When did this change (Semester, Year) to the current language?

What was wrong with the previous language to cause this change?

What languages were taught prior to this?

Trends in Novice Programming Languages

7. Are environments and/or tools beyond simple editors and command line compilers used to support teaching of the language in practical sessions?
- Yes
If Yes, what environments/tools are used to support teaching (on what platform are these implemented)?
- Syntax based editor _____
- Debugging tool _____
- Object modeling environment _____
- Other _____
- No
If No, why are environments/tools not used?
8. What paradigm is being taught using the first language (regardless of what is traditionally thought to apply to this language) ?
- Procedural
 Object-Oriented
 Functional
 Logical
9. Core What languages are taught later in the degree(s) (multiples expected)?
- C
 C++
 Java
 VB
 Basic
 Pascal
 Mini/Teaching Language _____
- Other _____
10. Would you like to be sent a copy of the findings of this study?
- Yes
 No

C.2. Census Document 2003

Trends in Introductory Programming Languages : Census Version 2

Trends in Introductory Programming Languages Census Version 2

Introduction to Participants

My name is Michael de Raadt. I am a research student at the University of Southern Queensland. During semester 1 of 2001 a census was undertaken which created a picture of the languages, tools and paradigms being used in introductory programming courses at universities around Australia, and why instructors chose to use them. Results of this census have been published in Australia and overseas.

I am hoping you can spare about 5 minutes of your time to participate in the second running of this census. Hopefully, with your participation a clearer picture of trends in this area can be created to help those involved in teaching.

This study abides by ethical considerations implied by USQ. As such:

- The results of this study are not intended for commercial benefit.
- Your individual responses to questions will not be published, and will remain secure, private and confidential.
- Any findings of this study will be made available to you on request.
- You may withdraw from the study at any time.

Notes

Date

Participant Order Number

Previous Order Number

Participant Name

Participant E-Mail

Participant Phone

University

School/Faculty

Course Code

Course Name

Trends in Introductory Programming Languages : Census Version 2

Census Questions

1. What programming language is being used in your course?

- Java
- VB
- C++
- Haskell
- C
- Eiffel
- Delphi
- Ada
- JBase
- C#
- Other _____

2. How many students are currently undertaking this course? _____

3. IF COURSE WAS COVERED BY CENSUS PREVIOUSLY...

Previously the language used in this course was... _____

2001			2002			2003	
S1	S2	S3	S1	S2	S3	S1	

Was there any changes in language between 2001 and now?

- Yes (Go to 5)
- No (Go to 6)

Trends in Introductory Programming Languages : Census Version 2

4. IF THE COURSE IS BEING COVERED FOR THE FIRST TIME...

What language was taught immediately prior to the current language?

- None (new course)
- Java
- VB
- C++
- Haskell
- C
- Eiffel
- Delphi
- Ada
- JBase
- C#
- Other _____

When did this change (Semester, Year) to the current language / When did the new course start?

Semester _____ Year _____

What was wrong with the previous language to cause this change?

What languages were taught prior to this?

5. Why was this language chosen?

- Industry relevance / Marketable to students
- Pedagogical benefits of language
- Structure of degree/dept politics
- OO language
- GUI interface
- Availability/Cost to students
- Easy to find appropriate texts
- OS/Machine limitations of dept
- Other _____
- Other _____
- Other _____

Trends in Introductory Programming Languages : Census Version 2

6. What paradigm is being taught (regardless of what is traditionally thought to apply to the language being taught)?

- Procedural
- Object-Oriented
- Functional
- Logical

7. Do you encourage students in your course to use environments and/or tools beyond simple editors and command line compilers?

- No Tool
- VB IDE
- Delphi IDE
- Other IDE
- Functional Env
- BlueJ
- Other Tool

8. Are there plans to change the first language?

- No (Go to 9)
- Yes _____

What programming language will be used?

- Java
- VB
- C++
- Haskell
- C
- Eiffel
- Delphi
- Ada
- JBase
- C#
- Other _____

Why is this planned?

- Industry relevance / Marketable to students
- Pedagogical benefits of language
- Structure of degree/dept politics
- OO language
- GUI interface
- Availability/Cost to students
- Easy to find appropriate texts
- OS/Machine limitations of dept
- Other _____
- Other _____
- Other _____

When is this to be undertaken?

Semester _____ Year _____

Trends in Introductory Programming Languages : Census Version 2

9 What textbook(s) do you use in your course?

Title 1: _____

Author: _____

Title2: _____

Author: _____

10 How many hours per week do on-campus students spend in...

Lectures

- None
- 1 hour/month
- 1 hour/2 weeks
- 1 hour
- 2 hours
- 3 hours
- 4 hours

Tutorials

- None
- 1 hour/month
- 1 hour/2 weeks
- 1 hour
- 2 hours
- 3 hours
- 4 hours

Practicals

- None
- 1 hour/month
- 1 hour/2 weeks
- 1 hour
- 2 hours
- 3 hours
- 4 hours

11 How many years have you been involved in teaching of introductory programming?

Trends in Introductory Programming Languages : Census Version 2

- 12.** This next few questions are designed to discover what problem solving strategies are being taught to your students. Because problem solving can be interpreted in different ways, the questions are designed to be as specific as possible, but feel free to add comments to your answers at any stage.

In lectures do you teach the process of taking a problem statement and from it developing a program?

- Yes
- No

Procedural / Functional / Other	Object Oriented
Do you use a top-down functional decomposition methodology in your teaching? <input type="checkbox"/> Yes <input type="checkbox"/> No (No to next question)	Do you use an object identification methodology in your teaching? <input type="checkbox"/> Yes <input type="checkbox"/> No (No to next question)
Do you attempt to show how an expert programmer identifies goals recognized in a problem statement when decomposing a problem? <input type="checkbox"/> Yes <input type="checkbox"/> No	Do you attempt to show how an expert programmer identifies goals recognized in a problem statement when identifying objects? <input type="checkbox"/> Yes <input type="checkbox"/> No

Can you give examples of the problem solving strategies you teach?

What percentage of your **lecture** time throughout the semester do you spend teaching such problem solving strategies?

What percentage of your **tutorial** time throughout the semester do you spend teaching such problem solving strategies?

Other than the textbook, do you provide resources to students to implement such problem solving strategies?

- 13.** Are there any new introductory programming courses at your university?

- Yes (Follow up).
- No

- 14.** Would you like to be sent a copy of the findings of this study?

- Yes
- No

C.3. Census Questions 2010

Trends in Introductory Programming Languages; Census 2010

**Trends in Introductory Programming Languages
Census Version 2010**

Notes	Date <input type="text" value=" / / 20"/>	Participant Order Number <input type="text"/>
		Previous Order Number <input type="text"/>
	Participant Name <input type="text"/>	
	Participant E-Mail <input type="text"/>	
	Participant Phone <input type="text"/>	
	University <input type="text"/>	
	School/Faculty <input type="text"/>	
	Course/Unit Code <input type="text"/>	
	Course/Unit Name <input type="text"/>	
Information Sheet	<input type="text"/> (date)	
Consent Given	<input type="text"/> (date)	
Transcribed	<input type="text"/> (date)	

Census Questions

1. How many students are currently undertaking this unit/course in 2010?

2. What programming language is being used in the first programming unit/course that new students encounter?

- Java
 - VB
 - C++
 - Haskell
 - C
 - Eiffel
 - Delphi
 - Ada
 - Basic
 - C#
 - Scheme
 - Python
 - Javascript
 - Other _____
-

3. On a scale of 1-9 where 1 is extremely easy, 5 is neither easy nor difficult and 9 is extremely difficult, how difficult do you think this language is for students to learn?

1 2 3 4 5 6 7 8 9

4a. IF COURSE/UNIT WAS COVERED BY CENSUS PREVIOUSLY...

Were there any changes in language in the first programming unit/course between 2003 and now?

- Yes (Fill in below and then go to Q5)
- No (Go to Q6)

Previously the language used in this unit/course was _____

2003	2004			2005		
2006			2007			2008

Trends in Introductory Programming Languages; Census 2010

4b. IF THE COURSE/UNIT IS BEING COVERED in CENSUS FOR THE FIRST TIME...

What language was taught immediately prior to the current language?

- None (new course – go to Q6)
- Java
- VB
- C++
- Haskell
- C
- Eiffel
- Delphi
- Ada
- JBasic
- C#
- Scheme
- Python
- Javascript
- Other _____

When did this change (Semester, Year) to the current language / When did the new unit/course start?

Semester _____ Year _____

What languages were taught prior to this (if any)? [if none, go to 6]

5. What was the issue found with the previous language that led to this change?

6. Why was your current language chosen?

- Industry relevance / Marketable to students
- Pedagogical benefits of language
- Structure of degree/dept politics
- OO language
- GUI interface
- Availability/Cost to students
- Easy to find appropriate texts
- OS/Machine limitations of dept
- Other _____
- Other _____
- Other _____

Trends in Introductory Programming Languages; Census 2010

7. Are there plans to change the first language?
- No (Go to 8)
 - Yes (fill in below)

What programming language will be used?

- Java
- VB
- C++
- Haskell
- C
- Eiffel
- Delphi
- Ada
- JBoss
- C#
- Scheme
- Python
- Javascript
- Other _____

Why is this planned?

- Industry relevance / Marketable to students
- Pedagogical benefits of language
- Structure of degree/dept politics
- OO language
- GUI interface
- Availability/Cost to students
- Easy to find appropriate texts
- OS/Machine limitations of dept
- Other _____
- Other _____
- Other _____

When is this to be undertaken?

Semester _____ Year _____

Trends in Introductory Programming Languages; Census 2010

- 8.** Do you encourage students in the first programming unit/course to use environments and/or tools beyond simple editors and command line compilers?
- No Tool (go to Q11)
 - VB IDE
 - Delphi IDE
 - Other IDE
 - Functional Env
 - BlueJ
 - Java
 - Alice
 - Other Tool _____
- 9.** Why was this environment/tool chosen?
- Industry relevance / Marketable to students
 - Pedagogical benefits of environment
 - Structure of degree/dept politics
 - Supports OO development
 - GUI interface
 - Availability/Cost to students
 - Easy to find appropriate texts
 - OS/Machine limitations of dept
 - Comes with the language
 - Other _____
 - Other _____
 - Other _____
- 10a.** Is this environment/tool used
- (or an initial part of the first programming unit/course only? or
 - throughout, the first programming unit/course?)
- 10b.** Is the environment/tool used in any other units/courses in the degree?
- No
 - Yes.....How many units/courses? _____
Do you use this environment/tool in a different way in subsequent units/courses?

- 10c.** How difficult do you find the environment to use (on a scale of 1 to 9 where 1 is extremely easy, 5 is neither easy nor difficult, and 9 is extremely difficult)
- | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|

Trends in Introductory Programming Languages; Census 2010

- 10d.** On average, how difficult do you believe the students find the environment to use (on a scale of 1 to 9, where 1 is extremely easy, 5 is neither easy nor difficult and 9 is extremely difficult):

1 2 3 4 5 6 7 8 9

- 11.** What paradigm is being taught (regardless of what is traditionally thought to apply to the language being taught)?
- Procedural
 - Object-Oriented
 - Functional
 - Logical

- 12.** What textbook(s) do you use in your unit/course?

Title 1: _____

Author: _____

Title2: _____

Author: _____

- 13.** How many hours per week do on-campus students spend in...

Lectures	Tutorials	Practicals
<input type="checkbox"/> None	<input type="checkbox"/> None	<input type="checkbox"/> None
<input type="checkbox"/> 1 hour/month	<input type="checkbox"/> 1 hour/month	<input type="checkbox"/> 1 hour/month
<input type="checkbox"/> 1 hour/2 weeks	<input type="checkbox"/> 1 hour/2 weeks	<input type="checkbox"/> 1 hour/2 weeks
<input type="checkbox"/> 1 hour	<input type="checkbox"/> 1 hour	<input type="checkbox"/> 1 hour
<input type="checkbox"/> 2 hours	<input type="checkbox"/> 2 hours	<input type="checkbox"/> 2 hours
<input type="checkbox"/> 3 hours	<input type="checkbox"/> 3 hours	<input type="checkbox"/> 3 hours
<input type="checkbox"/> 4 hours	<input type="checkbox"/> 4 hours	<input type="checkbox"/> 4 hours

- 14.** Do you offer external or online delivery of your unit/course?

- No (go to next question)
- Yes (fill in below)

How do you achieve online delivery and assessment?

Trends in Introductory Programming Languages; Census 2010

15. How many years have you been involved in teaching of introductory programming?

16. The next few questions are about the process of solving NOVICE programming problems. Think about a novice programming problem in your unit. I will ask you first about YOURSELF and then your students.

When YOU solve a novice programming problem:

- A. What level of mental effort do you estimate you use on understanding and processing the problem statement (on a scale of 1 to 9, where 1 is no mental effort and 9 is extreme mental effort):

1 2 3 4 5 6 7 8 9

- B. What level of mental effort do you estimate you use on navigating or using the environment, tools or language (on a scale of 1 to 9, where 1 is no mental effort and 9 is extreme mental effort):

1 2 3 4 5 6 7 8 9

- C. What level of mental effort do you estimate you use on learning from the problem and reinforcing previous concepts (on a scale of 1 to 9, where 1 is no mental effort and 9 is extreme mental effort):

1 2 3 4 5 6 7 8 9

- 16b. When one of your average students solves a novice programming problem:

- A. What level of mental effort do you estimate they use on understanding and processing the problem statement (on a scale of 1 to 9, where 1 is no mental effort and 9 is extreme mental effort):

1 2 3 4 5 6 7 8 9

- B. What level of mental effort do you estimate they use on navigating or using the environment, tools or language (on a scale of 1 to 9, where 1 is no mental effort and 9 is extreme mental effort):

1 2 3 4 5 6 7 8 9

- C. What level of mental effort do you estimate they use on learning from the problem and reinforcing previous concepts (on a scale of 1 to 9, where 1 is no mental effort and 9 is extreme mental effort):

1 2 3 4 5 6 7 8 9

- 16c.** How do you believe these levels of mental effort differ for a student in your bottom 10% of students?

understanding and processing the problem statement: _____

navigating or using the environment, tools or language: _____

learning from the problem and reinforcing previous concepts: _____

Trends in Introductory Programming Languages; Census 2010

- 17.** This next few questions are designed to discover what problem solving strategies are being taught to your students. Because problem solving can be interpreted in different ways, the questions are designed to be as specific as possible, but feel free to add comments to your answers at any stage.

In lectures do you teach the process of taking a problem statement and from it developing a program?

- Yes
- No

Procedural/Functional/Other	Object Oriented
Do you use a top-down functional decomposition methodology in your teaching? <input type="checkbox"/> Yes <input type="checkbox"/> No (No to next question)	Do you use an object identification methodology in your teaching? <input type="checkbox"/> Yes <input type="checkbox"/> No (No to next question)
Do you attempt to show how an expert programmer identifies goals recognized in a problem statement when decomposing a problem? <input type="checkbox"/> Yes <input type="checkbox"/> No	Do you attempt to show how an expert programmer identifies goals recognized in a problem statement when identifying objects? <input type="checkbox"/> Yes <input type="checkbox"/> No

Can you give examples of the problem solving strategies you teach?

What percentage of your lecture time throughout the semester do you spend teaching such problem solving strategies?

What percentage of your tutorial time throughout the semester do you spend teaching such problem solving strategies?

Other than the textbook, do you provide resources to students to implement such problem solving strategies?

18. Are there any new introductory programming units/courses at your University?

- Yes (Follow up)
- No

Trends in Introductory Programming Languages.: Census 2010

19. Are there any other comments you would like to offer about the process of teaching introductory programming?



Census Summary of Results? Yes / No

APPENDIX D: STUDENT SURVEYS

D.1. Introductory email to students

Introductory Email to Students – SAMPLE ONLY

My name is Raina Mason and I am a research student and Associate Lecturer at SCU. I am currently conducting research into introductory programming education as part of my PhD research.

This email is to invite you to participate in my research. This will require you filling in two surveys – one in week 3 or 4 of Session 1 and one in week 12 or 13 of Session 1. Each survey will take approximately 5 minutes to fill out. Should you choose to participate, I will send you a reminder email to fill out the second survey, in Week 11.

An information sheet about this research is attached to this email. Should you choose to participate, please click on the “Survey Monkey” link below, and fill in the survey, as soon as possible (at the latest by the end of Week 3 of Session 1). Participation is voluntary but very much appreciated!

<http://www.surveymonkey.com/s/7K7GMTW>

Kind regards,

Raina Mason

D.2. Information sheet for survey

INFORMATION SHEET

My name is Raina Mason and I am conducting research for my Ph.D. This research project is called "Mental Effort Assessment of Introductory Computer Programming Problems" and is part of my thesis topic "Investigations into a new iterative approach, and environments to teach introductory computer programming". I am inviting all SCU students enrolled in the unit "Applications Development" in Session 1 2011 to participate in this project. This research involves the completion of two surveys, in Week 3 or 4 of Session 1, and in Week 12 or 13 of Session 1.

Things you should know:

- Each survey will take approximately 5 minutes to complete;
- I will need to collect your name on the surveys so the two surveys can be matched, but as soon as they are matched, names will be removed and all results will be de-identified;
- At no time will your unit assessor or tutors see the completed surveys, thus this cannot affect your grades or your position in your course;
- No risk is anticipated;
- Your participation is voluntary, and non-participation will not affect your results in any way;
- You may withdraw at any time, without penalty, and the results of your survey will be immediately withdrawn from the study;
- The results of this research may be published in a peer-reviewed journal and presented at conferences, but only group data will be reported;
- The results of the surveys will be kept for 7 years in secure storage, and then destroyed;
- Your consent to participate in this survey is implied by your completion of the online survey;
- Group results of the study will be emailed to you if you choose to receive them (see survey sheet).

To find out more about this research, you can contact me at

Raina Mason
SCU Business School,
Southern Cross University, Hogbin Drive, Coffs Harbour, NSW 2450
02 66593190
raina.mason@scu.edu.au

Or my supervisor:

Dr Graham Cooper
SCU Business School,
Southern Cross University, Hogbin Drive, Coffs Harbour, NSW 2450
02 66593327
graham.cooper@scu.edu.au

This research has been approved by the Human Research Ethics Committee at Southern Cross University. The Approval number is ECN-11-039. Should you have any complaints about the ethical conduct of this research, please contact:

The Ethics Complaints Officer
Southern Cross University
PO Box 157
Lismore NSW 2480
ethics.lismore@scu.edu.au

D.3. First part of survey

Mental Effort Assessment of Introductory Computer Programming

1.

* 1. Your name (used to match this survey to the second survey in Week 12):

2. If you would like a copy of the group results of this study, please include your email address here:

2. About You

* 1. Your age group:
 under 20 21 - 25 26 - 35 over 35

* 2. Your gender:
 male female

* 3. What is your first language (your language spoken at home as a child)?

* 4. Your degree:
 B. Information Technology B. Technology Education
 B. Applied Computing Grad Dip of Information Technology
 B. Business
 Other (please specify)

* 5. Your study mode:
 on-campus International on-campus external (distance)

* 6. Time at University:
 1st year second year more than 2 years

3. About Programming

Mental Effort Assessment of Introductory Computer Programming

*1. What is your skill level in using a computer as a tool (for example, for researching using the internet, and producing assignments)?

New to computers	some experience in the Internet and producing documents	can use computers for advanced tasks, and format professional documents
Computing skill	<input type="radio"/>	

*2. What is your computer programming experience (prior to starting this unit)?

Never programmed	simple programming	have built more complex programs	programmed in 2+ languages	extensive experience in programming
Programming experience	<input type="radio"/>			

*3. The next questions are regarding mental effort. Mental effort can be defined as the level of thinking effort that you use to understand, solve or learn from a problem. It is not necessarily how hard the problem is. Think about a programming problem you have been given in the first weeks of this unit. When you solved this problem, what level of mental effort do you estimate you used on:

no mental effort	average mental effort	extreme mental effort
Understanding and processing the problem statement?	<input type="radio"/>	
Navigating and using Visual Studio, Visual Basic and any other tools?	<input type="radio"/>	
Learning from the problem and reinforcing previous concepts?	<input type="radio"/>	

Thank you for participating in the first survey for this study. You will be sent an email in Week 12 of Session 1, asking you to fill out the second part of this study. Thank you again.

D.4. Second part of survey

Mental Effort Assessment of Introductory Computer Programming

1.

*1. Your name:

*2. What is your skill level in using a computer as a tool (for example, for researching using the internet, and producing assignments)?

New to computers	some experience in using the Internet and producing documents	can use computers for advanced tasks, and format professional documents
Computing skill	<input type="radio"/>	

*3. What is your computer programming experience?

Never programmed	simple programming	have built more complex programs	programmed in 2+ languages	extensive experience in programming
Programming Experience	<input type="radio"/>			

*4. The next questions are regarding mental effort. Mental effort can be defined as the level of thinking that you use to understand, solve or learn from a problem. It is not necessarily how hard the problem is. Think about a programming problem you have been given this week in this unit. When you solved the problem, what level of mental effort do you estimate you used on:

understanding and processing the problem statement?	no mental effort	average mental effort	extreme mental effort
navigating and using Visual Studio, Visual Basic and any other tools?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
learning from the problem and reinforcing previous concepts?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

*5. How do you think these three areas of mental effort affected you while studying in this unit?

Thank you for your participation in this study.

D.5. Student Comments

“How do you think these three areas of mental effort affected you while studying in this unit?”

1. They made the unit pretty boring and useless to me.
2. understanding the problem: I have found the objectives of the problem to be not clear enough, comments on the discussion board have helped. Visual Studio: ok . VB.net difficult too many possible ways to do similar processes, a trawl of the net shows that others have the same questions and that there various ways to apply a solution. Other tools: no prob,(except Blackboard)
3. Not very much, all three areas are easy to understand and don't require much mental effort.
4. processing was difficult due to my struggles with the textbook and its examples. This made me feel like giving up on one particular occasion. I don't think the stress taught me anything at all and only caused me to pass on these frustrations at home.
Navigation of applications and Elluminates were all useful tools and helped me to develop understanding and learn with an acceptable amount of mental stress Follow up workshops are a big help in learning and reinforcement of concepts and provided reassurance of my efforts in terms of learning materials.
5. Understanding and processing the problem statement was most difficult for me personally. It was only because i had never had any programming experience and everything was new to me. If i had a basic structure to work off, i think i could've gone alot better. Simple explanations of items such as integers or strings were not discussed. I learnt off my own problems and combined with the internet and my text was able to justify some of my answers.
6. At the beginning of unit it took more mental effort in all 3 areas, as the unit has progressed its become easier. My knowledge of the programming has improved, especially while using VB.
7. they have helped me to develop my understanding of programming and has changed my way of thinking when it come to understanding the processing statement although i was stressed up by the amount of work to be done.
8. I found that I did not experience any 'blockages' with regards to problem solving, the program logic was all straight forward and it was mainly a process of ascertaining the right syntax within which to frame the required logic statements, and correcting errors in programming was simplified by the interface.
9. I have struggled in this unit. Having not been to school or studied in thirty years, this has been both mentally and physically draining as I have spent many late nights with my head in the books, when I finally do get to bed my brain won't let me sleep. I had my exam yesterday and was an emotional wreck, I felt like I couldn't breath, I only answered half the questions. I am seriously wondering if this degree is worth all the stress, with my family life suffering.

10. I need a time to understand the problem and more time to solve it. Have some problems to understand codes.
11. I feel the most amount of Mental effort was spent on figuring out how to write the code to solve the problem. Easy enough to figure out HOW to solve the problem, but mental effort required in how to write the code to solve the problem.
12. I dont think this subject posed any real mental difficulty at all. I would have liked to have seen more in depth tasks. The subject started off looking like it was going to be difficult, but I think that was more to do with unfamiliarity with the software. Once I was comfortable with VB, none of the tasks seemed to push any of the basics learned. In terms of mental effort in my studies, this subject had virtually no impact on my other subjects. i would say it was the other way around which unfortunatley meant that I left the tasks for this subject till last as I was not worried about being able to complete them at all. As a result any issues I had with this subject were purely time related (I work full time and study full time) and nothing really to do with content.
13. I found most of the assessment didn't require a great deal of mental effort but other parts did. It was just a matter of troubleshooting to make the code work, that was the only time I really felt I had to really 'think' about it.
14. It was a challenge, at first the problem seemed almost impossible. I spent many many hours reading, researching and trawling the net for answers. For the major in CSC00235 I wrote three drafts, each with more than 500 lines of code. If the project didn't require as much mental effort, I probably wouldn't have worked so hard to find a solution. Not that I know the other students, but if I am an average student facing this project and it requires that amount of effort from me, then I figured many students would not of worked so hard to make their programme work therefore I saw an opportunity to go from average to above average. (thats my plan anyway)
15. Understanding and processing the problem statement was often difficult due to client requirements being ambiguous (but this part of the learning)
16. In the final weeks of this unit, I have been having such a hard time understanding the material that when attempting the assignments, I've either been in tears or drinking liquor. I've never felt more stupid.
17. became very stressed developed nervous tick in left eye many things were left undone by spending so much time on this unit.
18. Certainly some of the areas within the unit made me think but others seemed to flow fairly easily. I have done some coding and things with HTML, Javascript and fiddled with VB and macros in other areas of study so some problems were no issue. At times I had to thing about some aspects of code that I was writing but once I sorted out the flow I could get it to work. At some stages also I had to check reference material that jogged the memory on certain parts and things then fell into place.

APPENDIX E: INTERFACE EXPERIMENT

E.1. Careers Day Information

ICT: START HERE GO ANYWHERE



Victory College IT Careers Day

Information Technology (IT) is an exciting and fulfilling career that can include flexible hours, the opportunity to travel, and learning about new technology. Victory College and Southern Cross University as part of 2011 ICT Careers week would like to invite your child to participate in an "Information Technology Careers Day" to be held during the week starting 1st August, 2011.

Activities during the day will include:

- **Alice workshop:** an interactive environment that lets you create 3D animations that tell a story or interact with the user—Pixar or Disney animation style. It uses 3D graphics and is “drag-and-drop” (and a lot of fun!);
- **Careers information—** learn more about the wide variety of careers available in information technology;
- **Study paths—**learn more about the pathways available to study information technology at school, TAFE and University;
- **Mindstorms robot workshops—**opportunity to participate in some research involving programming robots (see attached information sheet);
- **Touch and Speech workshops—**opportunity to participate in some research involving touch screens and speech commands (see attached information sheet).

Years 7 to 12 will have the opportunity to participate in this program.

Date:	1st to 5th August, 2011
Place:	Victory College
Cost:	Free!

Return permission note by 27/7/2011 to attend.

From film to finance to pharmaceuticals, employers are waiting for people with the right technology skills. Design websites, develop games, run projects or run your own business. Start here, go anywhere!

E.2. Mindstorms Research Information Sheet



Mindstorms Robotics Research Information Sheet

Dear Parent/Guardian,

This information sheet is to invite your child to participate in a study about ways of teaching programming using Lego Mindstorms Robots, as part of the Victory College IT Careers Day. Ms Raina Mason is conducting this study as part of her PhD research at Southern Cross University into ways of teaching introductory programming. All students from year 7 to 12 at Victory College are being invited to take part in this study.

What is involved?

The study involves your child participating in a Mindstorms Robot workshop for approximately one period (40 minutes), where he/she will interact with and program the robots. Students will participate in small groups of 4 to 10 students. After the workshop, students will be asked to take a 20 minute questionnaire which asks about their experience of programming the robots, including some questions designed to measure their understanding of the workshop concepts. Please note that the workshop design is being assessed, *not your child*. Participating students will also be given morning tea (biscuits and juice) while taking the questionnaire.



Responsibilities

There is no risk to students associated with this research. Participation is voluntary, and students can choose to stop participating at any time. The research has been approved by the Human Research Ethics Committee at Southern Cross University. The approval number is ECN-11-126.

To participate, please complete and return the attached consent form. Please note that general data collected as part of the IT Careers day sign-up process may be matched to your child's Mindstorms Robots questionnaire.

If you would like a summary of the results of the research, this will be made available to you (please mark on the consent form).

How we respect your privacy and confidentiality:

Results of the questionnaires are confidential, and will not be used for any other purpose than this research. No individually identifiable data will be stored. The results of this research may be published in a peer-reviewed journal or presented at conferences, but only group data will be reported. All data that is collected will be securely held in a locked filing cabinet at Southern Cross University for 7 years, and then destroyed.

Questions?

Contact **Ms Raina Mason** by email raina.mason@scu.edu.au or her PhD supervisor **Dr Graham Cooper** on 02 6659 3327 or via email graham.cooper@scu.edu.au.

If you have any concerns about the ethical conduct of this research or the researchers, write to the following:

The Ethics Complaints Officer
Southern Cross University
PO Box 257
Lismore NSW 2480
email: ethics.lismore@scu.edu.au

All complaints are treated as confidential and will be handled as soon as possible.

Please keep this information sheet for your records after detaching the consent form. The consent form should be signed by both parent/guardian and child and returned to Victory College by 27th July, 2011. **Only students who have returned their consent forms will be able to participate in the Mindstorms Robotics Research workshops.**

E.3. Mindstorms Research Consent Form

Mindstorms Robotics Research Consent Form



Parent/Guardian to complete:

- I have read the information on the attached "Mindstorms Robotics Research Information Sheet" and have consulted with my child, who wishes to participate in the study;
 - I consent to my child named below participating in the study conducted by Southern Cross University;
 - I understand that general information from the evaluation surveys given as part of the IT Careers Day may be used as part of this study;
 - I understand the study will be conducted at the school and no cost or travel is involved.
 - I wish to receive a summary of the results of this study (please include your email address or mailing address below): _____
-

Please complete by printing:

Student's name: _____ Year: _____ Age: _____

Parent/Guardian's name: _____

Parent/Guardian's signature: _____ Date: ___ / ___ / ___

Student to complete:

- I have read the information on the attached "Mindstorms Robotics Research Information Sheet" and I agree to take part in the Southern Cross University research project.

Student's signature: _____ Date: ___ / ___ / ___

Please detach the information sheet and keep for your records. This consent form should be signed by both parent/guardian and child and returned to Victory College by 27th July, 2011. Only students who have returned their consent forms will be able to participate in the Mindstorms Robotics Research workshops.

E.4. IT Careers Day Permission Note

IT Careers Day

Parent/Guardian to complete:

I give permission for my child _____ Year _____ to attend the IT Careers Day at Victory College, during the week beginning 1st August 2011. There is no cost involved for the day.

Parent/Guardian's signature: _____ Date: ___ / ___ / ___

Student to complete:

Your Age: _____

Your level of computer knowledge or skill?

none	average	expert
------	---------	--------

How much programming have you done?

none	average	Lots!
------	---------	-------

I know a lot about information technology (IT)

definitely	neither	I know heaps!
------------	---------	---------------

I am considering a career in IT

no way	neither	totally!
--------	---------	----------

I think programming generally is difficult

no way	neither	totally!
--------	---------	----------

I feel confident with programming

no way	neither	totally!
--------	---------	----------



E.5. Mindstorms test – Subset Group

MINDSTORMS ROBOTS

Your answers to the following questions will help us to improve how we teach programming to other students. Please answer all [questions](#), as completely as you can.

Name: _____

Think about the mental effort you used (how hard you had to think) during the Mindstorms activities. Tick the boxes to answer the questions below.

How much mental effort did you need to use to [understand what you had to do in each exercise](#)?

No effort	Average effort	Extreme Effort
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How much mental effort did you need to use to [navigate and use the NXT programming software](#)?

No effort	Average effort	Extreme Effort
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How much mental effort did you need to use to [learn and understand concepts](#)?

No effort	Average effort	Extreme Effort
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

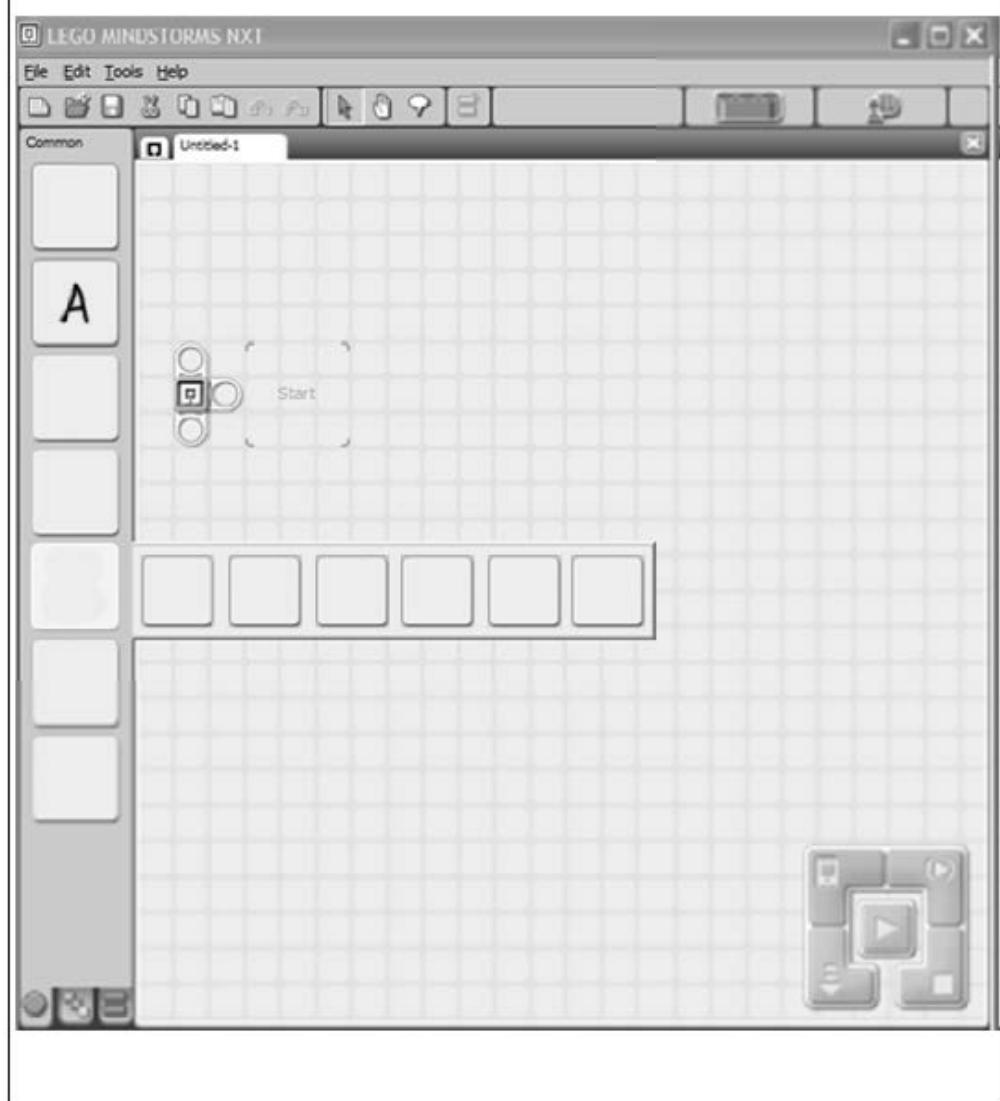
Describe briefly, in your own words, what each of the following program blocks does, when used in a Mindstorms program:







Below is a screenshot of the Mindstorms environment. Write on the screenshot to show where each of the program blocks should be placed. The first one is done for you.



The next questions are multiple choice—circle the correct letter for the answer; e.g: **(A)**

Which one of the following programs could make the robot say "Have a nice day"?

A.



B.



C.



D.

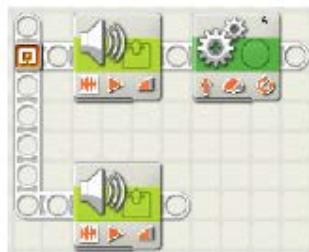


Which one of the following programs could make the robot say "Hello", move its arms 5 times and then say goodbye?

A.



B.



C.



D.



Which one of the following programs could make the robot say "Hello" 4 times?

A.



B.



C.



D.



Which one of the following programs could make the robot say something when its touch sensor is pressed?

A.



B.



C.



D.



Which of the following programs could make the robot repeatedly say "hello", and walk forward once the touch sensor is pressed?

A.



B.



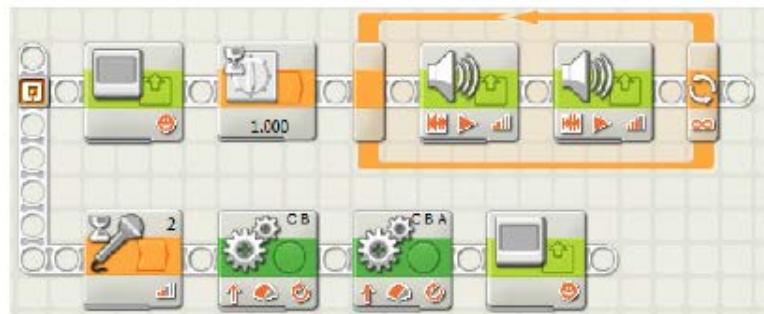
C.



D.



Describe in your own words what the following program could do:



Name: _____

Time Finished: _____

Program blocks like those in Mindstorms, can be used in different ways. Here are some program blocks that can be used to show dog tricks:



Bark



Beg



Fetch a ball



Lay Down



Play dead



Point



Shake hands

There is also a LOOP



a WAIT



and

a TIMELINE



Stick the programming blocks on the two pieces of paper given to build "programs" for the following dog tricks:

- dog fetches a ball twice, points, and then begs;
- dog shakes hands 5 times and then plays dead;
- dog lays down and waits until told to fetch a ball, then barks 3 times.

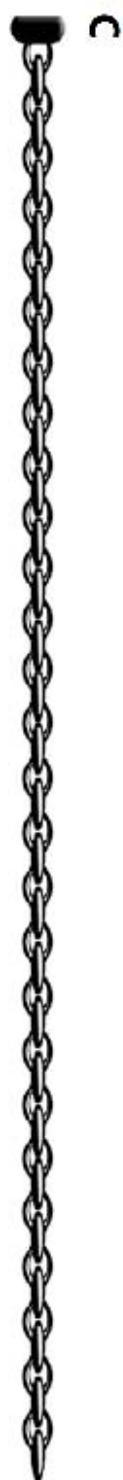
Name: _____

A



B





Name: _____

E.6. Mindstorms Test – Complete Group

MINDSTORMS ROBOTS

Your answers to the following questions will help us to improve how we teach programming to other students. Please answer all questions, as completely as you can.

Name: _____

Think about the mental effort you used (how hard you had to think) during the Mindstorms activities. Tick the boxes to answer the questions below.

How much mental effort did you need to use to understand what you had to do in each exercise?

No effort	Average effort	Extreme Effort
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How much mental effort did you need to use to navigate and use the NXT programming software?

No effort	Average effort	Extreme Effort
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

How much mental effort did you need to use to learn and understand concepts?

No effort	Average effort	Extreme Effort
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Describe briefly, in your own words, what each of the following program blocks does, when used in a Mindstorms program:



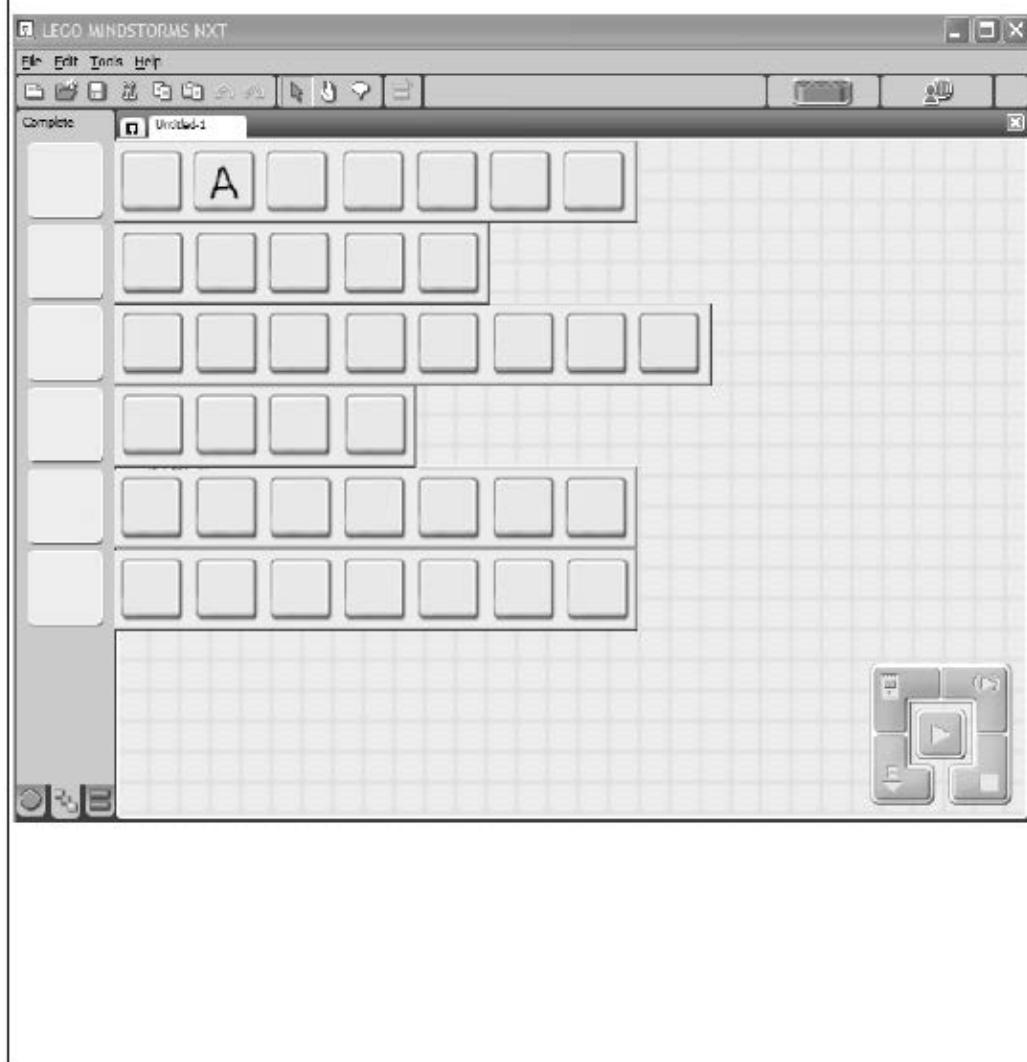




(Questions continue over the page)

Page 1

Below is a screenshot of the Mindstorms environment. Write on the screenshot to show where each of the program blocks should be placed. The first one is done for you.



The next questions are multiple choice—circle the correct letter for the answer; e.g: **(A)**

Which one of the following programs could make the robot say "Have a nice day"?

A.



B.



C.



D.



Which one of the following programs could make the robot say "Hello", move its arms 5 times and then say goodbye?

A.



B.



C.



D.



Which one of the following programs could make the robot say "Hello" 4 times?

A.



B.



C.



D.



Which one of the following programs could make the robot say something when its touch sensor is pressed?

A.



B.



C.



D.



Which of the following programs could make the robot repeatedly say "hello", and walk forward once the touch sensor is pressed?

A.



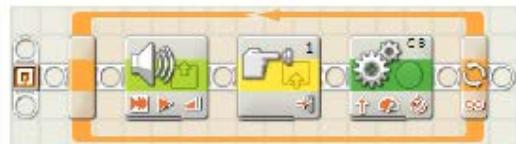
B.



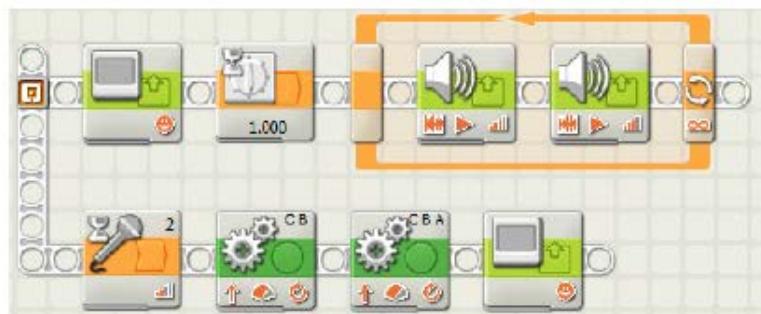
C.



D.



Describe in your own words what the following program could do:



Name: _____

Time Finished: _____

Program blocks like those in Mindstorms, can be used in different ways. Here are some program blocks that can be used to show dog tricks:



Bark



Beg



Fetch a ball



Lay Down



Play dead

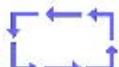


Point



Shake hands

There is also a LOOP



a WAIT



and

a TIMELINE

Stick the programming blocks on the two pieces of paper given to build "programs" for the following dog tricks:

- dog fetches a ball twice, points, and then begs;
- dog shakes hands 5 times and then plays dead;
- dog lays down and waits until told to fetch a ball, then barks 3 times.

Name: _____

A



B



Name: _____



C

E.7. IT Careers Day Evaluation



IT Careers Day - Evaluation

Name: _____ Year: _____

I know a lot about information technology (IT) I am considering a career in IT I think programming generally is difficult I feel confident with programming Programming with Alice 3D worlds is Programming with Alice 3D worlds is	definitely not neither I know heaps! <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> no way neither totally! <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> no way neither totally! <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> no way neither totally! <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> Not at all interesting neither Extremely interesting <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> Really easy neither Really difficult <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
--	---

If you participated in the Mindstorms Robots workshop – fill in the next two questions:

Programming with robots is ... Programming robots to do things is	Not at all interesting neither Extremely interesting <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> Really easy neither Really difficult <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
---	--

What did you enjoy most about the day?

What could we have done better?

Thank you

APPENDIX F: PUBLISHED PAPERS