Scaffolding Java Programming on a Mobile Phone for Novice Learners

By

Charity Chao Mbogo

A Dissertation Submitted For the Degree of
Doctor of Philosophy in the Department of Computer Science, Faculty of Science, University of Cape Town

September 2015

Supervised by:

Edwin Blake

Hussein Suleman

Declaration

I declare that this dissertation is my own original work. Where collaborations with other researchers are involved, or materials generated by other researchers are included, the parties and/or materials are acknowledged or are explicitly referenced as appropriate.

This work is being submitted for the degree of Doctor of Philosophy in Computer Science at the University of Cape Town, South Africa. This thesis has not been submitted to any other university or institution for any other degree or examination.

September 6th 2015	C.C.M
Date	Signature

Charity Chao Mbogo

Publications

Some ideas, figures and tables of this dissertation have previously appeared in the following five publications:

Mbogo, C., Blake, E., & Suleman, H. (2013, December 7 - 9). A Mobile Scaffolding Application to Support Novice Learners of Computer Programming. In *Proceedings of the Sixth International Conference on Information and Communications Technologies and Development*: Notes-Volume 2 (p. 84-87). ACM.

Mbogo, C., Blake, E., & Suleman, H. (2014, Feb 28 – Mar 2). Initial Evaluation of a Mobile Scaffolding Application that seeks to Support Novice Learners of Programming. In *Proceedings of the 10th International Conference on Mobile Learning*: (p. 175 – 182).

Mbogo, C., Blake, E., & Suleman, H. (2014, April 7- 9). Supporting the Construction of Programs on a Mobile Device: A Scaffolding Framework. In *Proceedings of 4th International Conference on M4D Mobile Communication for Development*: (p. 155).

Mbogo, C. (2014, October 8-10). Scaffolding Java Programming on a Mobile Phone for Novice Learners. In *Poster session at Grace Hopper Conference, 2014.*

Mbogo, C., Blake, E., & Suleman, H. (2015, Mar 14 – 16). Scaffolding Java Programming on a Mobile Phone for Novice Learners. In *Proceedings of the 11th International Conference on Mobile Learning*.

Dedication

To my family:

To mom for her love, patience and unwavering support.

To all of you for your love and laughter:

Mariam, Daniel, Jennifer, Shali, Hannah, Benjamin, Alice, Geoffrey, Joel, Shali, Naomi, Enoch, Shalom, Precious, Neema, James, Tabitha, Job, Hebron, Sarah, Ronnie, Mercy and Neema.

*****

To my late father:

I miss you every day. I have not forgotten the values you taught me of being teachable, hard work and persistence, all of which have seen me to this.

Acknowledgements

Heartfelt appreciation to my supervisors, Professor Edwin Blake and Associate Professor Hussein Suleman, for their invaluable support, guidance and mentorship right from the start. I am especially grateful that all our meetings always left me a better and more enlightened student. Special thanks to the late Gary Marsden for his support and guidance right from the time of applying to join the PhD. The PhD and fieldwork was funded by the Hasso Plattner Institute, and for the last part, by Google Anita Borg (EMEA) scholarship. I also received funding for conference facilitation from IPID, Google, ACM-W, Schlumberger's Faculty for the Future and the Department of Computer Science at UCT. I am immensely grateful for their financial support. Special thanks to the three examiners who took the time to read and review my thesis.

I appreciate the institutions that allowed access to their learners and to the learners who participated in the experiments at University of Cape Town, University of Western Cape, Kenya Methodist University and Jomo Kenyatta University of Agriculture and Technology. I would like to specifically thank the lecturers in the respective institutions for permission to access their learners: Audrey Mbogho, Bill Tucker, Robert Mutua, and John Njue.

All my colleagues in the ICTD lab have accompanied me on this journey through the hard days, lamentations, serious discussions and laughter. Special thanks to Chris Chepken, Mvurya Mgalla, Fiona Ssozi, Nasubo Ongoma, George Ng'ethe, Josiah Chavula, Aderonke Sakpere, Sinini Ncube and Lighton Phiri for their feedback on papers and posters. I thank the lecturers in the ICTD group for their constructive feedback during workshops and presentations. I am also thankful for Eve Gill for her support in arranging for conference travels.

Notable appreciation to Kenya Methodist University. Special thanks to Professor Robert Gateru, Dr Salesio Kiura, Ronald Wanyonyi, Lawrence Mwenda, Miriam Mwirebua, Daniel Muendo and Philip Oyier for their support and unselfish facilitation during my fieldwork in Kenya.

My love to my family whose unwavering love has been my rock and comfort. To my friends for their devotion and loyalty: Mercy Gacheri, Jemimah Nzale, Martha Mwangome, Vivian Ntinyari, Krystal Ndinda, and Doreen Areri. Thank you all for still being there even after my extended periods of silence while I was buried in work. I am grateful for my friends (Brian Laung, Krystal, Richard, Vivian, Doreen, Josh, and Michael), my brother James, and Professor Wallace Chigona, who took time to read sections of my thesis.

Most importantly, I thank God for the blessings that have been the last three years of good health, mental acuity, opportunities, rewards, great lessons, and the energy and ability to work on the project conscientiously, sometimes for long, long hours.

Table of Contents

Declaration ii
Publications iii
Dedication iv
Acknowledgements v
Table of Contents vi
List of Tables xi
List of Figures xiii
List of Abbreviations Used xvii
Abstract xviii
Chapter 1 Introduction 1
1.1 Scope of the Study 4
1.2 Problem Statement 4
1.3 Research Questions 4
1.4 Research Design and Approach 5
1.5 Research Contributions 8
1.6 Thesis Outline 9
Chapter 2 Constructivism and Programming 11
2.1 Choice of Constructivist Theory 11
2.2 Constructivist Theory 13
2.3 Constructivism in Programming 15
2.4 Scaffolding 16
2.5 Chapter Summary 18
Chapter 3 Related Work 20
3.1 Difficulties Faced by Novice Learners of Programming 20
3.2 Scaffolding Programming on PCs 23
3.2.1 New programming languages 23
3.2.2 Stand-alone applications 24
3.2.3 Teacher-learner architecture 25
3.2.4 Web-based applications 26
3.3 Using Mobile Phones for Learning 27
3.3.1 Limitations of mobile phones 29
3.4 Learning Programming using Mobile Phones 30
3.5 Summary of Gaps and Opportunities 33
Chapter 4 Design and Implementation of Scaffolding Techniques 36
4.1 Learner-Centered Design 36

4.2 Requirements 39
4.2.1 Learner-cited challenges 39
4.2.2 Limitations of mobile phones 42
4.3 Six-Level Scaffolding Framework 43
4.4 Implementation of Scaffolding Techniques 45
4.4.1 Learner challenge 1: Difficulty in connecting program parts into one 45
4.4.2 Learner challenge 2: Difficulty in debugging errors in programs 49
4.4.3 Learner challenge 3: Small screen size and small keypad of a mobile phone 51
4.4.4 Summary of scaffolding techniques 53
4.5 System Overview 55
4.5.1 First prototype 55
4.5.2 Second prototype 58
4.6 Example of a Simple Program Created Using the Scaffolding Techniques 60
4.7 Non-Scaffolded System Implementation 63
4.8 Chapter Summary 64
Chapter 5 Evaluation 65
5.1 Study Participants 65
5.2 Data Collection Methods 67
5.2.1 Electronic questionnaires 67
5.2.2 Computer logs 67
5.2.3 Video and image recordings 68
5.3 Internet-enabled mobile phones 68
5.4 Experiment Design 69
5.4.1 Programming tasks 69
5.4.2 Experiment procedure 70
5.5 Criteria to Address the First Research Question 71
5.5.1 Which scaffolding techniques were used to construct programs? 71
5.5.2 How were scaffolding techniques used to construct programs? 72
5.5.3 Qualitative Feedback 73
5.6 Criteria to Address the Second Research Question 73
5.6.1 Task Success 73
5.6.2 Time-on-task 74
5.6.3 Efficiency 75
5.6.4 Errors 75
5.6.5 Learnability 76
5.7 Summary of Criteria to Address Research Questions 76
5.8 Chapter Summary 77

Chapter 6 Results and Discussion 79
6.1 Participants and Experiments 79
6.1.1 First Experiment 79
6.1.2 Second Experiment 81
6.1.3 Third Experiment 81
6.2 Task Success 84
6.2.1 First Experiment 84
6.2.2 Second Experiment 89
6.2.3 Third Experiment 92
6.2.4 Summary of Task Success Results from all the Experiments 94
6.3 Time-on-task 96
6.3.1 Second Experiment 96
6.3.2 Third Experiment 101
6.3.3 Summary of Time-on-Task Results 106
6.4 Efficiency 107
6.5 Errors 109
6.5.1 Second Experiment 110
6.5.2 Third Experiment 113
6.5.3 Discussion: Error Results from the Second and Third Experiments 114
6.6 Scaffolding Techniques Used 115
6.6.1 Use of Static Scaffolding 116
6.6.2 Use of Automatic Scaffolding 120
6.6.3 User-initiated Scaffolding Techniques 123
6.6.4 Summary of Results on which Scaffolding Techniques were used 125
6.7 How the Scaffolding Techniques were used to Create Programs 126
6.7.1 Time-based Outliers 126
6.7.2 Learners who attempted to edit a chunk repeatedly before proceeding to the next one 126
6.7.3 Learners who cancelled the use of scaffolding techniques 127
6.7.4 Learners who unlocked the advanced interface 127
6.7.5 Summary of how scaffolding techniques were used 129
6.8 Chapter Summary 129
Chapter 7 Conclusion 132
7.1 Synthesis of Empirical Findings 132
7.1.1 Which of the theoretically-derived scaffolding techniques support construction of Java programs on a mobile phone? 132
7.1.2 What is the effect of using the scaffolding techniques to construct Java programs on a mobile phone? 132

7.2 Implications of the Study 133
7.2.1 Theory of constructivism 133
7.2.2 Design process 135
7.2.3 Novel scaffolding techniques and fading mechanisms 136
7.2.4 Understanding how learners use scaffolding techniques 137
7.2.5 Contribution to the field of ICT4D 137
7.3 Limitations of Research 138
7.4 Opportunities for Future Work 139
7.1.1 Extension of the system 139
7.1.2 Additional experiments 140
7.1.3 Evaluation with other existing tools 140
7.1.4 Model on fading of scaffolding 140
7.1.5 Use of the system to teach a class 140
REFERENCES 141
APPENDICES 152
Appendix A: Table of the Scaffolding Framework 152
Appendix B: Summary of Scaffolding Design Framework 158
Appendix C: Ethical Clearances 159
Appendix C1: Ethical clearance from University of Cape Town 159
Appendix C2: Permission to access learners at University of Cape Town 160
Appendix C3: Ethical clearance from Kenya Methodist University 161
Appendix D: Consent form signed by learners before participating in study 162
Appendix E: Questionnaires 163
Appendix E1: Experiment 1 questionnaire 163
Appendix E2: Experiment 2 and 3 questionnaire 172
Appendix E3: Experiment 3 questionnaire for control group 175
Appendix F: Screenshots of the second prototype with modifications 177
Appendix F1: Screenshot showing use of tabs in the main interface, a green run button at the top of the screen, and addition of ‘other class’ chunk 177
Appendix F2: Screenshot showing use of tabs in the editor 177
Appendix F3: Screenshot showing ‘public class’ keyword in main class disabled, showing menu options that can be selected to enable (left figure) or disable it (right figure) 178
Appendix F4: Screenshot showing instructions in the main class indicating that a user can proceed without creating the main class 178
Appendix F4: Screenshot showing a header dialog (left figure) that can be enabled using a menu option (right figure) 179
Appendix F5: Screenshot showing the Scanner class option (left figure) and the corresponding default text (right figure) that is to be edited and reused 179

Appendix F6: Screenshot showing the import statements that are automatically inserted in the imports chunk (left figure) and the resulting dialog box for user input when the program is compiled (right figure) 180
Appendix G: Raw Data for Number of Tasks 181
Appendix G1: Number of tasks attempted and completed per user for KeMU, Experiment 2 181
Appendix G2: Number of tasks attempted and completed per user for UWC, Experiment 2 181
Appendix G3: Number of tasks attempted and completed per user for JKUAT, Experiment 2 182
Appendix G4: Number of tasks attempted and completed per user for KeMU, Experiment 3 182
Appendix G5: Number of tasks attempted and completed per user for JKUAT, Experiment 3 183
Appendix H: Raw Data for Time-on-Task 184
Appendix H1: Time-on-task data for learners in Control and Experimental groups at UWC Experiment 2 184
Appendix H2: Time-on-task data for learners in Control and Experimental groups at JKUAT Experiment 2 186
Appendix H3: Time-on-task data for learners in Control and Experimental groups at KeMU Experiment 3 188
Appendix H4: Time-on-task data for learners in Control and Experimental groups at JKUAT Experiment 3 189
Appendix I: Raw Data for Verbatim User Feedback 191
Appendix I1: Survey responses at UWC, Experiment phase 2 191
Appendix I2: Survey responses at JKUAT, Experiment phase 2 192
Appendix I3: Survey responses at KeMU and JKUAT, Experiment phase 3 193
Appendix J: ERROR ANALYSIS 194
Appendix J1: Raw data showing error analysis of UWC data from the experimental group in the second Experiment 194
Appendix J2: Raw data showing error analysis of JKUAT data from the experimental group in the second Experiment 195
Appendix J3: Raw data showing error analysis of JKUAT data from the experimental group in the third Experiment 198

List of Tables

Table 2.1: Differences between behaviorism, cognitivism, and constructivism for learning 12
Table 4.1: Differences between User-Centered design and Learner-Centered design 37
Table 4.2: Table showing the designed scaffolding techniques, associated scaffolding type, fading capability and the related constructivist principle 54
Table 5.1: Total number of participants across the four institutions and the number of experiments conducted at each institution 66
Table 5.2: Number of experiments conducted at the four universities, the number of learners at each of the experiments, the groups involved in each experiment, and the data collection methods used at each experiment 70
Table 5.3: Summary of criteria to evaluate use of scaffolding techniques 72
Table 5.4: Table showing number of experiments, number of learners at each of the experiments, groups involved in each and the evaluation criteria addressed at each experiment 78
Table 6.1: Distribution of learners in the control and experimental groups across three experiments at four institutions 79
Table 6.2: Number of learners who completed each task at UCT, UWC and KeMU in the first experiment 84
Table 6.3: How UCT and UWC learners rated the different scaffolding techniques in terms of desirability to support construction of programs on a mobile phone 85
Table 6.4: How KeMU learners rated the different scaffolding techniques in terms of desirability to support construction of programs on a mobile phone 85
Table 6.5: Number of learners who attempted and completed each task in the Experimental groups at KeMU, UWC and JKUAT in the second Experiment 90
Table 6.6: Number of learners who attempted and completed each task in the Control groups at KeMU, UWC and JKUAT in the second Experiment 90
Table 6.7: Statistical task success results for attempted and completed tasks in the second Experiment 90
Table 6.8: Number of learners who attempted and completed tasks in the Experimental groups at KeMU and JKUAT in the third Experiment 93
Table 6.9: Number of learners who attempted and completed each task in the Control groups at KeMU and JKUAT in the third Experiment 93
Table 6.10: Statistical task success results for attempted and completed tasks in the third Experiment 93

Table 6.11: Statistical task success results in the second and third Experiments for attempted and completed tasks in Experimental and Control groups 95
Table 6.12: Statistical time-on-task results for all complete and incomplete tasks in the second Experiment 97
Table 6.13: Statistical time-on-task results per completed task in the second Experiment 99
Table 6.14: Statistical time-on-task results for all complete and incomplete tasks in the third Experiment 102
Table 6.15: Statistical time-on-task results per completed task in the third Experiment 105
Table 6.16: Statistical time-on-task results in the second and third Experiments for attempted and completed tasks in Experimental and Control groups 106
Table 6.17: Task completion rate, Average task time and Efficiency calculations for UWC, Experiment 2 108
Table 6.18: Task completion rate, Average task time and Efficiency calculations for JKUAT, Experiment 2 108
Table 6.19: Task completion rate, Average task time and Efficiency calculations for KeMU, Experiment 3 108
Table 6.20: Task completion rate, Average task time and Efficiency calculations for JKUAT, Experiment 3 109
Table 6.21: Statistical results on the mean number of errors for all tasks, first task, and second task at UWC and JKUAT in the second experiment 110
Table 6.22: Mean number of run-time errors and scaffolded errors in attempted tasks (per task) at UWC and JKUAT Second Experiment 112
Table 6.23: Statistical results on the mean number of errors for all tasks, first, second and third tasks at JKUAT in the third experiment 113
Table 6.24: Average number of run-time errors and scaffolded errors in attempted tasks in control and experimental groups, JKUAT Third Experiments 114
Table 6.25: Statistical error results from the second and third Experiments at UWC and JKUAT across all tasks and the first three tasks 115
Table 6.26: Summary of the sequence of program creation in the advanced interface by learners at JKUAT, Experiment 3 128

List of Figures

Figure 1.1: Graph showing percentage of respondents (Ghana N=2051, Kenya = 2000) who have the item at home in working order 2
Figure 1.2: Graph showing mobile phone ownership among Kenyan respondents (N = 2000) by education category 3
Figure 1.3: Flowchart showing mixed methods research design and research approach followed in the study 7
Figure 2.1: Comfort zone of proximal development (Anderson & Gegg-Harrison 2013) 17
Figure 3.1: TouchDevelop interface on a mobile device 32
Figure 3.2: Example of an AppInventor program 32
Figure 3.3: SAND IDE 32
Figure 3.4: Java Editor 32
Figure 4.1: LCD methodology followed in this study as adapted from the TILT model 39
Figure 4.2: Distribution of all respondents according to university 40
Figure 4.3: Distribution of all respondents according to course of study 40
Figure 4.4: Distribution of all respondents according to degree of study 41
Figure 4.5: Main interface showing only the main class activated 47
Figure 4.6: Main class default code 47
Figure 4.7: Creating the main class 47
Figure 4.8: Method default code 47
Figure 4.9: Creating a method 47
Figure 4.10: dialog for default statements 47
Figure 4.11: Main class completed (in green) and header button activated 49
Figure 4.12: Unrestricted interface 49
Figure 4.13: Error prompt 51
Figure 4.14: Incorrect creation of return statement 51
Figure 4.15: Error prompt indicating incorrect use of return statement 51
Figure 4.16: Hints displayed when creating main class 51
Figure 4.17: Main Class example displayed after clicking on a menu item 51
Figure 4.18: Creating method 53
Figure 4.19: Full program as was last saved 53
Figure 4.20: Prompt for unchanged main class 53

Figure 4.21: System overview of the first prototype showing the scaffolding techniques in blue at the main interface and at the editor 56
Figure 4.22: System overview of the second prototype showing the scaffolding techniques in blue at the main interface and the editor 59
Figure 4.23: Screenshot showing the main interface of the second prototype with three tabs, a button for other class, and a quick-access run button 60
Figure 4.24: Editor with three tabs for instructions, editing and full program 60
Figure 4.25: The header dialog in the editor 60
Figure 4.26: Restricted main class keywords at the editor 60
Figure 4.27: Main class active 61
Figure 4.28: Main class clicked 61
Figure 4.29: Editing main class 61
Figure 4.30: Error prompt 62
Figure 4.31: Header activated 62
Figure 4.32: Saved on device 62
Figure 4.33: Header clicked 62
Figure 4.34: Full program 62
Figure 4.35: Creating header 62
Figure 4.36: Main method clicked 62
Figure 4.37: Default statements 62
Figure 4.38: Edit main method 62
Figure 4.39: Completed program parts 63
Figure 4.40: Completed full program 63
Figure 4.41: Output of program after compilation 63
Figure 4.42: Interfaces for the non-scaffolded application 63
Figure 5.1: Samsung Galaxy Pocket S5300 used during the experiments 68
Figure 6.1: First Experiment session at UCT 80
Figure 6.2: First Experiment session at UWC 80
Figure 6.3: Programming task attempted by learners in the first experiment 80
Figure 6.4: Class session at KeMU during the first experiment 81
Figure 6.5: Second Experiment session at KeMU 82
Figure 6.6: Second Experiment session at UWC 82
Figure 6.7: Programming tasks attempted by learners in the second Experiment at UWC, KeMU and JKUAT 82

Figure 6.8: Third Experiment session at KeMU 83
Figure 6.9: Third Experiment session at JKUAT 83
Figure 6.10: Programming tasks attempted by learners in the third Experiment 83
Figure 6.11: Full program written within the main class chunk where only the class name is required 87
Figure 6.12: Inappropriate completion of the main class 87
Figure 6.13: Dialog box showing default statements 87
Figure 6.14: Learner typing statement from scratch 87
Figure 6.15: Video screenshot of a learner at the main interface of the application 87
Figure 6.16: Video screenshot showing soft keypad covering half the screen 87
Figure 6.17: Box plots showing time-on-task for incomplete tasks, completed tasks and total time for Experimental and Control group at UWC, Experiment 2 96
Figure 6.18: Box plots showing time-on-task distribution for incomplete tasks, completed tasks and total time on task for Experimental and Control groups at JKUAT, Experiment 2 97
Figure 6.19: Box plot showing time on completed tasks per-task in the Experimental and Control group at UWC, Experiment 2 98
Figure 6.20: Box plot showing time on completed tasks per-task for Experimental and Control group at JKUAT, Experiment 2 99
Figure 6.21: Box plots showing time-on-task distribution for incomplete tasks, completed tasks and total time on task for Experimental and Control groups at KeMU, Experiment 3 101
Figure 6.22: Box plots showing time-on-task distribution for all incomplete tasks, completed tasks and total time on task for Experimental and Control groups of JKUAT, Experiment 3 102
Figure 6.23: Box plot showing task completion rates across completed tasks for Experimental and Control groups at KeMU, Experiment 3 104
Figure 6.24: Box plot showing task completion rates across completed tasks for Experimental and Control groups at JKUAT, Experiment 3 104
Figure 6.25: Error prompt showing incorrect creation of the main class 112
Figure 6.26: Error prompt showing incorrect completion of the for-loop 112
Figure 6.27 A program showing the Keywords ‘String’ and ‘System’ written in lower case ‘s’ (in bold) 112
Figure 6.28: Error prompts encountered within the main class in italics 114
Figure 6.29: Error prompts encountered within the main method in italics 114
Figure 6.30: Comparison of use of static scaffolding techniques between incomplete and complete programs at UWC, KeMU and JKUAT in Experiments 2 and 3 116

Figure 6.31: A section of a learner's logs showing several attempts at adding an extra line within the main class chunk 118
Figure 6.32: Progression of use of static scaffolding techniques in incomplete and complete programs at JKUAT Experiment 3 119
Figure 6.33: Use of automatic scaffolding techniques in incomplete and complete programs at JKUAT Experiment 3 121
Figure 6.34: Sequence of program creation showing the statement dialog cancelled twice while creating the main method, and then enabled on the third attempt 121
Figure 6.35: Progression of use of automatic scaffolding techniques in incomplete and complete programs at JKUAT Experiment 3 123
Figure 6.36 : Use of user-initiated scaffolding techniques in all programs across the four experiment sessions 124
Figure 6.37: Graph showing when the full program was viewed and the average view per learner at UWC, Experiment 2 124
Figure.6.38: Graph showing when the full program was viewed and the average view per learner at JKUAT, Experiment 3 125

List of Abbreviations Used

IT – Information Technology

PC – Personal Computer

3D – Three-dimensional

IDE – Integrated Development Environment

LCD – Learner-Centered Design

UCD – User-Centered Design

API – Application Programing Interface

CSE – Computer Science Education

ZPD – Zone of Proximal Development

CZPD – Comfort Zone of Proximal Development

GUI – Graphical User Interface

ICT – Information Communication Technology

ACM – Association for Computing Machinery

IEEE – Institute of Electrical and Electronics Engineers

ICT4D – Information and Communication Technologies for Development

Abstract

The ubiquity of mobile phones provides an opportunity to use them for learning programming beyond the classroom. This would be particularly useful for novice learners of programming in resource-constrained environments. However, limitations of mobile phones, such as small screens and small keypads, impede their use as typical programming environments. This study proposed that mobile programming environments could include scaffolding techniques specifically designed for mobile phones, and designed based on learners' needs.

A six-level theoretic framework was used to design scaffolding techniques to support construction of Java programs on a mobile phone. The scaffolding techniques were implemented on an Android platform. Using the prototype, three experiments were conducted with 182 learners of programming from four universities in South Africa and Kenya. Evaluation was conducted to investigate: (i) which scaffolding techniques could support the construction of Java programs on a mobile phone; and (ii) the effect on learners of using these scaffolding techniques to construct Java programs on a mobile phone. Data was collected using computer logs, questionnaires, and image and video recordings.

It was found that static scaffolding, such as a program overview and constructing a program one part at a time, supported the construction of programs on a mobile phone. It was also found that automatic scaffolding, such as error prompts and statement dialogs, and user-initiated scaffolding, such as viewing of the full program while creating parts of a program, supported learners to construct programs on the mobile phone. The study also found that the scaffolding techniques enabled learners to attempt and complete more tasks than a non-scaffolded environment. Further, the scaffolding techniques enabled learners to complete programs efficiently, and captured syntactical errors early during program creation. The results also indicated that after the initial familiarization with the scaffolded environment, the scaffolding techniques could enable faster completion of programs. Learners' feedback indicated that they found the scaffolding techniques useful in supporting programming on a mobile phone and in meeting learners' needs.

This study provides empirical evidence that scaffolding techniques specifically designed for mobile phones and designed based on learners' needs could support the construction of programs on a mobile phone.

Chapter 1 Introduction

Computer programming is a difficult subject for most learners of programming. Research indicates this to be a universal problem, especially among novice learners (Piteira & Costa 2012; Watson & Li 2014). Novice learners of programming may be defined as learners enrolled in a university-level, introduction to programming course (Maleko et al. 2012). This research adopts this definition of a novice learner. The learning difficulties in the subject indicate that some programming skills are beyond the novice learners' efforts. Scaffolding refers to support provided so that the learner can engage in activities that would otherwise be beyond their abilities or their unassisted efforts (Jackson et al. 1998; Wood et al. 1976). For example, an adult could support a child who is learning how to walk by holding the child's hands. Likewise, support structures erected around an upcoming building enable a construction worker to access a higher part of the building. Both the adult's hands and the building's support structures offer scaffolding. Thus, a novice's learning process can also be scaffolded in different ways.

A child learns how to walk by actually trying to walk. Similarly, programming is best learnt by attempting to write programs and not just reading or memorizing programs. This principle of learning by writing programs is embedded in the constructivist theory of knowledge building, which focuses on learning through doing (Fosnot 2005). As a child is learning to walk, the adult's hands can be withdrawn when the child is more stable on their feet, but the adult's hands should be available to the child if they still need support. Thus, the constructivist theory supports the notion of scaffolding because, as learners construct programs, they can be provided with support that could later fade away. Because it underlies the principles of learning by doing and scaffolding, constructivism was used as the theoretical framework for this research.

In order to contribute towards tackling learning difficulties in programming, novice learners can be supported to construct programs while they are outside the classroom. This makes any such support to be additional to the learner's classroom learning, and not a replacement. Further, learners may not always have access to the school's computer laboratories where they can practice programming. Support to learners outside the classroom can be provided using PC-based applications. Indeed, several studies have offered scaffolded environments on PC platforms targeting novice learners of programming, for example, 3D environments such as Alice (Dann et al. 2011), and teacher-learner assessment environments such as Test My Code (Vihavainen et al. 2013).

However, most learners who are in resource-constrained environments, such as in parts of Africa, have limited access to PCs while they are outside the classroom. In fact, in many developing

countries, people are much more likely to use computers at school or at work than to own them at home. For example, a survey conducted in Ghana and Kenya to investigate the ownership of information and communication technologies at home showed that only 10% of respondents in Ghana and 5% in Kenya have a computer at home (Bowen & Goldstein 2010). This is illustrated in Figure 1.1. The limited access to PCs outside the classroom aggravates the learning difficulties faced by such learners because resource constraints present their own challenges in developing a good programming foundation (D'Souza et al. 2008). Further, research conducted in Tanzania highlights that one of the contributors to learners struggling in programming is lack of adequate access to computers, which limits hands-on learning (Apiola et al. 2011).

The ubiquity of mobile devices provides an opportunity to use them as a resource to support learning of programming beyond the classroom. This is especially because, in developing countries, mobile devices hold enormous promise as the single ICT most likely to deliver education, and to do so in a sustainable, equitable and scalable basis (Traxler 2011). Mobile devices include laptops, tablets and mobile phones. Of these, mobile phones are the most widely used mobile devices among learners in developing countries (Kafyulilo 2012). Further, Figure 1.1 shows that the percentage of respondents in Ghana and Kenya who own mobile phones was higher in comparison to the percentage of respondents who own computers at home. In addition, Figure 1.2 shows a graph from a study conducted in Kenya, indicating that most of the respondents studying for university degrees or higher own mobile phones (Hannah 2010). For these reasons, the mobile phone was selected as the resource that could be used to construct programs outside the classroom.

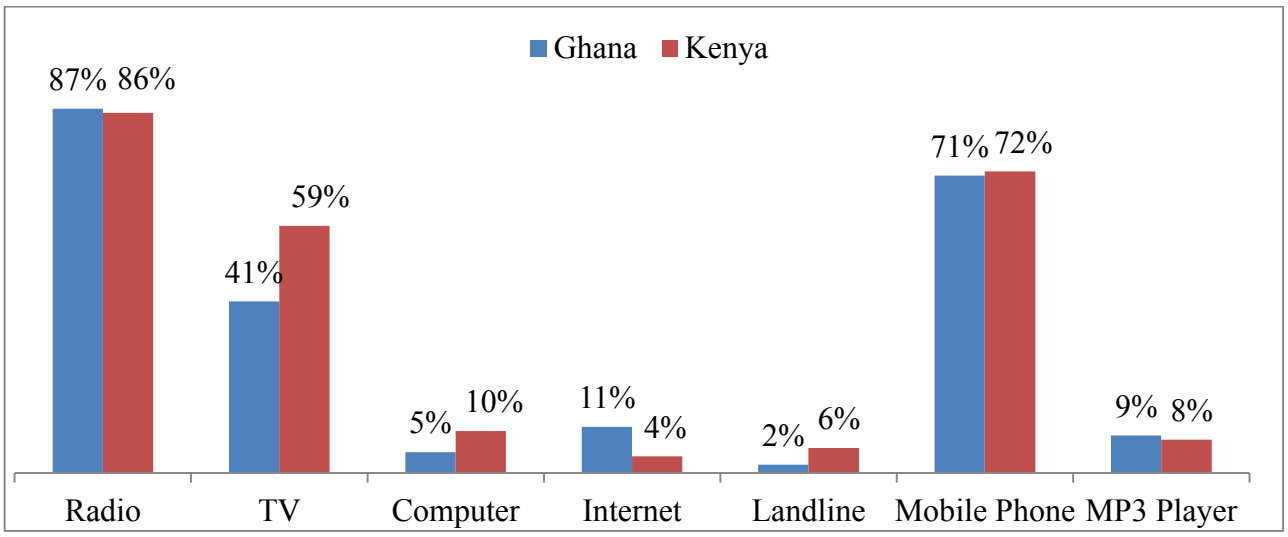


Figure 1.1: Graph showing percentage of respondents (Ghana N=2051, Kenya = 2000) who have the item at home in working order

Source of Data: (Bowen & Goldstein 2010)

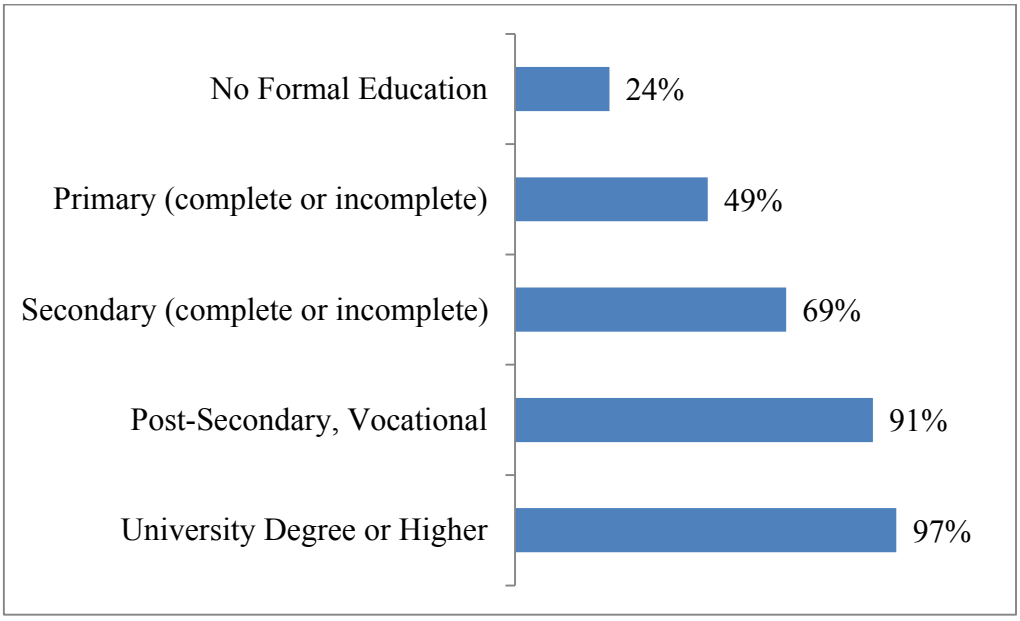


Figure 1.2: Graph showing mobile phone ownership among Kenyan respondents (N = 2000) by education category

Source of Data: (Hannah 2010)

However, limitations of mobile phones, such as a small screen size and a small keypad, impede their use as typical programming environments. To deal with these limitations, and for handheld devices to become effective learning tools, the unique design challenges inherent in such a system must be understood (Luchini et al. 2002). In fact, even when designing for Web-based GUIs that run on a mobile device, it has been suggested that interfaces on mobile devices should be tailored for such devices (Alonso-Ríos et al. 2014; Zimmerman & Yohon 2009).

There are mobile programming environments that can be used by novice learners. Some, such as SAND IDE[1], can be used to create standard programs. Others, such as App Inventor[2], can be used to create mobile applications. However, mobile programming environments such as SAND IDE mostly mimic PC IDEs and do not address the limitations of mobile phones. Further, it was not the aim of this study to support the creation of mobile applications, but to support the creation of standard programs that would typically be created in an introduction to programming class.

In addition to addressing limitations of mobile phones, the challenges faced by learners of programming should be considered. This is because addressing these challenges maximizes the potential of meeting learners' needs. The aim of this research was to support novice learners by

[1] http://goo.gl/708IuE

[2] http://appinventor.mit.edu/explore/

scaffolding the construction of programs. Therefore, in providing scaffolding, the needs of learners can be placed at the center of the design process. Such an approach was defined as learner-centered design, which claims that software can embody scaffolding that can address learners' needs (Soloway et al. 1996). Further, learner-centered design understands learners as a unique group of novices who are trying to learn the content and work practices of unfamiliar domains (Luchini et al. 2002). In addition, learner-centered design should provide tools that provides learners with an active process of learning by doing where learners manipulate the material they are learning (Quintana et al. 2000; Soloway et al. 1996). Such an approach is embodied in the constructivist theory, which is the underlying theoretical framework for this research.

Consequently, this research proposed that programming environments on mobile phones could include scaffolding techniques that are specifically designed for mobile phones, and designed based on learners' needs.

1.1 Scope of the Study

The study focused on introduction to programming courses taught using Java. Java was selected as the language for construction of programs because it was the common language taught across the institutions that participated in this research. In addition, most novice learners learn an introductory programming course using object-oriented programming languages such as Java (Black et al. 2013). Further, the programs that were used in the study are programs that were created in an introduction to programming class. This focus was deemed appropriate because the aim of the study was to support novice learners of programming. The learners who participated in the study were from institutions in South Africa and Kenya. The institutions from these two locations were selected because of their convenience in terms of having established contacts. Further, the two locations were deemed appropriate since they are both developing countries where learners could have limited access to PCs and laptops outside the classroom. Lastly, the focus of the study was on the use of a mobile phone as a programming environment and not the use of other mobile devices such as tablets, or the use of desktops and laptops.

1.2 Problem Statement

The aim of this research was to identify which scaffolding techniques could support Java programming on a mobile phone and, further, to evaluate the effect on learners of using these scaffolding techniques to construct Java programs on a mobile phone.

1.3 Research Questions

To address the research problem, two research questions were posed:

1. *Which of the theoretically-derived scaffolding techniques support construction of Java programs on a mobile phone?*

To design scaffolding techniques that could support construction of Java programs on a mobile phone, a six-level scaffolding framework was used. This framework consisted of theoretical guidelines that were followed in order to design specific scaffolding techniques. This scaffolding framework is discussed in Chapter 4. To address this research question, first, an analysis was conducted to identify which of the scaffolding techniques were used to construct programs. Then the scaffolding techniques were analyzed to identify how learners used them to construct programs. Further, learners were asked: if they found the scaffolding techniques useful; which scaffolding techniques they found useful; and to comment on their experiences while using the scaffolding techniques.

2. *What is the effect on learners of using the scaffolding techniques to construct Java programs on a mobile phone?*

By learners constructing programs using the derived scaffolding techniques and some constructing programs using a non-scaffolded environment, the study investigated the effect of the scaffolding techniques. The data from the two groups of learners was analyzed to measure: the number of tasks completed; the amount of time spent on the tasks; the errors encountered while constructing the tasks; and the efficiency with which the programs were constructed. In addition, the learnability of the scaffolded environment was analyzed.

1.4 Research Design and Approach

To conduct the research, a mixed methods design was used. Mixed methods research involves collecting, analyzing, and interpreting quantitative and qualitative data in a single study or in a series of studies that investigate the same underlying phenomenon (Leech & Onwuegbuzie 2007). Mixed methods research is based on the idea that the use of quantitative and qualitative approaches in combination provides a better understanding of a research problem than either approach alone (Azorín & Cameron 2010).

To address the first research question, qualitative data was collected in order to analyze which of the specifically-designed scaffolding techniques were used to construct programs. Further, qualitative data was collected in order to understand the perception of learners of the scaffolding techniques, and their experiences while using the scaffolding techniques. In addition, quantitative data was collected in order to analyze the frequency of use of the scaffolding techniques.

To address the second research question, data was collected to measure quantities such as number of completed tasks and time-on-task. Collectively, both research questions were addressed using both quantitative and qualitative data.

This study followed a combination of a multiphase design and embedded design of the mixed methods research. A multiphase design combines both sequential and concurrent use of qualitative and quantitative data over a period of time (Creswell & Clark 2007). An embedded design collects and analyzes both quantitative and qualitative data within a traditional quantitative or qualitative design (Creswell & Clark 2007). Figure 1.3 shows these phases in blue. First, qualitative data was collected during the design phase. This qualitative data informed the design of the scaffolded environment. Thereafter, both qualitative and quantitative data was collected and analyzed in the evaluation phase. Further, Figure 1.3 summarizes the overall research approach as described next.

The aim of this research was to contribute towards tackling learning difficulties in programming. Therefore, the first step was to understand the challenges that learners face in the subject. These challenges were elicited from learners of programming using an online survey and were used as part of the requirements in the design process. The elicited learner challenges and limitations of mobile phones were integrated within a six-level scaffolding framework to select scaffolding techniques that could support Java programming on a mobile phone. The framework was based on a theory-driven model that has four main levels (Quintana et al. 2004): challenges experienced by learners; cognitive type of the learning challenges; scaffolding guidelines; and scaffolding strategies that implement the guidelines. In addition to these four levels, two other levels were added in order to accommodate: a model for categorizing the type of scaffolding to use (Jackson et al. 1998); and selection of specific scaffolding techniques that could support construction of Java programs on a mobile phone (Mbogo et al. 2014).

To implement the selected scaffolding techniques in a mobile programming environment, an Android prototype was developed. Android was selected as the platform of implementation because it is open source, and it has an 85% market share among smartphone users (Hornyak 2014). Apart from the scaffolded environment, a non-scaffolded prototype was designed to be used in the experiments.

These prototypes were used in three experiments with a total of 182 learners of introductory programming courses taught using Java, from four institutions in South Africa and Kenya. In these experiments, learners attempted Java programming tasks and data was collected using computer logs, questionnaires, video recordings and image recordings. In the first experiment, only an experimental group participated in the study, where all the learners used the scaffolded environment. In the second

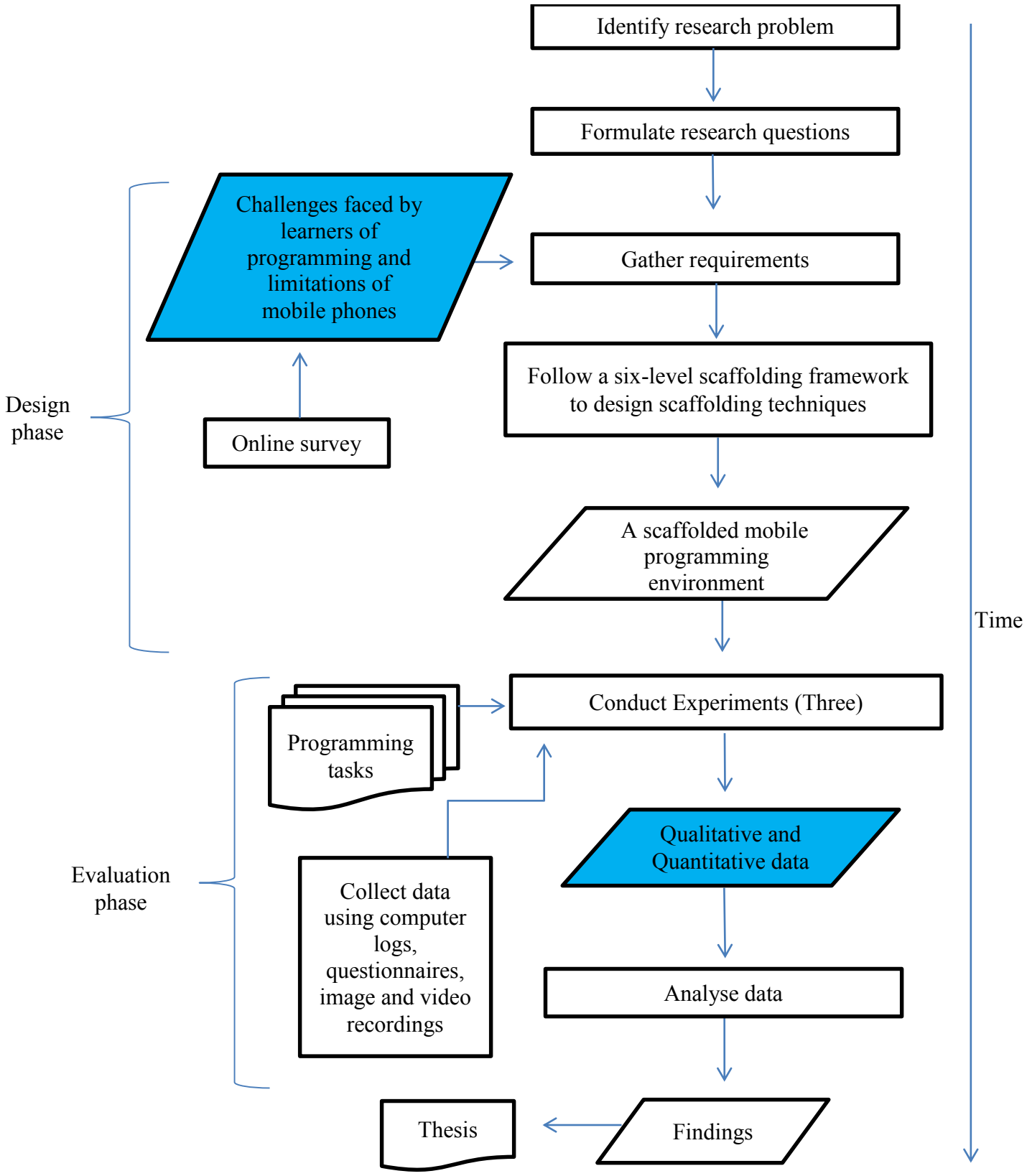


Figure 1.3: Flowchart showing mixed methods research design and research approach followed in the study

and third experiments, control and experimental groups participated in the study, where the control group used the non-scaffolded environment.

Evaluation was conducted while learners used the mobile programming environments to construct programming tasks. Conducting evaluation while considering data about learners' interaction is encouraged in educational evaluation models such as the micro-meso-macro framework (Vavoula & Sharples 2009) and the CIAO model (Jones et al. 1999). Following this recommendation, the evaluation criteria derived from the research questions were used to analyze the data. For example, to identify which scaffolding techniques were used to construct the programs, first, task success was measured by analyzing if a programming task was successfully completed or not. Thereafter, analysis was conducted on which scaffolding techniques were used to construct the complete and incomplete tasks. This evaluation process led to the research findings.

1.5 Research Contributions

In addressing the research questions, it was expected that this research would make the following five contributions:

1. Application of constructivist principles in designing scaffolding techniques on mobile programming environments.
2. A theory-driven process of designing scaffolding techniques for a mobile programming environment.
3. A proof-of-concept prototype with which novice learners can construct Java programs while supported by scaffolding techniques.
4. Empirical evidence about which scaffolding techniques could support Java programming on a mobile phone.
5. Empirical evidence about the effect on learners of using scaffolding techniques to support Java programming on a mobile phone.

It was anticipated that making the above contributions would generate interest among educators and researchers working on designing mobile-based tools that support learning, especially in subjects such as programming that require a hands-on approach.

In addition, the study would contribute towards tackling the challenges in learning programming among novice learners, especially in resource-constrained environments where learners own mobile phones but could have limited access to PCs or laptops outside the classroom. In using ICT (scaffolding techniques on a mobile phone) to foster development (improving skills by learning), the results of this study would be relevant to the field of ICT for Development. Further, this study showed a theoretic

and methodological process for designing a programming environment on a mobile phone; such a methodological process was emphasized as important in conducting Mobile for Development research (Svensson & Wamala 2012).

1.6 Thesis Outline

Chapter 2: Constructivism and Programming

In this chapter, the constructivism theory and its use in programming is discussed. Other learning theories are presented, leading to a discussion on the choice of constructivism as a grounding theory for this research. Finally, scaffolding as a principle of constructivism is discussed.

Chapter 3: Related Work

Previous work that relates to the use of scaffolding in programming is discussed in this chapter. In order to guide the structure of the chapter, discussion is divided into four parts: difficulties faced by novice learners of programming; scaffolding programming on PCs; using mobile phones for learning and the limitations of mobile phones; and learning programming using mobile phones. The chapter concludes with a summary of gaps and opportunities that have been identified in related work.

Chapter 4: Design and Implementation

The design of a prototype that offers scaffolding techniques to construct Java programs on a mobile phone is presented in this chapter. A six-level scaffolding framework that culminates in the choice of scaffolding techniques guides the design process. How the scaffolding techniques were implemented on a mobile phone is discussed, followed by a summary of the scaffolding techniques. Thereafter, the system overview is presented followed by an example of how a simple program can be created on the scaffolded environment. The chapter concludes by describing a non-scaffolded environment that was used by learners in a control group.

Chapter 5: Evaluation

This chapter discusses how evaluation was conducted in order to address the two research questions. The chapter describes the participants who took part in the study, and the data collection methods and materials used. Further, the chapter discusses the number of experiments that were conducted and how they were conducted. Thereafter, the evaluation criteria derived to address the research questions and the related hypotheses are presented. The chapter concludes with a summary of the criteria used to address the two research questions.

Chapter 6: Results

In this chapter, the results and analyses of the collected quantitative and qualitative data as per the evaluation metrics used to address the research questions are presented and discussed. The chapter starts with a discussion of the participants who took part in the study and a review of how they

participated in the experiments. Thereafter, results and related discussions are presented for each of the three experiments. The chapter concludes with a summary of the research findings.

Chapter 7: Conclusion

This chapter begins by restating the research problem and the research questions. A synthesis follows of how the empirical findings addressed the research questions. Thereafter, the chapter discusses the implications of the study. Finally, the chapter discusses the limitations of the study and ideas for future research.

Chapter 2 Constructivism and Programming

The previous chapter introduced the purpose and motivation of the research and briefly showed that the constructivist theory supports learning by doing and scaffolding. This chapter describes the constructivist theory and its application to programming. Thereafter, scaffolding is described as a principle of constructivism.

2.1 Choice of Constructivist Theory

The need to choose a learning theory was influenced by two factors: to select a learning theory that supports the nature of programming as a practical course; and to select a learning theory that can underlie the concept of supporting learners. There are several learning theories such as behaviorism, cognitivism, and constructivism. Table 2.1 shows the differences among these three learning theories as outlined by Ertmer & Newby (2008) using the five definitive questions described by Schunk and Dale (2011). Further, using the example of a child learning how to walk, the last row of Table 2.1 illustrates how each of the theories could be applied to this example.

As illustrated in the table, behaviorism and cognitivism focus on response to stimuli and internal mental processes, respectively, while constructivism focuses on interaction between the learners and the environment. In this research, new learners of programming interact with a programing environment in order to learn a programming skill. In addition, since the aim was to support learners outside the classroom, there was need for a learning theory that emphasizes on individual learning since the learners are assumed to be working on their own. This made constructivism the appropriate choice of a theoretical framework.

However, one criticism of constructivism is that it is relativist, where anyone's constructions are as good as anyone else's and where we are unable to judge the value or truth of constructions with any degree of certainty (Cunningham & Duffy 1996). While this is a genuine criticism, its negativity is lessened in the context of programming by using correct outcomes of programs as the criteria for validity. A second concern is that the individualistic nature of constructivism leads to an inability to communicate (Cunningham & Duffy 1996). That is, learners are unable to talk to one another because learning occurs through personal experience. Since the aim of this research was to provide support outside the classroom alongside other modes of learning, there was room for learners to communicate. A third criticism of constructivism is that researchers attempt to implement the theory by promoting active knowledge construction while giving minimal guidance (Kirschner et al. 2006). Such minimal guidance is only provided if the learner needs it, hence the learner is required to construct most of the new knowledge on their own. It seems that such a constructivist approach reflects how programming

Table 2.1: Differences between behaviorism, cognitivism, and constructivism for learning

	Behaviorism (Response to stimuli)	**Cognitivism (Mental processing)**	**Constructivism (creating meaning from ones experiences)**
How does learning occur?	When a proper response is demonstrated to specific stimuli.	Learning focuses on what learners know.	Emphasis is on creating meaning.
What factors influence learning?	External stimuli.	Environmental conditioning.	Learner and environmental factors.
What is the role of memory?	Emphasis is placed more on habit rather than reliance on memory.	Learning results when information is stored in the memory in an organized manner.	Understanding is developed through continual use.
How does prior learning affect new learning?	Through generalization.	Retrieving knowledge from memory.	Engaging the learner in actual use of tools in real world situations.
What types of learning are best explained by the theory?	Recalling facts, generalizations, associations, and chaining (automatically performing a specified procedure).	Reasoning, problem solving, information retrieval.	Transitional learning that equips new learners with skills they can use to become advanced learners.
Theory applied to example of a child learning to walk.	The stimuli could be the adult holding out his hands in front of the child and encouraging the child to reach out.	A child learns how to walk depending on the stage of their cognitive development. (Piaget 1964)	An adult holding the child's hands learns how to walk by using the adult's hands as support as they take one step after another.

is taught, where the emphasis is that people should learn programming by constructing programs from the basic information of the language, and they should do it in the same way that experts do (Guzdial 2015). Such an approach capitalizes on the learners' working-memory (and not the long-term memory), which retains information only temporarily, if at all (Kirschner et al. 2006). This study addressed this criticism of constructivism in two ways. First, the aim of the study was to provide a scaffolded environment alongside a classroom learning experience. This way, the learners could still receive active instructions in the classroom, which they could then apply in creating programs using the scaffolding techniques. Secondly, the scaffolding techniques were designed to provide strongly guided learning while learners construct programs. Such an approach was recommended as one that enables deeper learning than one with minimal guidance (Moreno 2004). Further, some forms of strongly guided learning approaches are worked examples or process worksheets (Kirschner et al. 2006). In the design of the scaffolding techniques in this study, such approaches were considered.

2.2 Constructivist Theory

Constructivism stems from the field of cognitive science, particularly the work of Jean Piaget and the socio-historical work of Lev Vygotsky (Fosnot & Randall 1996). In addition, Seymour Papert developed a theory of learning based on Piaget's constructivism (Ackermann 2001). A description follows on how Piaget, Papert, and Vygotsky described constructivism.

Piaget's constructivism suggests that knowledge expands from within according to complex laws of self-organization (Ackermann 2001). As such, children's perceptions of the world are determined by innate processes and not what an adult says is wrong or right. However, this does not mean that children's perceptions do not change. Indeed, they are continually evolving as they interact with their environment. But, for a child to abandon their current view, they must go through experiences and actions in the world (Ackermann 2001). Piaget's view describes two implications on education: (i) learning is not a direct process that is influenced by external factors, but one that happens innately from within; and (ii) a learner grows from their innate knowledge by going through an experience. It is this focus on internal cognition by Piaget that Papert diverts from.

Papert's description of constructivism focuses on learning through making rather than overall cognitive potentials (Ackermann 2001). Papert's view stresses that learning happens through context and knowledge is acquired when a learner expresses himself, which in turn makes that idea tangible and therefore can be shared. Stressing the importance of a learner expressing themselves to an external environment is not new; Vygotsky stressed on social interaction to foster learning.

Vygotsky's theory focuses on socially elaborated learning where he emphasized that it is in the course of interaction between children and adults that young learners identify effective means for

remembering (Vygotsky 1978). He further argued that the lack of recognition among educators of the ways in which an experienced learner can share his knowledge with a less advantaged learner, limits the intellectual development of many learners (Vygotsky 1978). Therefore, people gain by receiving guidance from others. This is a notion that had been discussed by Bruner.

In the book *Toward a Theory of Instruction,* Jerome Bruner talks about how instruction is achieved through showing and not telling (Bruner 1966). In the final chapter of this book, he tells of a scenario observed between children and adults of a hunter-gatherer community where there were very few instances of 'telling' or teaching as we know it, but children imitated what they saw adults do. This book could be said to have begun the first illustrations of application of constructivism in education, without explicitly calling it so.

Despite the different definitions, they seem to all share three key characteristics that form the core of constructivism:

(i) knowledge is gained when a learner goes through an experience that enables them to learn;

(ii) learners are active builders of their own knowledge through expression and interaction with other people or other things in their environment; and

(iii) there is a relation between existing knowledge and any new knowledge that is acquired by the learner.

Constructivism has been applied to many domains, including education and educational software. However, it is noted that constructivism is a theory about learning, not a description of teaching (Fosnot & Randall 1996). This distinction stems from the illustration that knowledge cannot be merely copied from a teacher to a learner. Instead, knowledge is acquired when learners are given an opportunity for meaningful experience based on the information given by the teacher, through which they can ask questions, interact with the information and create their own mental models.

In order to practically apply constructivism to the design of applications, some researchers derived a set of constructivist principles (Winterbottom & Blake 2004; Winterbottom 2010). These principles are:

i. *Atomic simplicity*, where new pieces of information are kept as simple as possible and the complexity of knowledge can be built through links between the simpler parts, and there is provision for incremental building of knowledge.

ii. *Multiplicity*, which encourages multiple perspectives on concepts and methods of approaching a problem.

iii. *Active exploration*, which supports the active learning nature of constructivism. Part of exploration is making mistakes and learning from them. Therefore, errors can be seen

as a mechanism for users to gain insight, and this means that they should be easily identified.

iv. *Reflection,* which can enable people to form viable theories about their knowledge and how it fits together. The process of constructing knowledge requires acting with reflection so as to build effective connections between bits of knowledge.

v. *User control,* which implies that active construction is the idea of personal control where people gain power over their learning processes by actively constructing their own knowledge. While learners have control over their learning process, they could be provided with support that facilitates this learning that with time, adjusts according to their needs. Such support is known as scaffolding.

vi. *Scaffolding,* which describes guidance provided in the form of artefacts, advice and tutorials, which allow learners to perform tasks that would normally be beyond their ability, but which fall away when learners have constructed the knowledge and skill to accomplish the task alone.

These six principles could be applied to programming. Programming being a complex subject, the aim must be to simplify as much as possible the interface that is presented to a new learner. In so doing, a programming environment can provide multiple views of a program and then support the learner to connect them into a single unit. Further, a programming environment could provide feedback when errors are encountered. In addition, programming learners need to be supported to think about the programs they are creating. This can be achieved by providing multiple representation and feedback mechanisms. Lastly, while learners construct programs, they could be provided with support that adjusts over time.

The definitions of constructivism, and the aforementioned characteristics and constructivist principles, illustrate the suitability of the constructivist theory for application to learning of programming. If knowledge is acquired through doing, then it is possible to conclude that if learners are adequately supported while constructing programs, they will be able to learn programming. The next section discusses the application of constructivism in programming.

2.3 Constructivism in Programming

One of the widely cited papers that examines the application of constructivism in Computer Science Education (CSE) indicates that, at the time it was written, the constructivist theory had been widely influential in science and mathematics education but not in CSE (Ben-Ari 1998). This paper asserted that the application of constructivism to CSE must take into account two characteristics that do not appear in natural sciences: (i) a novice CSE student does not come to the course with a mental model

of how to work with a computer or how a computer works; and (ii) a computer forms an accessible source of correct answers with its own feedback system.

Therefore, since a new CSE learner does not come to the course with a preconceived model, a viable model must be constructed in order to guide the learner in acquiring new knowledge. Further, application of the constructivist theory to programming suggests that some knowledge must first be shared with the learner (perhaps through teaching) in order for the learner to use that knowledge to create their own experience (perhaps through trying out exercises). This research aims at providing support outside the classroom, alongside a learner's classroom experience, thereby meeting this characteristic of constructivism in CSE.

In the last decade, there has been significant interest in studying the application of constructivism in programming. There have been different ways that constructivism has been applied to programming. Some of these are:

(i) programming as a collaborative effort between learners, for example, where learners are engaged in a collaborative code development environment using a smartphone interface, which allows for individualized feedback (Pears & Rogalli 2011);

(ii) programming where another resource is needed, for example, where a learner is required to first open a textbook and then use an environment that provides guides and prompts on how to complete examples from the textbook (Esper et al. 2012); and

(iii) teaching programming based on questions from learners, for example, where learners ask questions and post this to a Blackboard portal and then the next lesson is taught based on the questions that the learners thought were most relevant (Boyer et al. 2008).

What is similar across these different approaches is the focus on learners working on the programs themselves and the availability of some kind of support that guides the learner. In addition, and to emphasize Ben-Ari's assertion, such support should enable a new learner to create the correct mental model. Such support is known as scaffolding.

2.4 Scaffolding

Vygotsky illustrated the concept of scaffolding when he defined the Zone of Proximal Development (ZPD) in relation to support that a child receives from an adult (Vygotsky 1978). However, Vygotsky did not explicitly call this support 'scaffolding' but he implied it from his description of two types of school-going children.

Suppose there are two children, both of them 10-year old chronologically and 8-years old in terms of mental development. These two children can be said to be of the same age mentally because they can independently deal with tasks up to the degree of difficulty that has been standardized for an

8-year old. Suppose that these two children are thereafter shown various ways of dealing with a particular problem by a tutor. For example, one way might be to ask a child to repeat some words after the tutor, another might be to initiate a solution and then ask the child to finish it. Under these circumstances, it may turn out that the first child can deal with problems up to a 12-year old's level and the second child up to a 9-year old's level. So at this point it can be concluded that these children do not have the same mental capacity (Vygotsky 1978).

Borrowing from this analogy of two children, suppose we have two novice learners of programming, both of whom have no prior experience in programming and are both taking their first class of programming. At this point, these two learners can be said to be at the same level. Suppose that these two learners are thereafter provided with different ways to tackle programming exercises. For example, they could be provided with a programming environment that provides coaching, such as in Emile (Guzdial 1994), or one that provides small incremental steps to complete a program, such as in Test My Code (Vihavainen et al. 2013). While working in these environments, it then turns out that the first learner is able to progress quickly to work on more advanced programs, and the second learner takes more time on simpler programs. The difference between these two learners is the ZPD.

Therefore, ZPD can be defined as '*the distance between the actual development as determined by independent problem solving and the level of potential development as determined through problem solving while under adult guidance or in collaboration with more capable peers.'* (Vygotsky 1978). ZPD has been applied to an introduction to programming course by combining ZPD and a comfort zone to result in the *comfort zone of proximal development* (CZPD) as shown in Figure 2.1 (Anderson & Gegg-Harrison 2013). In this particular example, the zone of proximal development was the concepts taught in an introductory object oriented programming course. The comfort zone was an extra-credit course that introduced the learners to the development of iPhone applications. Therefore, CZPD combined the provided support by the teachers and resources in the course (ZPD) with an approach that was deemed interesting to the learners (comfort zone).

Figure 2.1: Comfort zone of proximal development (Anderson & Gegg-Harrison 2013)

In both ZPD and CZPD, it seems that learners are provided with an additional entity that aims to support their learning. Further, both ZPD and CZPD illustrate that a learner has greater potential that can be arrived at with extra support. Thus, scaffolding refers to support provided so that the learner can engage in activities that would otherwise be beyond their abilities or their unassisted effort (Wood et al. 1976; Jackson et al. 1998). In addition, scaffolding involves providing learners with supportive aids in the form of tools, strategies, and guides within the parameters of their ZPDs, to assist them in progressing to their next, potential level of development (Saye & Brush 2001).

Bruner jointly wrote a paper with Wood (Wood et al. 1976) where they illustrated the concept of scaffolding in an experiment where children were required to arrange blocks into a pyramid with the tutor's assistance; as the child became more proficient, the tutor provided less assistance. Further, while the child assembled the blocks, the tutor would provide assistance depending on how the child progressed. For example, if the child had tried to assemble pieces for himself but had overlooked a feature, then the tutor would verbally draw his attention to the fact that the construction was not complete (Wood et al. 1976). In addition, the tutor finally left the child to his own devices. It is only if the tutor noticed that the child was struggling, would the tutor intervene to offer guidance. This illustrates a critical component of scaffolding known as fading. However, even after the scaffolding has faded, support should still be available to the learner should they still need it.

From these descriptions, three characteristics of scaffolding emerge:

(i) Scaffolding should be provided while a learner is performing a task.

(ii) Scaffolding should be suited to the different needs of individual learners.

(iii) Scaffolding needs to fade, but with possibility of the learner enabling it.

In addition, scaffolding addresses the proposed principles of constructivism. Scaffolding provides learners with *control* over the tasks that they perform by supporting them to actually perform a task. It also enables *exploration* by providing support such as feedback from errors. By offering different kinds of scaffolding, *multiplicity* is offered. Finally, as scaffolding supports a learner to complete a task, it enables *reflection* of the process.

2.5 Chapter Summary

This chapter has described the constructivist theory, its origins and application to learning of programming. The choice of constructivism as a theoretical framework was justified by comparing it to behaviorism and cognitivism theories. Further, criticisms of constructivism were addressed in four ways: (i) the aim of the study was to provide additional support to construction of programs, alongside active class instruction; (ii) the designed scaffolding techniques aimed to provide guided support to creating programs; (iii) the correct output of programs could be used as a criterion for validity; and

(iii) since learners were to use the scaffolded environment alongside other resources and classroom experience, there was room for learners to communicate. The different definitions of constructivism from Piaget, Papert, Vygotsky and Bruner led to three common characteristics of constructivism: (i) knowledge is gained through experience; (ii) learners are active builders of knowledge; and (iii) existing knowledge is used to create new knowledge. Thereafter, the discussion of scaffolding from the works of Vygotsky, Bruner and other researchers showed that: support to learners needs to enable active constructions of programs; this support should fit different learners' needs and fade over time; and this support should provide atomic simplicity, support different representations, enable user control, and support reflection.

The following chapter provides a synthesis and analysis of previous work that relates to this research.

Chapter 3 Related Work

Scaffolding has been used to support learning of various subjects such as physics (Guzdial 1994), bird watching (Yuh-Shyan Chen et al. 2002), chemistry (Girault & D'Ham 2013), and programming (Vihavainen et al. 2013). The aim of this research was to contribute towards tackling learning difficulties in programming. Therefore, this chapter focuses on programming and begins by reviewing previous work on difficulties faced by novice learners in the subject.

Support to learners of programming can be provided by humans, such as by mentors (D'Souza et al. 2008), or by software. This chapter focuses the discussion on related work that proposed software-based support. Significant work has been done on scaffolding learners while they use PCs to program in various programming languages. Therefore, this chapter reviews how scaffolding has been used to support learning of programming on PCs. Thereafter, this chapter reviews the use of mobile phones as learning environments, especially in resource-constrained environments. The discussion reviews the limitations of mobile phones and design recommendations by several researchers. This is followed by a discussion on related works that use mobile phones as programming environments. This chapter concludes with a summary of the gaps and opportunities identified in the related work.

3.1 Difficulties Faced by Novice Learners of Programming

It takes ten years for a novice programmer to become an expert (Winslow 1996). If true, this claim implies that novice learners require a significant amount of effort to learn programming. In fact, a study conducted at the University of Cape Town shows that programming elicits feelings of fear among learners (Rogerson & Scott 2010). This study conducted several interviews where participants described how the word "programming" evoked feelings of apprehension or discomfort. Further, the study stated that one of the causes of this feeling could be that many learners are first exposed to programming at the beginning of their tertiary level studies, or that those with prior experience of programming may be confronted with a very different level of expectation. The study established that the fear factor has implications such as a low level of comfort and self-confidence and increased levels of anxiety that inhibit the appreciation of programming. Indeed, when learners struggle in programming, it affects most facets of their study, for example: their progress through their study program, their study habits, their confidence, and their time management (D'Souza et al. 2008). These studies emphasize the need to provide support to novice learners.

Several factors contribute towards the difficulties in learning programming (Jenkins 2002): programming requires multiple skills; programming involves multiple processes; the language used

for teaching; the novelty of programming; lack of interest by the learner; reputation of programming as difficult; and the pace of teaching programming.

Programming requires multiple skills and processes. Apart from learning the novel syntax of the programming language itself, learners have to learn how to create algorithms, how to write code using proper style, and how to identify bugs in their programs. Novice learners struggle with these skills and most times with problem solving. Yet, there is a positive correlation between a learner's problem solving ability and programming performance (Pillay & Jugoo 2005). Further, studies concluded that novice programmers may know the syntax and semantics of individual statements but they do not know how to combine these features into valid programs (Winslow 1996). Therefore, there is need to support novice learners to build on fundamental skills such as how to combine different parts of a program into a working program.

There has been debate on which language should be used to teach novice learners of programming. Some have argued that Python is a suitable language for novice learners (Grandell et al. 2006), others have recommended the use of Scratch to introduce programming (Wolz et al. 2009), while others have experimented with more than one language at the same time (DeClue et al. 2012). Yet, studies have shown that the pass rate in introductory programming is largely unaffected by the programming language taught in the course (Bennedsen & Caspersen 2007; Watson & Li 2014). This could be because the purpose of an introductory programming course is to teach the students to program; the intention is not to, for example, "teach them Java" (Jenkins 2002).

Java emerged as the most widely used first programming language beyond 2006, whereas C++ remained the runner-up throughout this time, with Python showing an increase in use from 2006 to 2011 (Farooq et al. 2014). A recent survey indicates Python as the leading language in use in introductory programming courses in parts of the world such as in the US (Shein 2015; Guo 2014). The survey indicated that 27 of the top 39 universities in the US teach Python in introductory programming courses. This trend was rightly predicted by Guzdial (2011). However, some universities in developing countries do not yet offer Python at all in the introductory courses. For example, two of the four universities that participated in this research (both from Kenya) do not currently offer any programming course using Python. One of the recommendations from a research conducted in Tanzania was that perhaps there should be a move from Java to a simpler language such as Python for introductory programming courses (Apiola & Tedre 2012). Yet, Java is still widely used to teach introductory programming. Therefore, there is a need to still contribute towards tackling learning difficulties in courses taught using object oriented languages. The research in this thesis focuses on Java.

Programming has been considered as a boring subject (Jenkins 2002; Ibrahim et al. 2010), with learners having negative perceptions about it because of the difficulty in the subject and from external feedback from others (Rogerson & Scott 2010). Unfortunately, such views are shared among learners and, as a result, novice learners expect to struggle in the subject. To aggravate this perception, programming at university-level is taught within a fixed set of time, following a set curriculum. This means that the learner is unable to learn at his or her own pace and is required to pass programming assessments at set periods. However, since it is more difficult to change the pace of an existing curriculum than it is to offer additional support to novice learners, this research aims at providing additional support.

There have been a significant number of tools created to support novice learners of programming. However, a recent study (Watson & Li 2014), which extended the work by Bennedsen and Caspersen (2007), shows that despite the increase in the number of tools available to support learning of programming, the average pass rates have not improved over the years. This asserts the need to continually experiment with new and existing pedagogical approaches in order to contribute towards tackling the learning difficulties in programming.

The studies by Bennedsen and Caspersen (2007) and Watson and Li (2014) consisted of data with at most 2% representation from Africa, specifically from South Africa. This is a minimal representation of the African context. Perhaps the reason for this could be that there is little research conducted on novice learners' programming experiences in developing countries. Therefore, there is need for further research to understand the specific issues and provide solutions to learners in a developing country's context. Indeed, applying western pedagogies to developing countries' context may prove counterproductive and there is a call for contextualized curricula to fit resource-constrained environments (Apiola et al. 2011). Even though the research in this thesis does not focus on contextualizing an existing curriculum, it contributes towards filling this gap since it was conducted within a developing country's context, specifically in Kenya and South Africa.

Irrespective of the causes of difficulties in programming or the context in terms of country and availability of resources, related studies stress the importance of learning programing by doing. The more practical and concrete the learning situations and materials are, the more learning takes place. Learning by doing should be a part of the studies all the time (Lahtinen et al. 2005). Further, studies indicate that learners of programming consider learning by doing as motivating and rewarding (Vihavainen et al. 2011). This is in line with the constructivist theory, which is the underlying theory of this research.

Learning programing by doing requires access to resources such as PCs and laptops. However, most learners at institutions in parts of Africa are in resource-constrained environments where they

have limited access to such resources, especially while they are outside the classroom. Even within the institutions, the available computer laboratories are sometimes used for lectures and tutorials, as opposed to individual practice sessions. Further, some schools have a limited number of desktop computers that could be shared among learners. For example, even in a relatively well-resourced developing country like South Africa, it is not uncommon for a school of 1,000 learners to have only one computer room with 30 PCs (Traxler & Vosloo 2014). The lack of adequate resources is a concern because research conducted at a university in Tanzania proves that difficulties in programming are aggravated in resource-constrained environments where learners do not have easy access to computers (Apiola et al. 2011). Similarly, research conducted at an institution in Ethiopia found that learning difficulties among novice learners were aggravated by lack of practice; instead, learners solved programs on paper and rarely used the computer laboratory (Bati et al. 2014). Indeed, poor infrastructure and facilities is one of the major challenges faced by higher education in Africa (Yizengaw 2008). The research in this thesis was motivated by the resource constraints in a developing country's context.

In order to tackle the learning difficulties in programming in resource-constrained environments, some pedagogical approaches have been proposed: redesigning the ACM/IEEE IT curriculum to fit within a Tanzanian context (Apiola & Tedre 2011); and a blended learning approach that combined face-to-face and technology-supported instructions to tackle the problem of large programming classes in an Ethiopian context (Bati et al. 2014). These approaches differ with the one in this study since this study focused only on provision of software-related support, and not change of curriculum or inclusion of a face-to-face approach. To begin the discussion on software-related support, the next section reviews scaffolding programming on PCs.

3.2 Scaffolding Programming on PCs

There have been a significant number of studies that tap into the computational powers of PCs and the Web in order to support novice learners of programming. To focus the discussion, this section reviews these works in four categories: (i) new programming languages; (ii) Web-based applications; (iii) stand-alone applications; and (iv) applications based on teacher-learner architectures.

3.2.1 New programming languages

The need for a new language to teach introduction to programming is not a new concept. For instance, Turing was designed to overcome some of the weaknesses of Pascal in order to enable ease of learning as one of its goals (Holt & Cordy 1988). Indeed, a 'Hello world' program written in Turing is merely one line long, as opposed to at least seven lines long in Java. However, a criticism of Turing is that it

was not a useful language in the real world (Chatley 2001). To address this criticism, Kenya was designed as a language that is simple enough to use to create programs, but which the development environment translates into Java code (Chatley 2001). In addition, Kenya was designed to reduce some of the syntax that was found in Java. For example, programs written in Kenya did not require the use of the 'main' line declaration that is required in Java. This approach was later shared by designers of a new programming language, Grace, who asserted that there is no good reason to subject novices to 'public static void main(String [ ] args)' early in a first course, or to have them obsess over which lines should end with semicolons (Black et al. 2013).

Consequently, Grace was designed to provide a language that represents the key concepts underlying object-oriented programming in a way that can be easily explained. Grace involved the design of a programming environment and language specifically to support novices. On the contrary, further work with Kenya integrated it with Eclipse in order to provide a trimmed down workbench for a new learner (Chatley & Timbul 2005). Indeed, several Eclipse plug-ins have been designed to overcome the overhead of programming within a complicated IDE, especially for novice learners (Mueller & Hosking 2003; Storey et al. 2003; Reis & Cartwright 2004). These examples show that most existing desktop IDEs are complex for a novice learner.

3.2.2 Stand-alone applications

Earlier work on scaffolding programming on PCs provided environments where the process of creating a program could be done on a single interface. For example, the Goal-Plan-Code editor (GPCEditor) enabled construction of Pascal programs on a single interface in three steps: creating a goal, planning, and composition (Guzdial et al. 1998). Although the evaluation of GPCeditor showed that it effectively supported the construction of programs on a PC, the use of such a single interface for all the processes may not be suitable for the small screens of mobile phones.

However, GPCEditor utilised some techniques that could be explored for use on a mobile programming environment. For instance, in the planning stage of the GPCEditor the menu items associated with the plans were disabled until a goal was created (Guzdial et al. 1998). Further, the editor constrained the order of how the plans could be assembled. Such restrictions in code construction could be useful on a mobile programming environment because different sections of a program could be decomposed and presented one at a time. Decomposition of tasks was suggested as a suitable scaffolding technique for handheld devices (Luchini et al. 2004). In addition, on first use of the GPCeditor, there was provision of some basic Pascal statements that learners could reuse. This was also implemented in the Code Restructuring Tool (CORT) (Garner 2004), which allowed part complete solutions to programming problems to be displayed in one window and possible lines of code to be

inserted into the solution within another window. Such a technique could be useful on a mobile programming interface because provision of some default statements makes small interfaces usable by limiting user input (Luchini et al. 2003).

Over the years, visual environments such as Alice (Cooper et al. 2000), JELIOT (Ben-Bassat Levy et al. 2003), and BlueJ (Kölling et al. 2010) have been developed to enable novices to learn programming within 3D environments. For example, Alice provides a drag-and-drop development environment to prevent students from making syntax errors. It also enables the writing of simple scripts in which its users can control 3D object appearance and behavior. The benefit of using Alice is that it allows students to be involved and at the same time have the ability to develop an intuitive understanding of basic concepts in a visual feedback environment (Cooper et al. 2000; Dann et al. 2011; Dann et al. 2001). However, environments such as Alice are highly graphical and take advantage of the computing power of PCs. Given the limitations of mobile phones, such a highly animated environment may not be suitable for a mobile programing environment. Further, it was observed that learners who could program within the Alice environment had difficulties programming when presented with a textual programming environment (Powers et al. 2007). In addition, it was observed that learners became so engrossed in manipulating the 3D objects that they would overlook the more important goal of learning basic programming concepts (Powers et al. 2007). Therefore, perhaps programming environments on mobile phones could use a combination of less graphical visual objects and text input.

3.2.3 Teacher-learner architecture

Some recent studies have focused on teacher-learner environments where an instructor can track the learners' solution to a programming problem. Test My Code (TMC) (Vihavainen et al. 2013), the programming exercise teaching assistant (PETCHA) (Queirós & Leal 2012), and Java Programming Laboratory (JPL) (Pullan et al. 2013), are such environments that were used alongside existing IDEs to support learners to program on PCs.

Test My Code (TMC) is a NetBeans plugin that is part of a client-server architecture, which enables learners to submit code to a remote server, from which instructors can perform code reviews (Vihavainen et al. 2013). The NetBeans plugin retrieves and updates programming exercises from an assessment server, displays built-in scaffolding messages during the coding process, submits exercises to the assessment server, allows giving and receiving direct feedback during the exercise, and gathers data from learners' programming courses. TMC offers scaffolding in the form of pre-designed exercises that contain code snippets, a set of tests provided to enable incremental completion of the program, and the expected output of the program. In TMC, fading of scaffolding was provided using

open exercises that do not enforce any specific program structure or approach. For example, before fading is implemented an exercise could contain sample input/output and code snippets. When fading is implemented the exercises does not contain code snippets, but could contain only a program description and sample/input output. This approach could be useful in fading the scaffolding on a mobile programming environment.

PETCHA is a programming exercise teaching assistant that enabled exercise authoring by a teacher and exercise solving by a learner (Queirós & Leal 2012). PETCHA works with an IDE where the learner reads the exercise description on PETCHA and solves it on an IDE. PETCHA is part of a learning management system that includes an automatic evaluator of the learners' code. After testing the code, the learner submits the solutions to an evaluation engine that checks the solutions against the teacher's test cases. PETCHA was evaluated by comparing its use and that of a traditional classroom, which had no software support. The results indicated that users of PETCHA were able to attempt and solve a significantly higher number of tasks. Similar to TMC, PETCHA was used alongside an IDE to create the exercises. However, using a PC based IDE alongside a supporting tool may not be a suitable approach for construction of programs on a mobile programming environment that could be used by learners outside the classroom, away from PCs or laptops.

The Java Programming Laboratory (JPL) is a cloud-based integrated environment that contains video tutorials, a website that contains programming problems, and is integrated with an IDE based on the Dr Java IDE (Pullan et al. 2013). Like TMC and PETCHA, the integrated use of an IDE may not be suitable away from PCs or laptops. An integral part of JPL is the use of short video tutorials explaining programming concepts and problem solving techniques. Further, JPL offers scaffolding by providing different ways of completing programs depending on the level of the learner. These include multiple choice questions and 'fill-in-the-blank' exercises that provide templates for learners to complete. The learner then uses the JPL automated testing to check for correct logic. The use of multiple choice questions and 'fill-in-the-black' exercises differs with the aim of this study, which is enabling learners to construct programs as opposed to completing exercises.

3.2.4 Web-based applications

There has been a trend to move IDEs from the desktop to the cloud. The Java Wiki Integrated Development Environment (JavaWIDE) is one of the new online IDEs (Jenkins et al. 2010; Jenkins et al. 2012). However, environments like JavaWIDE have been criticized as being similar to desktop IDEs with a plethora of menus, toolbar buttons, tabs, and docked views for project management and program input/output (Edwards et al. 2014). To address such a criticism, Pythy was designed to provide a cleaner web-based environment with a complete ecosystem for learners of Python (Edwards et al.

2014). A different application for Python is an interactive textbook that incorporates a number of active components such as video, code editing and execution, and code visualization as a way to enhance the typical static electronic book format (Miller & Ranum 2012). Whereas such integrated environments could be suitable for larger interfaces such as PCs and perhaps tablets, they may not be suitable on a mobile phone. Indeed, although excellent in desktop environments, the usability of such systems is lacking on mobile touch devices where the screen space is limited (Ihantola et al. 2013).

Ideone[3] is a free online compiler and debugging tool that allows online creation, compilation and execution of source code in more than 60 programming languages. In addition, Ideone offers a sphere engine that enables remote execution of code. For this reason, it can be used alongside the relevant APIs to implement programming environments on a mobile phone. This way, a program can be created on a mobile programming environment and then sent to ideone for compiling, with the output received on the mobile phone. For example, Ideone has been used by IDEdroid[4], a mobile programming environment. Therefore, ideone was selected as the compiler to use in this research.

Lastly, Codecademy[5] and Khan Academy[6] are online platforms where learners can write programs regardless of location. For example, Khan Academy enables creation of Python and JavaScript programs. When creating JavaScript programs in Khan Academy, each change to the code is executed immediately and the output is seen on the right hand side of the interface. Khan academy offers scaffolding in the form of hints and error checks. Whereas these environments provide useful tools for programming on the Web, their interfaces were not designed for mobile programming environments.

3.3 Using Mobile Phones for Learning

With increased mobile phone penetration, it is hardly surprising that the use of mobile phones for learning has attracted considerable attention in recent years. In Africa, factors such as the general lack of infrastructure, sporadic supply of electricity, lack of skilled technical support, the high cost of installing and maintaining a network and the easy to use interface of mobile phones have contributed to the high rate of adoption of mobile technology (Traxler & Leach 2006).

Despite the penetration of mobile devices in most parts of the world, their use in learning is underexplored in developing countries. For example, most of the eLearning technologies implemented in higher education in East Africa are based on desktop computers (Mtebe & Raisamo 2014). Yet,

[3] https://ideone.com/sphere-engine

[4] http://goo.gl/U53s4o

[5] http://www.codecademy.com/

[6] https://www.khanacademy.org/

studies conducted in developing countries show that learners in higher education believe that learning using mobiles is useful, and could enable them to accomplish their learning activities faster and more efficiently (Mtebe & Raisamo 2014; Ibrahim et al. 2010; Kafyulilo 2012). This shows that there is a gap in providing learning environments on mobile phones in developing regions.

Despite the claim that there is little implementation of learning using mobiles, there are some related studies conducted within developing countries. For example, an SMS-based mobile learning application was tested at University of Cape Town; it enabled learners to ask questions and get responses from the teacher and from each other (Ng'ambi 2005). Similarly, a study in Tanzania implemented a mobile Web-based system to facilitate the dissemination of course information including reading materials and assessments (Ajayi et al. 2011). While such SMS and text-based approaches enabled instructors to provide individualized effort and information that could reach many learners at the same time, they may not be suitable in a course such as programming where the learner needs to write programs as opposed to sending queries or receiving text-based information.

Dr Math is a mobile tutoring service that provides access to credible personal on-demand tutoring in Mathematics (Butgereit 2012). The service is accessed through the MXit mobile social networking service. Dr Math links South African primary and secondary school pupils to university students for help with their mathematics homework. Feedback support is provided using chat messages on MXit where a learner sends a mathematics question and the tutor responds through chat and guides the learner towards an answer. While the approach used by Dr Math has been successful in supporting learners of mathematics (Butgereit 2012), the aim of this research was to enable learners to construct their own programs, and therefore no tutors were involved.

The mobile applications in these examples were all designed with the aim of supporting learners. Indeed, the advent of mobile phones for learning offers new opportunities to extend the benefits of learner-centered design software to mobile learning tools (Luchini et al. 2002). Learner-centered design focuses on a learner as a user who has changing needs due to learning, and who needs support to learn by doing (Soloway et al. 1994; Guzdial et al. 1995). By involving the learner in the design and consequently the evaluation phase, the potential of meeting the learners' needs is maximized. This research was guided by the principles of learner-centered design.

A concern among researchers is the evaluation of mobile technologies for learning (Traxler & Kukulska-Hulme 2005; Taylor 2006; Vavoula & Sharples 2009; Jones et al. 1999). One framework that was proposed was the three-level evaluation framework (Vavoula & Sharples 2009). These levels are Micro, Meso and Macro levels. The micro level evaluates the usability of the application and seeks to find out if the application is designed in such a way that it is usable. The meso level evaluates the user experience and seeks to find out if the use of the application is effective and what the learners'

experiences were while using the application. Indeed, evaluation models such as the CIAO model (Jones et al. 1999) have outlined that while evaluating educational technology one should consider data about learners' interaction with the software. This can be evaluated using log analysis that yields data about learners' interaction with the tool (Taylor 2006). Lastly, the macro level evaluates the impact of the application on learning practices. This research considered these aspects during evaluation.

Undoubtedly, mobile phones provide an opportunity to be used as programming environments and there are existing recommendations for design and evaluation of these technologies. Yet, the idea that mobile platforms are more attractive for programming based on the belief that learners like mobile platforms was challenged when learners indicated a preference of the desktop to the mobile environment for programming (Azadmanesh et al. 2014). Their arguments against smartphones included the small screen size, limited performance and battery life, and feature limitations in mobile apps (Azadmanesh et al. 2014). The research in this thesis was motivated by such limitations of mobile phones.

3.3.1 Limitations of mobile phones

Despite the advantages of ubiquity and flexibility that mobile phones present, they also pose several limitations. The key limitation of handheld technology for the delivery of learning objects is the small screen that is available (Churchill & Hedberg 2008). Consequently, there are recommended guidelines for designing scaffolds for handheld learning tools.

The first recommendation is to sequence the learning task into multiple handheld screens (Luchini et al. 2002). This design guideline is supported by using activity decomposition that develops separate workspaces for each component task (Luchini et al. 2004) to package contents in small chunks (Elias 2011). As earlier noted, implementation of programming processes in a single interface such as in GPCEditor (Guzdial et al. 1998) is more suitable for PC programming environments than mobile environments.

A second recommendation is to tightly couple tools and scaffolds to the current activity (Luchini et al. 2002). This guideline addresses the challenge of making scaffolds visible onscreen while not displaying so much information that the handheld tool becomes unusable (Luchini et al. 2003). Indeed, it was recommended that when developing educational software for handheld computers with small screens, whenever possible design interface elements should serve a dual role by providing both functionality and scaffolding (Luchini et al. 2004). This study explored these guidelines in designing scaffolding techniques for a mobile programming environment.

In addition to these recommendations, other works indicate that the following strategies could address the small screen sizes of mobile phones while designing for learning:

i. Minimize scrolling as much as possible (Churchill & Hedberg 2008). Scrolling can be reduced by placing navigational features near the top of the pages in a fixed place (M. Jones et al. 1999). Touch screen devices also enable swiping across, which could be used to move between different page views and hence minimize scrolling downwards.

ii. Provide one step interaction, which can be achieved by immediate update upon interacting with a widget or a button (Churchill & Hedberg 2008).

iii. Use focus and content visualization technique. Users can view local information they are interested in (focus) in detail on a segment of the screen, while other peripheral information (context) is shown in the surrounding area with the reduced granularity of detail (Adipat & Zhang 2005).

Related to the limitation of small screen sizes, especially on touch-screen mobile phones, is the soft keypad that pops up when typing, hence literally covering nearly half the screen. The small size of the keypad also presents a limitation for those with poor manual dexterity or fat fingers and those who have difficulty in selecting tiny buttons on mobile devices (Siek et al. 2005). While typing is needed to write a program, automating some tasks could minimize the disadvantage of having to type on a small keypad. However, care should be taken not to have an interface that is too automated such that students complete the task by rote rather than mindfully engaging and learning about the task (Luchini et al. 2004).

These design recommendations were explored while designing the scaffolding techniques for a mobile programming environment, as discussed in the next chapter.

3.4 Learning Programming using Mobile Phones

The ubiquity of mobile phones provides an opportunity to use them as programming environments outside the classroom, especially in resource-constrained environments. There are some existing applications that enable learning of programming using mobile phones by providing static text, visual environments or ability to construct programs.

Some applications enable learning of programming using tutorials and exercises on the mobile phone. For example, mJeliot enable learners to make predictions about execution behavior of code (Pears & Rogalli 2011). Another example is Sortko that was designed for learning sorting, where the learner selects a sorting algorithm and then applies it on a sequence of numbers. In addition, algorithm visualization has been implemented on mobile devices (Hürst et al. 2007). Recently, a study

investigated the use of mobile technology and Facebook as tools to support the learning of programming through discussions, chats and brainstorming among novices (Maleko 2014). However, the constructivist theory dictates that learning of programming requires a more active role by the learner than just viewing content. Further, it was not the aim of this research to incorporate the use of a social media tool such as Facebook.

Some mobile programming environments enable creation of GUIs (such as Mobidev (Seifert et al. 2011)), others enable creation of mobile applications (such as TouchDevelop (Tillmann et al. 2011)), while others enable creation of standard programs that can run on a PC (such as Sand IDE[7]). Mobidev (Seifert et al. 2011) is a mobile programming environment that was developed to create simple GUI applications in three ways: by defining the UI in code; by using a graphical GUI designer; and by drawing a sketch of the desired UI on a piece of paper that is photographed with the mobile phone's camera and further transformed into a UI. However, despite acknowledging that mobile phones have limitations, MobiDev did not offer design techniques to overcome these limitations. Evaluation of Mobidev measured time-on-task and used the t-test to calculate the significance between creating a UI using the GUI designer and creating one using a sketch builder. These metrics provided an indication of what could be evaluated to measure the effect of using scaffolding techniques on a mobile programming environment. The results showed that participants preferred taking photographs of drawn sketches that were then translated into UI than they did creating one using the GUI designer. However, the application of Mobidev differs from the one of this study since the aim was not to transform paper prototypes into executable code.

Recent work by Microsoft enables development of mobile apps using a new language - TouchDevelop - on the TouchDevelop programming environment where much of the code is created by tapping through menus (Tillmann et al. 2011). TouchDevelop is intended to let users customize their phone's behavior to provide real-time support for their personal lives (Athreya et al. 2012). TouchDevelop also provide fading mechanism such as providing instructional prompts in the first program, then encouraging the user to try and complete the program on their own in the second program. However, TouchDevelop (Figure 3.1) is a specialized language that was designed for a visual programming environment that creates mobile applications. In contrast, this study does not develop a specialized language.

App Inventor (Figure 3.2) is a visual "blocks" programming language designed to introduce learners to programming through creation of mobile applications (Wolber 2011). Learners create applications by dragging and connecting various blocks. App Inventor has been successful in

[7] http://goo.gl/708IuE

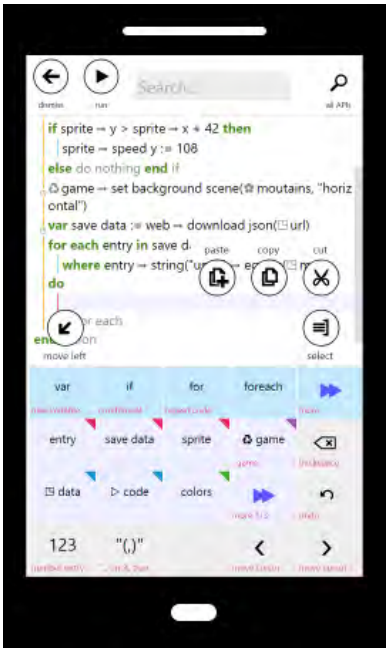


Figure 3.1: TouchDevelop interface on a mobile device (Source: https://www.touchdevelop.com/)

Figure 3.2: Example of an AppInventor program Source: (Wolber 2011)

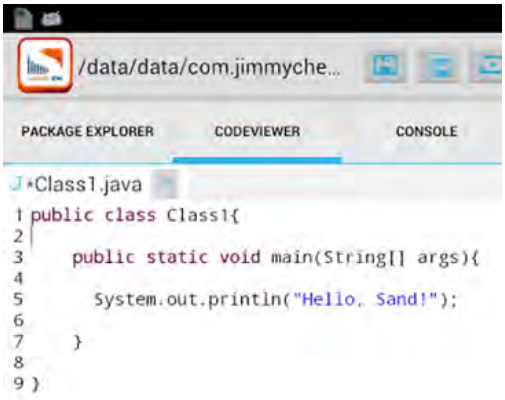


Figure 3.3: SAND IDE (Source: Google Play Store)

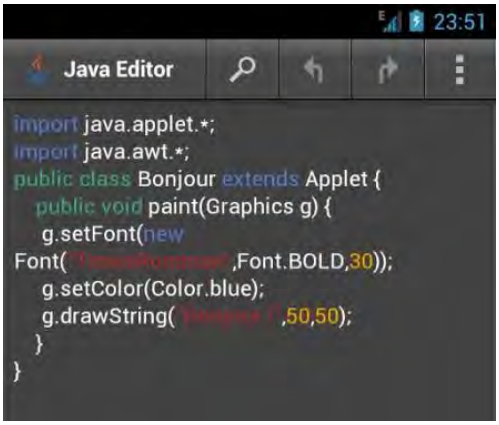


Figure 3.4: Java Editor (Source: Google Play Store)

motivating learners to create real world applications and has been widely used (Wolber 2011; Wagner et al. 2013; Roy 2012). In contrast, the aim of this research is to support construction of programs that are typically taught in an introductory course using Java, as opposed to creating mobile apps such as in TouchDevelop and App Inventor.

There are several mobile IDEs for Java programming available on the Google Play store, such as Sand IDE (Figure 3.3) and Java Editor (Figure 3.4). However, the interfaces of these IDEs mostly mimic PC-based IDEs and they do not offer scaffolding techniques that could support a novice learner

or address the limitations of mobile phones. Similarly, mobProg was designed to offer a platform for creating Java programs on a mobile phone (Hashim 2007). The design of mobProg was based on scenarios and much of the testing was done using an emulator and not with real learners. Further, mobProg enabled writing of Java programs much the same as a PC IDE would, with the addition of syntax highlighting and ability to compile and run the program.

Existing programming environments on mobile phones seem to be based on mobile applications, specialized languages, viewing static material, block-based languages, or exporting IDE concepts and environments directly from desktop environments to the mobile context. Mobile programming environments that use less graphical displays or text to create Java programs and that address the limitations of mobile phones seem to be missing. This study aimed to addresses this gap.

3.5 Summary of Gaps and Opportunities

The related work highlighted some gaps that motivated this study. These are summarized below.

i. There is need to support novice learners of programming in:
   a. Resource constrained environments such as in developing countries.
   b. Object oriented courses taught using programming languages such as Java.
   c. Fundamental programming skills such as combining different parts of a program into a working program. This implies that the needs of learners should be understood.

ii. Most existing PC IDEs are complex, use highly graphic interfaces, or work in integrated architectures that may not be suitable to implement as is on a mobile programming environment.

iii. Use of mobile phones for learning is underexplored in developing countries, especially in subjects such as programming. Further, existing techniques for supporting learners, such as using SMSs and chats, may not fully support learning through doing which is encouraged in learning programming.

iv. Existing mobile programming environments have some limitations, while some differed with the aim of this study:
   a. The IDEs that are used to create standard programs mostly mimic PC IDEs and they do not provide scaffolding techniques that are specifically designed to address the limitations of mobile phones.
   b. Some IDEs are used to convert paper prototypes into user interfaces, which was not the aim of this study.
   c. Some IDEs enable creation of mobile applications, which was not the aim of this study.

d. Some IDEs use specialized languages that cannot be trivially applied to Object Oriented languages such as Java.

These gaps implied that there was need to provide a programming environment on a mobile phone that included scaffolding techniques specifically designed for mobile phones and designed based on learners' needs. Therefore, the next logical question was, which scaffolding techniques would support Java programming on a mobile phone? Once such scaffolding techniques were designed, it was deemed important to establish their effect on constructing Java programs on a mobile phone. Consequently, and as described in Section 1.3, the following two research questions were posed:

1. *Which of the theoretically derived scaffolding techniques support Java programming on a mobile phone?*
2. *What is the effect of using scaffolding techniques to construct Java programming on a mobile phone?*

This study was conducted to address these research questions.

The related work provided some opportunities that could be explored when designing scaffolding techniques on a mobile phone. These are summarized below.

i. When designing scaffolding techniques on a mobile programming environment, consider:
   a. Decomposition the tasks;
   b. Constraining the order of program creation;
   c. Providing default statements that learners can reuse;
   d. Using a text based environment;
   e. Not using a single interface for all the processes, due to the small size of the screen;
   f. Minimizing scrolling as much as possible;
   g. Provide one step interaction;
   h. Include movable, collapsible, overlapping and semi-transparent interactive panels; and
   i. Use focus and content visualization technique.

ii. Fading of scaffolding can be provided by removing the restriction to the structure of a program.

iii. Use of ideone as a compiler.

iv. Design of scaffolding techniques could follow a learner-centered design and consider recommended guidelines for designing on mobile environments.

v. Evaluation could consider the following:
   a. Use of a three-level framework that addresses micro, macro and meso levels;
   b. Using log-analysis to measure user interactions;
   c. Measuring metrics such as time-on-task; and

d. Use of t-test.

The following chapter discusses the design and implementation of scaffolding techniques for a mobile programming environment and indicates how the identified opportunities were integrated within the design process.

Chapter 4 Design and Implementation of Scaffolding Techniques

The proposition of this research is that programming environments on mobile phones could include scaffolding techniques that are specifically designed for mobile phones, and designed based on learners' needs. Therefore, the first step was to understand the needs of programming learners. To achieve this, an online survey was used to elicit the challenges that learners face in the subject. These learner-cited challenges informed the design of the mobile intervention. Such a learner-centered design (LCD) approach recognizes that learner-centered software incorporates scaffolding to support learners as they do new work (Quintana et al. 2001). LCD was relevant to this study since programming is learnt by doing, and the aim was to scaffold learners as they construct programs. This chapter begins by describing LCD.

Apart from considering challenges that are faced by learners, limitations of mobile phones were considered. Using specific examples, this chapter reports on learner-cited challenges and mobile phone limitations. Thereafter, these challenges and limitations are used as requirements in the first phase of a six-level scaffolding framework. The requirements are then applied to the second to fifth levels of the framework, leading to the selection of specific scaffolding techniques. The second to fifth levels of the scaffolding framework are: categorizing the challenges into cognitive types (Quintana et al. 2004); selecting the type of scaffolding to use to address the challenges (Jackson et al. 1998); selecting scaffolding guidelines that could address the challenges; and selecting scaffolding strategies that implement the guidelines (Quintana et al. 2004). The sixth level consists of selecting specific scaffolding techniques that could support construction of Java programs on a mobile phone. This chapter shows how these scaffolding techniques were implemented on an Android platform. The designed scaffolding techniques were of three types: (i) static scaffolding; (ii) automatic scaffolding: (iii) and user-initiated scaffolding. The chapter then presents a system overview of the developed mobile application, discussing its various modules. Using an example, the chapter then shows how a program can be created using the designed scaffolding techniques. The chapter concludes by describing a non-scaffolded environment that was used in the experiments.

4.1 Learner-Centered Design

To differentiate between user-centered design (UCD) and learner-centered design, a structured definition for learner-centered design was provided (Quintana et al. 2000). The differences were described along three aspects: the targeted audience; the central problem being addressed; and the underlying approach that each paradigm takes. Table 4.1 shows the differences between UCD and LCD as described by Quintana, Krajcik and Soloway (Quintana et al. 2000).

Table 4.1: Differences between User-Centered design and Learner-Centered design

	User-Centered Design	**Learner-Centered Design**
Targeted audience	Users are assumed to understand the work domain in which they are working. Users often perform tasks that are similar. Hence the design of tools can rely on a representative user. Users often need tools to complete their work, not trying to learn about their work using the tools.	Learners are assumed to have no knowledge about their work domain. Learners often have different skills and backgrounds and perform varying tasks. Learners often need tools to learn about their work, and not just to complete the work. Hence the tools need to change as a learner's skills grow.
Central problem being addressed	The user uses a tool to execute a series of action towards a specific goal. Once the actions are executed the user evaluates the tool's resulting state in terms of their goals.	In addition to a learner using a tool to execute a series of actions towards a specific goal, the learner uses a tool to gain skills in the work domain and build on their expertise.
Underlying approach	Using a theory of action that explains how users generally perform tasks in a given scenario.	In addition to understanding how learners generally perform tasks in a given scenario, LCD uses existing theories that support learning through active engagement, such as constructivism or social constructivism. Learners need additional support to understand the work domain.

Table 4.1 indicates three key differences between UCD and LCD:

i) The focus of UCD is to support users who are knowledgeable about their work and who often perform similar tasks, to complete their tasks. The focus of LCD is to support learners with varying learning skills who often perform different tasks, to gain knowledge in a new work domain.

ii) In UCD, the aim is to have a usable tool that supports a user to reach a specific goal. In LCD, the aim is not only to have a usable tool, but also one that enables a learner to build their skills.

iii) In UCD, design of tools is based on how users generally perform a task. In LCD, in addition to designing tools based on how learners generally perform a task, the focus is on designing tools that provide support while learners actively engage in a task.

In relation to these three differences, LCD was suitable to this research in the following ways: (i) the aim of the research was to support novice learners of programming who have different abilities; (ii) the aim of the research was to enable learners to actively construct programs; and (iii) the aim of the research was to provide support (scaffolding) to learners as they construct programs. Further, it has been emphasized that the focus of an eLearning system should be to support the learning process, motivate the learners, and adapt itself to the needs of the learners (Dhar & Yammiyavar 2012).

The LCD methodology that this research used relates to the TILT model (Tools, Interfaces, Learners' needs, Tasks) (Soloway et al. 1994). Figure 4.1 shows the overall structure of the LCD model adapted in this study. The learners' needs were placed at the center of the design process and include the challenges faced by learners of programming, the limitations of mobile phones and the feedback obtained during evaluation of the designed prototype. The tasks refer to activities that need to be undertaken in the software; tools must be adaptable in order to support a learner to grow in expertise; and interfaces designed must take into account the use of different media and modes of expression (Soloway et al. 1994). The scaffolding techniques are used by the learners to complete programming tasks. From the review of related work, some scaffolding techniques such as decomposing tasks into smaller parts, and constraining the order of program creation and provision of default code, were recommended. The designed scaffolding techniques should be adaptable. The programming environment offered a text-based interface and required the use of the Internet in order to use the online compiler, ideone. The TILT model has also been adopted in other studies such as one that designed an adaptive phone interface for low-literate users (Lalji & Good 2008).

Following the LCD methodology, the first step was to understand the needs of programing learners as discussed in the next section.

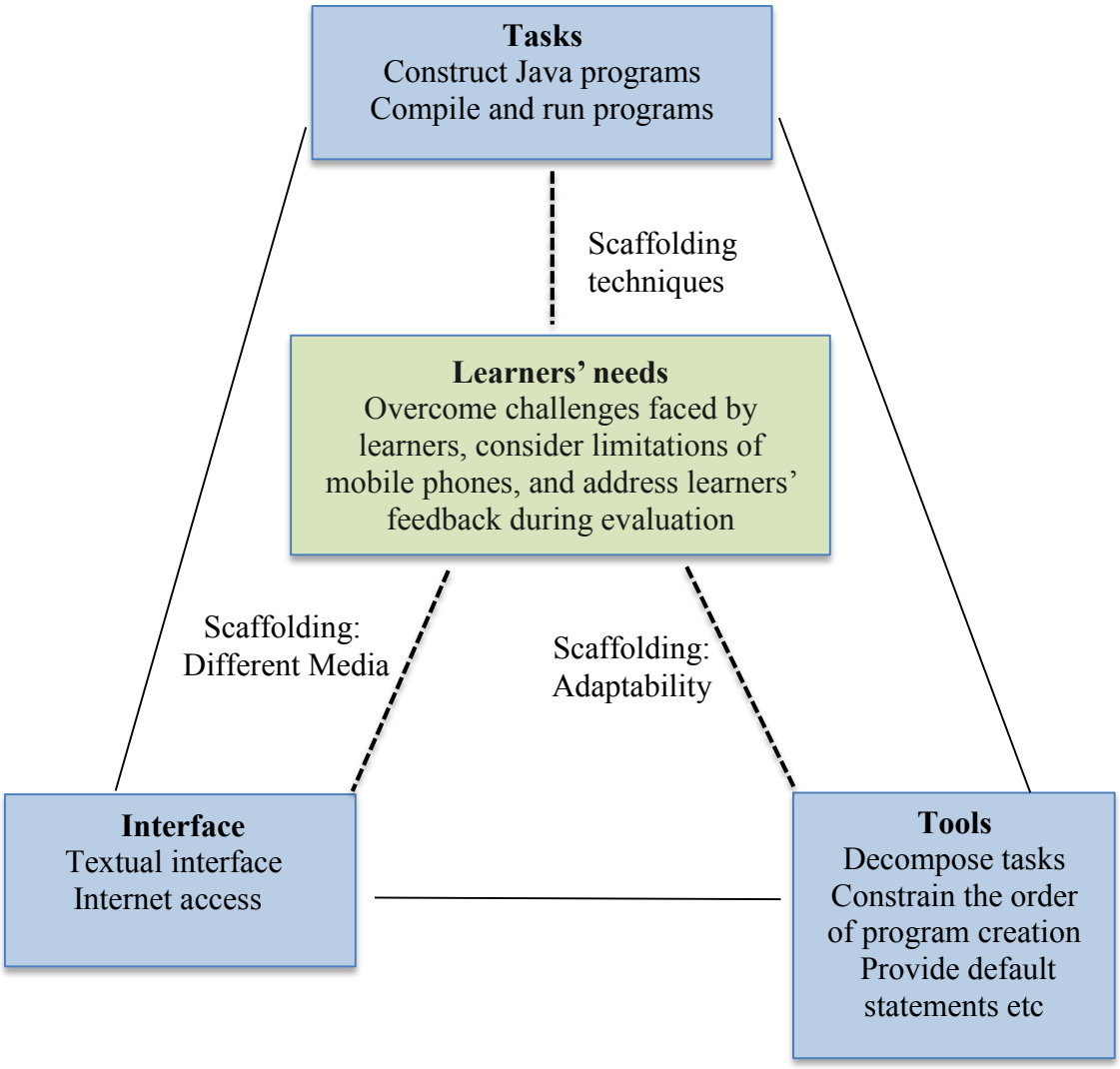


Figure 4.1: LCD methodology followed in this study as adapted from the TILT model

4.2 Requirements

4.2.1 Learner-cited challenges

In order to understand the needs of learners of programming, an online survey was conducted among 160 learners of programming from three universities: University of Cape Town (UCT) (61 learners); University of Western Cape (UWC) (37 learners); and Kenya Methodist University (KeMU) (62 learners). The three universities were chosen because of their convenience in terms of having established contacts. This survey was conducted in April 2013. Although the study targeted 210 participants (70 from each institution), 160 complete submissions were received, a response rate of 76%. Participation in the survey was voluntary.

An electronic questionnaire was sent to students. At UCT, the invitation to participate in the survey was sent to Computer Science class groups via the local learning management system. At UWC,

the invitation to participate in the survey was sent to first year Computer Science students' email addresses by their lecturer. At KeMU, the invitation to participate in the survey was sent to the students' online forum.

The questionnaire had four sections:

1. Demographic information;
2. Learners' experience and challenges with programming;
3. Access to and ownership of technology; and
4. Experience with using mobile devices to construct programs.

The survey responses were anonymous and no incentives were offered to the respondents.

Respondents Demographics

The distribution of the respondents over the participating universities is presented in Figure 4.2. The distributions of the respondents according to course of study and degree of study are presented in Figures 4.3 and 4.4 respectively. The distributions are shown in both percentages and absolute numbers of total respondents.

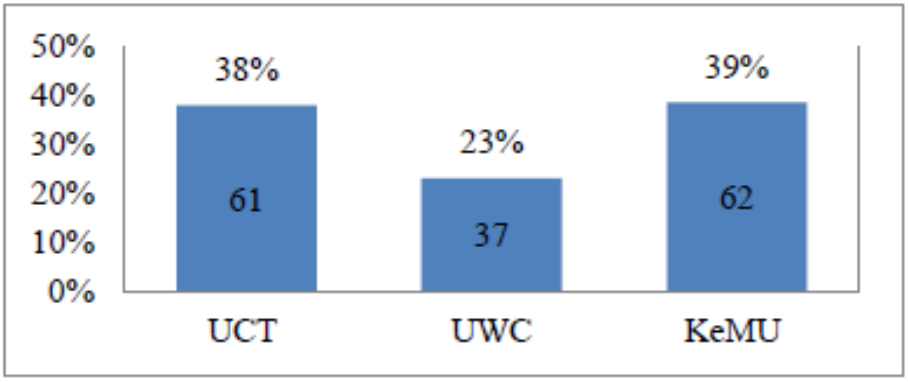


Figure 4.2: Distribution of all respondents according to university

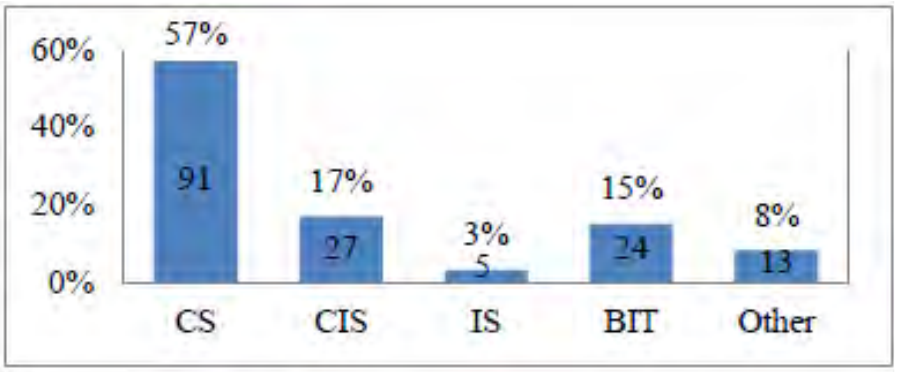


Figure 4.3: Distribution of all respondents according to course of study

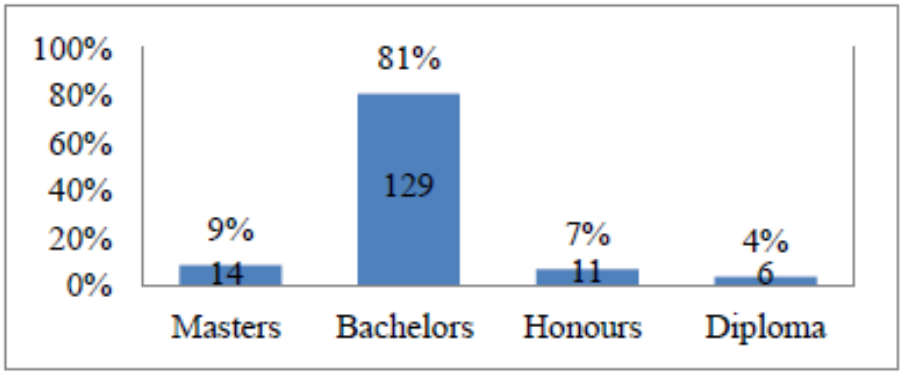


Figure 4.4: Distribution of all respondents according to degree of study

Learners from computing related courses were specifically targeted because programming is part of their curriculum. Other respondents were learners in courses such as Information Science, Engineering and Actuarial Science (Figure 4.3). The learners in these other courses learn basic programming courses as indicated in their course curricula. A high number of respondents in Computer Science (CS) can be explained by the targeted announcements via emails and class announcements to undergraduate Computer Science groups and classes at UCT and UWC. Further, the lecturers at UWC who emailed their students were lecturers of undergraduate programming courses.

This also explains the high number of undergraduate participants in Figure 4.4. KeMU offers both Computer Information Systems (CIS) and Business Information Technology (BIT), which explains the almost equal distribution in the two courses in Figure 4.3. KeMU also offers other courses such as Health Systems Management and Business Administration, which had a few respondents who took part in the survey. Such respondents formed part of the other courses in Figure 4.3, and indicated having learnt programming out of personal interest.

Findings

64% of the learners who responded to the survey indicated that they had not studied any programming course prior to joining university. This indicates that most of the learners join higher education without any experience in programming. 98% of the respondents indicated that they own mobile phones. The learners were also asked if they had constructed programs on a mobile phone. 91% of the total respondents indicated that they had never constructed programs on a mobile phone. This indicates that the use of mobile phones as programming environments is underexplored. Of the remaining 9%, some indicated that they had used QPython, which is a Python script engine that can run on Android devices. The learners who had used QPython cited challenges such as: no allowance for indentation of code; small screen size, which is restrictive; and not being able to transfer the code to a computer in the required format.

76% of the total respondents indicated one challenge or the other that they face while learning programming. These challenges are shown in the second column of the table in Appendix A. For the sake of providing detailed illustration, the three challenges below are selected from the ones cited, and will be used as running examples for the rest of the chapter.

i. Difficulty in combining required program parts into a working program and hence making logic or sense out of a program is challenging. This challenge is further supported by research pointing to two key problems preventing success at programming among novice learners (Guzdial et al. 1998).
   a. *Decomposition problem:* Learners have difficulty choosing which of the available program components are needed for problem solution.
   b. *Composition problem:* Even when learners identify program components, they have difficulty assembling the modules into a proposed solution.

ii. Unclear error messages while debugging. This challenge is supported by a study that indicated that even though compilers may flag some of the error messages while programming, often the error messages are so cryptic to students that they have a hard time understanding them (Hristova et al. 2003). Importantly, what some may assume as basic and simple in programming may be complex and misunderstood by others (Mohamed et al. 2011). This is illustrated by a study in which only a handful of learners managed to discover that Java is case sensitive, and a number of learners indicated that the purpose of 'import.java.io' is to import the input and output of the program to other systems (Mohamed et al. 2011).

iii. Small screens of mobile devices pose a challenge in using them to learn programming.

4.2.2 Limitations of mobile phones

As indicated above, a majority of the surveyed learners indicated that they had never constructed programs on a mobile phone. Some of the reasons given as barriers to using these devices for programming were:

i. A preference for bigger screens.
ii. Programming on a phone would require having knowledge of the language since it would be difficult to refer to help when stuck.
iii. Learner not aware of any mobile IDEs.
iv. Not having a smartphone.
v. Typing on the small keyboards would be difficult.
vi. Phone has minimal memory hence storage and compilation would be a problem.

vii. Data and airtime costs would be expensive.
viii. Learner has never had the need to write programs on a mobile phone.
ix. Programs would load slowly because even currently available apps take a while to load.
x. Accessing the special characters needed for programming would be too cumbersome and time-consuming on a mobile phone.

Certainly, many factors have to be taken into considerations when it comes to mobile phones since they present potential usability problems (Kukulska-Hulme 2005; Kukulska-Hulme 2007). However, to define the scope of which mobile limitations to consider, and as pointed out by the learners, this chapter will look at the small screen size and the small keypad. Considering these limitations is crucial because, in writing a program, a learner must see on a screen display what they are constructing using the mobile phone keypad. Further, as was discussed in the related work, some design recommendations were provided to overcome these two limitations. These recommendations were considered while designing scaffolding techniques using a six-level scaffolding framework.

4.3 Six-Level Scaffolding Framework

A six-level scaffolding framework was used to select scaffolding techniques that could support Java programming on a mobile phone (Mbogo et al. 2014). The framework was based on a theory-driven model which has four main phases (Quintana et al. 2004): challenges experienced by learners; cognitive type of the learning challenges; scaffolding guidelines; and scaffolding strategies that implement the guidelines. In this study, the learner challenges included limitations of mobile phones. In addition to these four phases, two other phases were added in order to accommodate: a model for categorizing the types of scaffolding to use (Jackson et al. 1998); and selection of scaffolding techniques that could support construction of Java programs on a mobile phone. The combination of the four-phase model, categorizing the types of scaffolding, and the process of selecting scaffolding techniques form a six-level framework that was used in this study.

Having identified learners' challenges and mobile phone limitations, the next step was to integrate them within a scaffolding framework. The aim of this exercise was to guide the selection of scaffolding techniques that could be implemented on a mobile phone to support construction of Java programs. This was done by following the six-level framework in the following steps:

i. Step 1: Identify learner challenges and limitations of mobile phones. These have been identified in the previous section.
ii. Step 2: Categorize each learner challenge into either of three types of cognitive challenges (Quintana et al. 2004):
    a. Sense making, which involves the basic operations of interpreting data.

b. Process management, which involves strategic decisions in controlling an inquiry process.
c. Articulation and reflection, which is the process of constructing, evaluating and articulating what has been learnt.

iii. Step 3: Identify what kind of scaffolding the learner challenge may need, from three types (Jackson et al. 1998):
   a. Supportive scaffolding, which offers support for doing the task while the task itself remains unchanged.
   b. Reflective scaffolding, which offers support for thinking about the task.
   c. Intrinsic scaffolding, which offers support that changes the task itself and reduces complexity.

iv. Step 4: Identity the scaffolding guideline that specify ways in which tools can modify tasks to help learners overcome the learning challenges. Seven scaffolding guidelines have been recommended to address the learner cognitive challenges (Quintana et al. 2004). These were redefined to fit into this study.

   To address *sense making*, these guidelines were recommended (Quintana et al. 2004):
   a. Guideline 1: Use representation and language that bridge learners' understanding of programming.
   b. Guideline 2: Organize the scaffolding techniques around the semantics of the programming language.
   c. Guideline 3: Use representations that learners can inspect in different ways to reveal important properties about underlying data.

   To address *process management*, these guidelines were recommended:
   d. Guideline 4: Provide structure for complex tasks and functionality.
   e. Guideline 5: Embed expert guidance about programming practices.
   f. Guideline 6: Automatically handle routine tasks.

   To address *articulation and reflection*, this guideline was recommended:
   g. Guideline 7: Facilitate on going articulation and reflection during program construction.

v. Step 5: Associate each guideline with proposed scaffolding strategies that could support construction of programs on a mobile phone. These scaffolding strategies were recommended to provide specific types of implementation approaches that could achieve a given guideline (Quintana et al. 2004). For example, in order to provide structure for complex tasks and functionality (guideline 4), a scaffolding strategy that

could be used is to restrict a complex task by setting useful boundaries for learners. Appendix B contains the complete table that shows the recommended strategy for each guideline.

vi. Step 6: Following the selected scaffolding strategies in step 5, propose specific scaffolding techniques that could support constructions of Java programs on a mobile phone.

The next section describes how steps 2 to 6 were applied to the three learner challenges selected as examples. In order to implement the selected scaffolding techniques in a mobile programming environment, an Android application was developed for Android version 2.2 and later.

4.4 Implementation of Scaffolding Techniques

4.4.1 Learner challenge 1: Difficulty in connecting program parts into one

Step 2: Categorizing challenge into a cognitive type

This learner challenge is one of sense making because it involves being able to make sense out of a program and its constituent parts. It is also one of process management because it requires scaffolding strategies that can control the learner's inquiry process so that the learner can effectively make sense of how the different program parts connect into one.

Step 3: Identifying the scaffolding types that the learner challenge may need

Supportive scaffolding can provide support while the learner is attempting to make sense of the different parts and functionality of a program. At the same time, intrinsic scaffolding can reduce the complexity while creating the program.

Step 4: Identifying scaffolding guidelines that may address challenge cognitive type

In order to support sense making, using representation and language that bridge learners' understanding was selected as a scaffolding guideline. In order to support process management, providing structure for complex tasks and functionality was selected as a scaffolding guideline. These two scaffolding guidelines can be met by the scaffolding strategies described next.

Step 5: Select scaffolding strategies that implement the scaffolding guidelines

In order to provide representation and language that bridge learners' understanding, the following two scaffolding strategies were selected (Quintana et al. 2004)

a) Provide visual organizers to give access to functionality.
b) Embed expert guidance to help learners use the content.

The first strategy was selected because it offers supportive scaffolding. By providing a visual organizer, learners could access and interact with the software functionality in a way that allows them to think about the deeper concepts and structure (Quintana et al. 2004). Such a visual organizer could enable the learner to see the different parts of a program, and through interaction with these parts, see how these parts connect with each other. The second strategy was selected because it offers intrinsic scaffolding. Using embedded expert guidance, learners could be prompted towards proper creation of the program parts, and how to connect the program parts into a full program.

In order to provide structure for complex tasks and functionality, the following scaffolding strategy was selected (Quintana et al. 2004)

c) Restrict a complex task by setting useful boundaries for learners.

This strategy was selected because it offers intrinsic scaffolding. By restricting the process of completing a task, learners could systematically move from one part to another and therefore learn how to combine different program parts into one. This could reduce the complexity of program creation. The discussion below addresses each of these three strategies and the associated scaffolding techniques that were selected and implemented on a mobile phone.

Step 6: Propose and implement specific scaffolding techniques that could support constructions of Java programs on a mobile phone

Provide visual organizers in order to give access to functionality

This strategy was implemented by providing a layout of the parts of a Java program in order to give an overview of the program. The order of the parts in the interface was guided by standard Java coding guidelines (Sun-Microsystems 1997), where a Java source file has the following ordering: beginning comments, package and import statements, and class and interface declarations. Figure 4.5 shows the designed main interface with parts of a Java program.

This layout at the main interface uses clickable buttons that provide additional functionality: (i) collapsible and expandable views; (ii) access to create individual chunks of the program. Further, the use of expandable and collapsible buttons is intuitive to learners who have used PC IDEs, such as Eclipse, that provide foldable interfaces. Besides, such a collapsible and expandable interface was recommended for small screens (Churchill & Hedberg 2008). In addition, the use of the buttons for both the layout and the additional functions provides a dual role of functionality and scaffolding, which was recommended as a way of designing handheld devices (Luchini et al. 2004).

Provision of a program layout is a static scaffolding technique since it does not change or fade away with time. Further, while creating a program, this overview has to be used in order to access the

different parts of a program. Such scaffolds were termed as ‘essential’ and the design of essential scaffolds was encouraged because, if designed as optional, learners may bypass them and miss the support needed to perform certain tasks (Quintana et al. 2002b).

This scaffolding technique provides atomic simplicity, a characteristic of constructivism, by providing a visual layout showing the most basic units of Java programs. Further, the interface was designed in a simple layout that supports learners to see the different parts of a Java program. Through interaction, learners are able to create code that combines these parts into a simple program.

Figure 4.5: Main interface showing only the main class activated

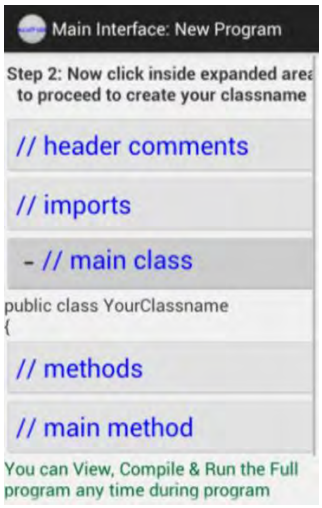


Figure 4.6: Main class default code

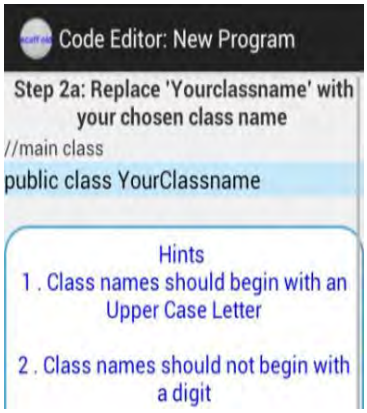


Figure 4.7: Creating the main class

Figure 4.8: Method default code

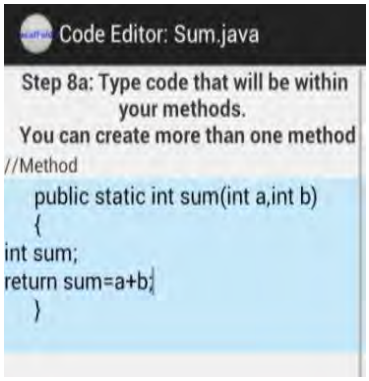


Figure 4.9: Creating a method

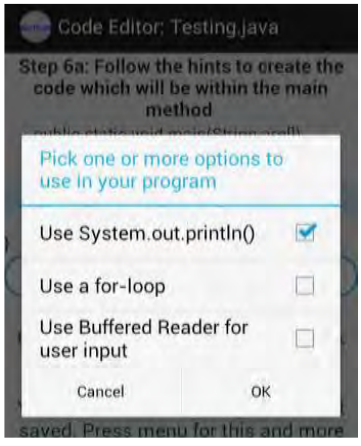


Figure 4.10: dialog for default statements

Embed expert guidance to help learners use content

This strategy was implemented in two ways: (i) providing supportive guidance to enable use of the scaffolded environment; and (ii) providing default code related to specific parts of a program. Figure 4.5 and Figure 4.6 show steps at the top of the screen that guide the learner on which button to click. Both figures show instructions at the bottom of the screen. Steps and instruction were faded after the second program, after which they could be viewed by selecting related menus. Steps and instructions are automatic scaffolding techniques that fade with time. Figure 4.6 shows implementation of default code in creating the main class (revealed when the button is clicked), which the learner could then edit (as in Figure 4.7). Figure 4.8 shows a method's default code, which the learner could then edit (as in Figure 4.9). Figure 4.10 shows a dialog box that pops up when creating the main method and the method. On selection of any of these, the related default code is populated in the text field. These default statements were based on standard coding guidelines. For example, there should be no space between a method name and the parenthesis "(" starting its parameter list (Sun-Microsystems 1997) as shown in Figure 4.9. Provision of default code is automatic scaffolding that is provided by default. Provision of examples is user-enabled scaffolding since a learner has to initiate its use. Provision of default code supports active exploration by supporting correct construction of program parts.

Restrict a complex task by setting useful boundaries for learners

This strategy was implemented by restricting a learner to complete a program in a certain order. First, the main class is completed because it is also used as the name of the program. Then the header comment is completed in order to guide the learner to give the description of the program. Then the main method is completed as the entry point of the program. Then the methods and import sections can be completed if needed. Figure 4.5 shows only the main class activated when the program is started, while Figure 4.11 shows the main class completed (in green) and the header comment activated.

After successful completion of three programs in this restricted order, a learner is presented with an interface where all the parts are enabled and the learner is able to complete the program in any order (Figure 4.12). A similar technique was used in a recent study where fading of scaffold was realized by using open exercises that do not enforce any specific program structure or approach (Vihavainen et al. 2013). While the learner can work with the interface in Figure 4.12, they are able to go back to the restricted interface if they wish to. This also provides structure to complete the task using ordered decomposition (restricted) and unordered decomposition (unrestricted) (Quintana et al. 2004).

Figure 4.11: Main class completed (in green) and header button activated

Figure 4.12: Unrestricted interface

Restriction of a program's creation is automatic scaffolding that then fades over time. This scaffolding technique provides atomic simplicity, which is a characteristic of constructivism, by providing an incremental process of creating the program, one unit at a time. Further, this scaffolding technique provides active exploration by guiding the learner on the order of interaction with the program parts. In addition, after the learner reaches the unrestricted interface, they gain user control such that they can choose which interface to work on. User control is a characteristic of constructivism.

4.4.2 Learner challenge 2: Difficulty in debugging errors in programs

Step 2: Categorizing challenge into a cognitive type

This learner challenge is one of process management because it requires scaffolding strategies that can contribute to the learner's inquiry process while debugging a program. It is also one of articulation and reflection because it contributes to thinking about and evaluating what is been constructed.

Step 3: Identifying the scaffolding types that the learner challenge may need

Intrinsic scaffolding could be provided to reduce the complexity while creating or debugging the program. Reflective scaffolding could be provided to enable the learner to think about the program.

Step 4: Identifying scaffolding guidelines that may address challenge cognitive type

In order to support process management, the intervention should embed expert guidance about the scientific practice, in this case being Java coding guidelines. In order to support articulation and reflection, the intervention should provide ongoing articulation and reflection during completion of the program. These two scaffolding guidelines were met by the scaffolding strategies described next.

Step 5: Select scaffolding strategies that implement the scaffolding guidelines

In order to embed expert guidance and to facilitate a learner to reflect about the task, the selected scaffolding strategy is one that embeds expert guidance to clarify characteristics of Java practices (Quintana et al. 2004). This scaffolding strategy was selected because expert guidance that relates to standard Java guidelines could be suitable to support learners to debug programs.

The discussion below addresses this strategy and the associated scaffolding techniques that were selected and implemented on a mobile phone.

Step 6: Propose and implement specific scaffolding techniques that could support constructions of Java programs on a mobile phone

Embed expert guidance to clarify characteristics of Java practices

This scaffolding strategy was implemented in three ways: (i) provision of error prompts; (ii) provision of hints; and (iii) provision of examples.

While a novice learner constructs a program, they inevitably make mistakes that lead to compile time or run time errors. While it was not possible to predict all the types of mistakes that learners could make, this study attempted to address Java syntax related issues. This is because several learners indicated syntax to be a difficulty in the subject. Further, several studies have shown that learners often express difficulties related to the syntax of the language they are using (Apiola et al. 2011; Gaspar & Langevin 2007). Figure 4.13 shows creation of a main class, albeit using an incorrect syntax of starting a Java class name using lower case. If the learner proceeds with this class name creation, then an error message is displayed indicating the same as shown in Figure 4.13. Figure 4.14 shows creation of a main method. Assuming a learner writes the return statement here, an error prompt indicates this error (Figure 4.15). These error prompts are based on standard coding guidelines. For example, a main method should not contain a return statement.

Figure 4.16 shows implementation of hints for the main class. These hints were based on standard coding guidelines. For example, class names should be written with the first letter of each internal word capitalized (Sun-Microsystems 1997). Figure 4.17 shows implementation of examples for the main class. Examples pop up when the example menu is selected.

Error prompts are automatic scaffolding techniques that are displayed when a syntactical error is encountered. They support reflection since the learner is supported to think about the task to correct the error. Hints are automatic scaffolding techniques that are provided by default. Provision of examples is user-enabled scaffolding since a learner has to initiate its use. Error prompts, hints and examples support reflection by enabling the learner to think about how to correct or construct the content of the part they are creating. Reflection is a characteristic of constructivism.

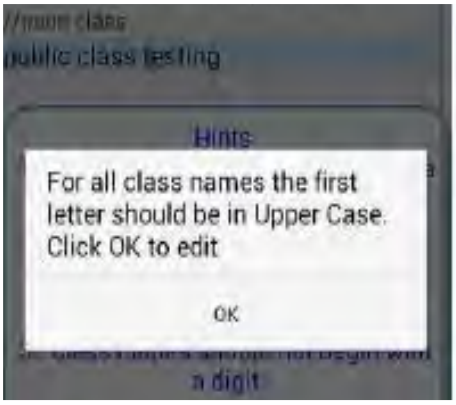


Figure 4.13: Error prompt

Error prompt indicating incorrect completion of main class

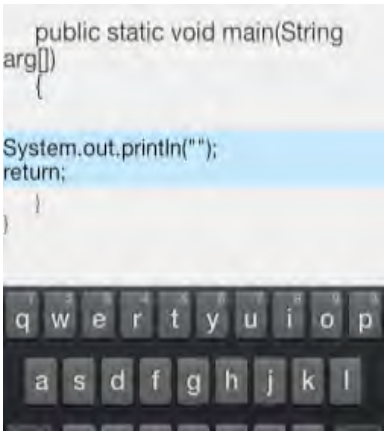


Figure 4.14: Incorrect creation of return statement

Figure 4.15: Error prompt indicating incorrect use of return statement

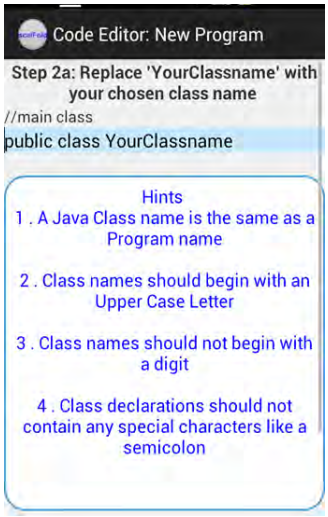


Figure 4.16: Hints displayed when creating main class

Figure 4.17: Main Class example displayed after clicking on a menu item

4.4.3 Learner challenge 3: Small screen size and small keypad of a mobile phone

Step 2: Categorizing the learner challenge into a cognitive type

This learner challenge is one of process management because it requires scaffolding strategies that can support the learner's inquiry process on a mobile device, which has screen size and input limitations.

Step 3: Identifying the scaffolding types that the learner challenge may need

Supportive scaffolding can provide support while the learner is using the small screen size and small keypad to construct a program.

Step 4: Identifying scaffolding guidelines that may address challenge In order to support process management, providing structure for complex tasks and automatically handling routine tasks were selected as scaffolding guidelines. The first scaffolding strategy was selected because in defining the structure of how a program should be created the limitations of mobile phones could be addressed. The second strategy was selected because automating some tasks could minimize the disadvantage of having to type on a small keypad.

The discussion below addresses each of these strategies and the associated scaffolding techniques that were selected and implemented on a mobile phone.

Step 5: Select scaffolding strategies that implement the scaffolding guidelines

In order to provide structure for complex tasks, three scaffolding strategies were recommended (Quintana et al. 2004): (i) restrict a complex task by setting useful boundaries; (ii) describe a complex task by using ordered and unordered task decomposition; and (iii) constraining the space of activities by using functional modes. In order to handle routine tasks, it was recommended to automate non salient portions of tasks to reduce cognitive demands (Quintana et al. 2004).

Step 6: Propose and implement specific scaffolding techniques that could support constructions of Java programs on a mobile phone

Setting boundaries, using ordered and unordered decomposition, and constraining the space of activities by using functional modes

This strategy was implemented in two ways: (i) constructing a program one part at a time; and (ii) viewing the full program.

In the main interface, the learner clicks on the button that relates to the part they wish to work on. This opens an interface with an editor that provides creation of only the selected chunk. For example, Figure 4.18 shows creation of a method. Ability to work on a part of the program at a time uses activity decomposition to package small chunks (Luchini et al. 2004; Elias 2011). This could assist in working with the small screen. Enabling completion of the program one part of a time provides atomic simplicity, which is a characteristic of constructivism. Because of the restriction of a small screen size, which remains unchanged, this scaffold is static and does not fade.

Figures 4.18 shows how working on one program part at a time could assist in addressing the soft keypad taking up nearly half the screen, and hence minimize scrolling. By placing the task to be edited near the top of the screen, the soft keypad does not cover much of the task, if at all. The interfaces show use of navigation labels at the top of the screen as was recommended for small interfaces (Jones et al. 1999). However, for a learner to have a mental image of how the different parts of the program work together, learners should be able to inspect the task they are working on in multiple ways.

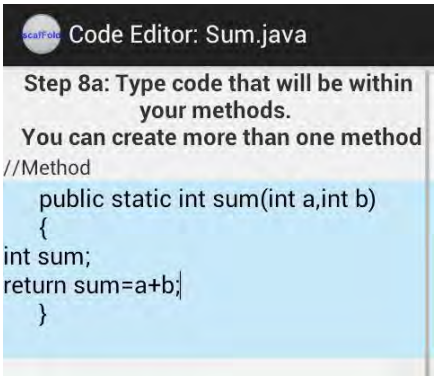


Figure 4.18: Creating method

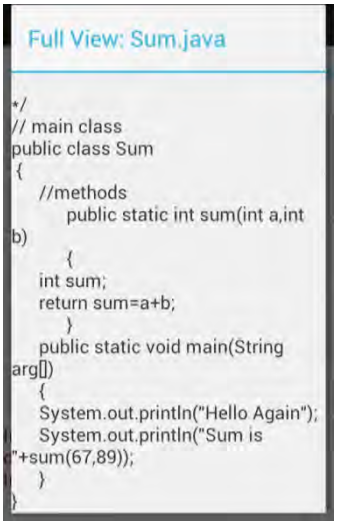


Figure 4.19: Full program as was last saved

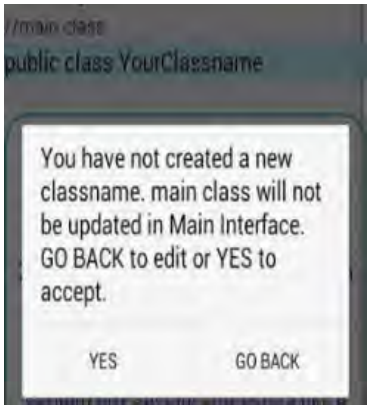


Figure 4.20: Prompt for unchanged main class

In this case, while working on a program part (for example editing the main method in Figure 4.18, a learner could click on the full program menu and view the whole program (Figure 4.19) at the state at which it was last saved. This ability to move between a program part and the whole promotes cognitive growth by keeping the learner connected to the chunks, while at the same time being able to appreciate existence of the whole problem (Ackermann 1996). Viewing of the full program while working on one part supports multiplicity, which is a characteristic of constructivism. Multiplicity encourages provision of multiple perspective of a concept.

Automate non-salient portions of tasks

Because of provision of some default code, the learner is spared from typing all the code from scratch using the small keypad. However, the learner is still required to complete the program parts and hence they need to mindfully engage and hence learn the task, as recommended (Luchini et al., 2004). Further, the learner should be able to exit without completing a program part, but a message indicating that the task has not been changed could assist in making sure that a learner actually completes a task for it to be created in the program (Figure 4.20).

4.4.4 Summary of scaffolding techniques

The design process in the previous subsections has shown how the learner challenges and limitations of mobile phones guided the selection of scaffolding techniques. With each selection, the scaffolding technique was described as either static, automatic or user-enabled. Further, the technique's fading characteristic, if any, was also described. In addition, the related constructivist characteristic was defined. Table 4.2 shows a summary of the derived scaffolding techniques and their associated characteristics. The next section presents an overview of the system.

Table 4.2: Table showing the designed scaffolding techniques, associated scaffolding type, fading capability and the related constructivist principle

Scaffolding Technique	Scaffolding type	Fading capability	Related constructivist principle
Java program overview showing parts of a Java program: header, imports, method, main class, main method. Related to restriction of the order of program creation.	Static	None	Atomic simplicity
Restrict program creation in the order: main class, header, main method, method and/or imports	Automatic	Fades after three successful tasks	Atomic simplicity Active exploration User control
Steps and instructions	Automatic	Fade after the first program	Active exploration
Default code	Automatic	None	Active exploration
Hints	Automatic	None	Reflection
Examples	User initiated	None - back button removes it from the screen	Reflection
Create program a part at a time	Static	None	Atomic Simplicity
Viewing full program while working on program parts	User-initiated	None - back button removes it from the screen	Multiplicity, Reflection
Error prompts	Automatic	None, pressing the OK button removes it from the screen	Reflection

4.5 System Overview

The scaffolding techniques designed in the previous section were implemented on an Android platform. Eclipse was used to write the code for the interface, and PHP and JSON scripts were used to send data to and from the databases. This section presents an overview of the designed mobile programming environment. Two prototypes were used in this study. Figure 4.21 shows an overview of the first prototype with the scaffolding techniques designed in the previous subsections shown in blue. The next subsection presents the second prototype.

4.5.1 First prototype

4.5.1.1 Registration and login

First, the user registered using their email address and created a password. If either the email address or the password fields were empty, an error message was displayed. The purpose of registration was to keep track of the number of users and also to uniquely identify each user for the purpose of computer logs. The users' data were stored in a secure server at the department of Computer Science at University of Cape Town. Using the registered username, the user could log into the application. If the email address or password fields were empty, or the password was incorrect, or the username was not registered, a relevant error message was displayed. Upon successful login, the main interface was displayed. The user's login state was retained unless they logged out.

4.5.1.2 Main interface

The main interface shows steps and instructions. Steps are displayed at the top of the screen while instructions are displayed at the bottom of the screen. The instructions and steps are automatically displayed in the first two programs. In subsequent programs, these can be accessed through a menu. The main interface also shows a layout of five parts of a Java program: header comments, imports, main class, method and main method. This layout provides clickable buttons that expand to reveal default code and allows access to creation of individual chunks. The buttons could also be collapsed to hide the default code. In the first three programs, a learner is restricted to construct a program in a certain order: main class, header, main method, and then method and/or imports. After a learner successfully completes three programs, the main interface changes to one which allows creation of a program in any order after creation of the main class. The full program can be viewed by clicking on a menu. A program is compiled by pressing on a run menu. The output of the program is displayed on a new screen. To exit the output screen, the phone's back button is pressed to return to the main interface. Clicking on any active button related to a program chunk takes the user to the editor.

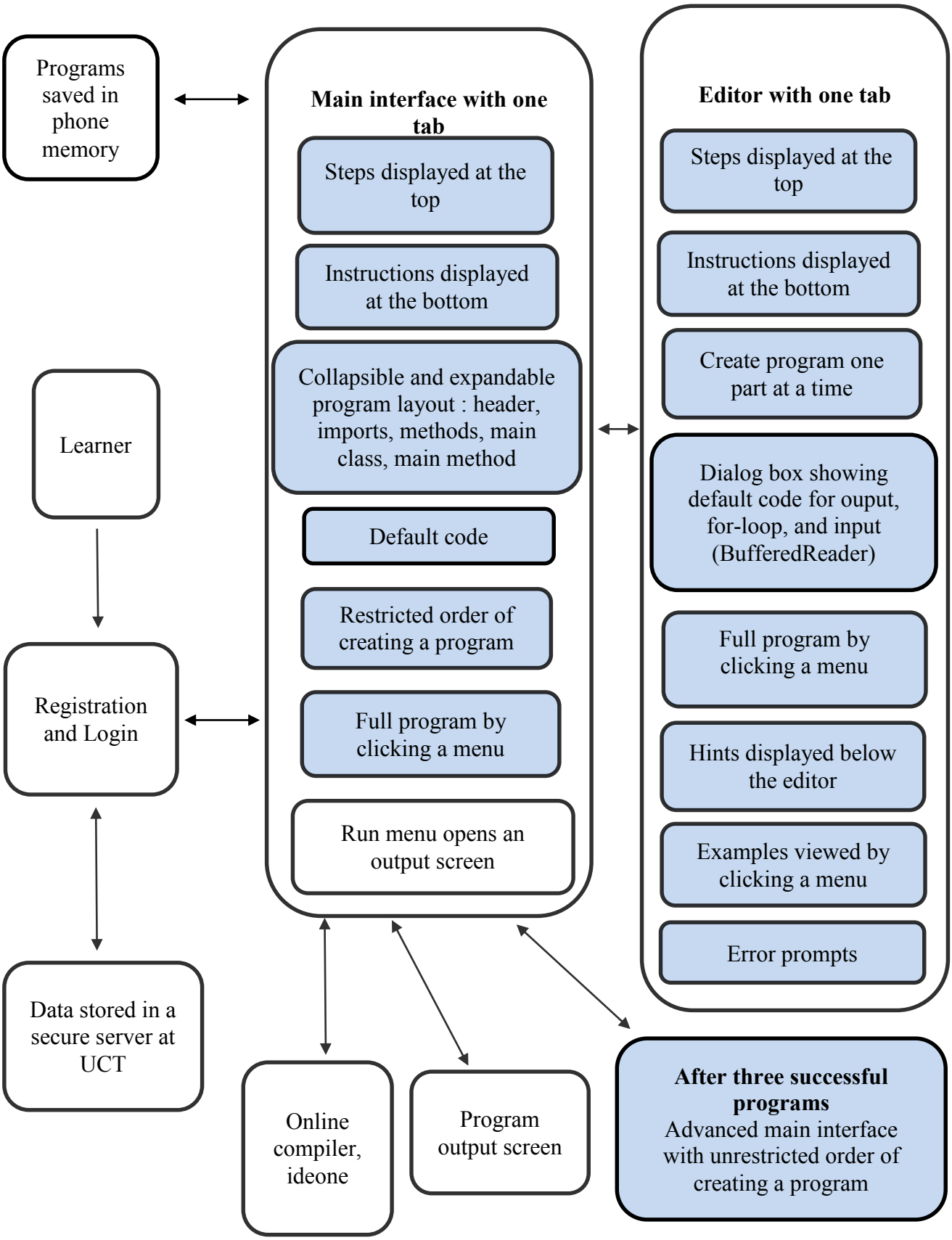


Figure 4.21: System overview of the first prototype showing the scaffolding techniques in blue at the main interface and at the editor

4.5.1.3 Editor interface

The editor shows steps and instructions. Steps are displayed at the top of the screen while instructions are displayed at the bottom of the screen. The instructions and steps are automatically displayed in the first two programs. In subsequent programs, these can be accessed through a menu. At the editor, each program is created only one part at a time. Default code is provided in the form of a statement dialog box that allows selection of three default statements: System.out.println(); for-loop; and BufferedReader. On selection of any of these default statements, the related default code is populated in the editor. Hints are displayed below the editor and are specific to the program part that is being created. Examples can be accessed by clicking on a menu, and are also specific to the program part that is being created. Error prompts are also part-specific and only pop up if a program part has a syntax error. The full program can be viewed by clicking on a menu. To go back to the main interface, the user presses on the phone's back button. This operation also automatically saves the program.

4.5.1.4 Program storage

The application saves the program in the phone's internal memory. These programs can be reopened by clicking on a related menu. This opens a screen with a list of all the saved programs. Upon clicking the required program, the user is asked if they want to load the program or to delete it. When the user clicks on the required program it is loaded back to the main interface. Upon reopening, the program is split into the program parts in order to correctly display it using the program layout. For example, if the main method was already edited in the saved program, the program layout should show the main method in green. Upon expanding the main method's button, the code underneath should display the last saved state of the program's main method.

4.5.1.5 The ideone online compiler

To compile and run the programs, this application used the free ideone online compiler (Sphere Research Labs 2010). This is because at the time of development of the application in this study, there was no free Java compiler that could be installed and run on a phone. Further, ideone had been used successfully by several mobile programming environments.

To use ideone, an online account was required in order to receive a unique username and API password that was to be used to link the application with the registered account. Thereafter, several methods were implemented to indicate the use of Java (ideone implements 60+ languages), and to send the code to the online server each time the run button was clicked. However, there were challenges experienced in using ideone that required the development of additional algorithms to suit this study. For example, the Web-based ideone interface requires that the input is typed at the console before the program is run. Therefore, this was expected even on a mobile programming environment. However,

IDEs such as JCreator or Eclipse, which learners use in the classroom, first run the program and then ask the user for input. This was the desired approach. In addition, there was a need to design an appropriate way to display the input message to the user and to fetch the input from the user on a mobile phone. Therefore, a solution was implemented to suit these requirements. As soon as the program was run, a dialog box with the appropriate message was displayed and the dialog box was used to fetch the user input. Another challenge was that ideone requires the class name of the main class to be 'Main'. Therefore, a solution was implemented that extracted all the class names and replaced them with 'Main' before the program was sent to the ideone compiler.

4.5.2 Second prototype

Figure 4.22 shows an overview of the second prototype showing some modifications from the first prototype. These modifications resulted from feedback from the first experiment. The details of the results that led to the modifications are discussed in Section 6.2.1. The registration, login, program storage, and use of ideone compiler are the same as in the first prototype. The modifications to the main interface and the editor are described next.

4.5.2.1 Modifications to the main interface

The main interface of the second prototype was modified to contain three tabs: one for instructions, one for the program layout, and one for the full program. In the first program, the instructions tab is automatically displayed. A user can then swipe to the required tab. A button for creating other classes was added to the program layout. Further, instead of accessing the run option via a menu, a quick-access run button was provided at the top of the screen. These modifications are shown in Figure 4.23.

4.5.2.2 Modifications to the editor

The editor of the second prototype was modified to contain three tabs: one for instructions, one for the editor, and one for the full program. A user can then swipe to the required tab. These modifications are shown in Figure 4.24. Further, a header dialog box is automatically provided in the first two programs to guide the creation of the header comments. Thereafter, the header dialog can be accessed via a related menu. The header dialog is a type of automatic scaffolding. The header dialog provides active exploration by supporting correct construction of the header comments. Active exploration is a characteristic of constructivism. Figure 4.25 shows the header dialog. In the second prototype the default main class code was disabled from being edited. Figure 4.26 shows that the default code 'public class' is locked from editing and the user needs to only create the classname. The main class keyword restriction can be disabled by the user via a menu. Figure 4.26 also shows the menus to access examples and hints. Lastly, the second prototype provided the use of the Scanner class for input.

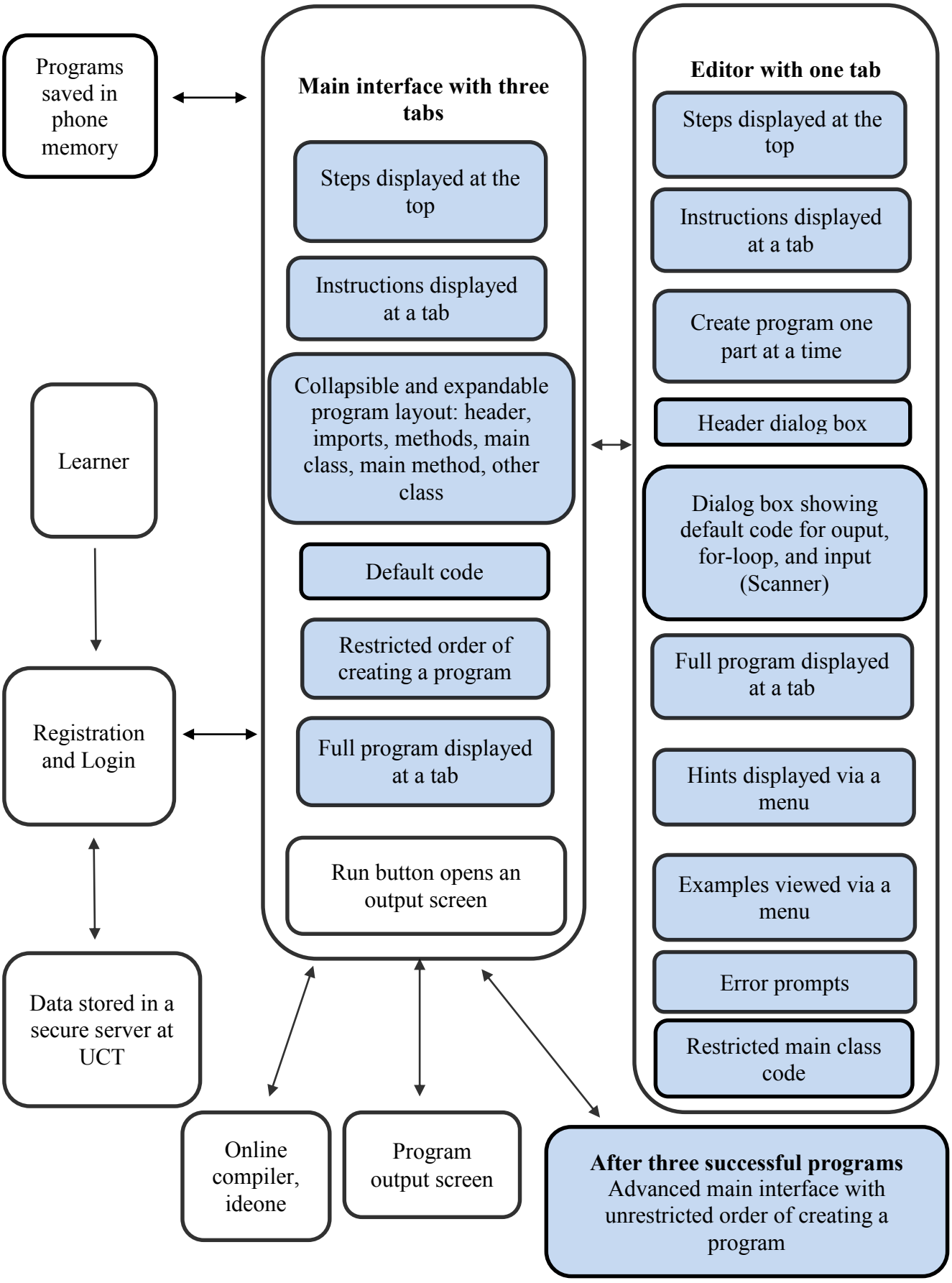


Figure 4.22: System overview of the second prototype showing the scaffolding techniques in blue at the main interface and the editor

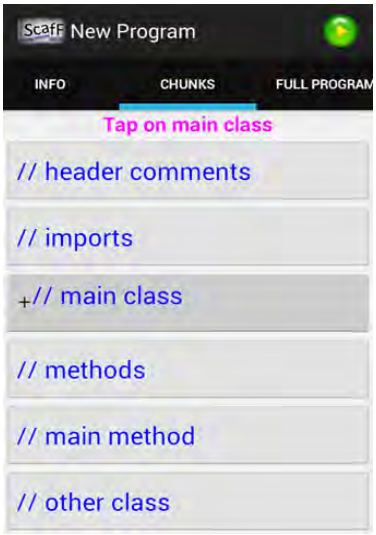


Figure 4.23: Screenshot showing the main interface of the second prototype with three tabs, a button for other class, and a quick-access run button

Figure 4.24: Editor with three tabs for instructions, editing and full program

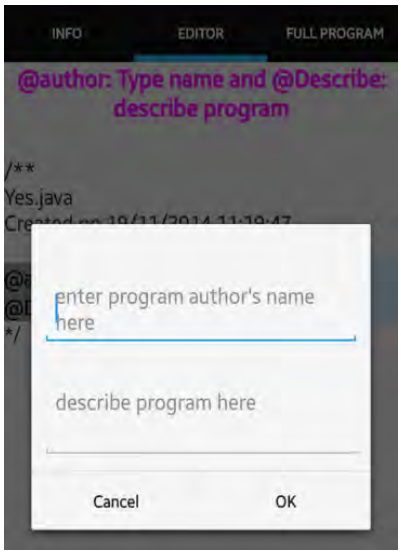


Figure 4.25: The header dialog in the editor

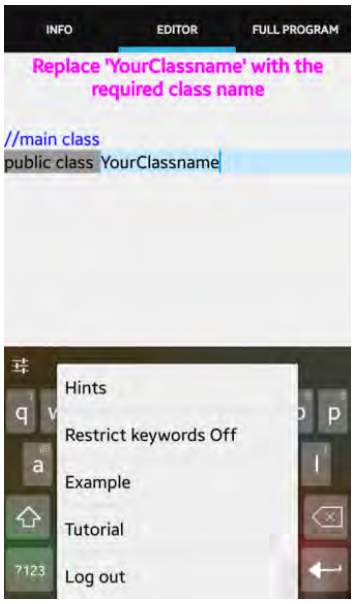


Figure 4.26: Restricted main class keywords at the editor

Using a simple example, the next section illustrates how the designed scaffolding techniques are used to write the program on the mobile programming environment.

4.6 Example of a Simple Program Created Using the Scaffolding Techniques

Problem: Write a program called 'Testing' that prints the words 'This works!'.

Upon successful login, the learner is presented with the main interface as shown in Figure 4.27. On start, the main class is the only one enabled. Figure 4.28 shows the main class clicked and steps are

shown at the top of the screen that instructs the learner on what to do next. On clicking inside the expanded area of the main class, the learner is taken to the code editing screen as shown in Figure 4.29, where the step at the top of the screen guides the user on what to do next. If the learner completes the class name starting with a lower case letter, an error prompt is displayed (Figure 4.30). On successful creation of class name and on pressing the phone back button, the main interface is displayed (Figure 4.31) and the program is saved onto device (Figure 4.32). The main class is highlighted in green to indicate completion and header comments part is now activated. The header comment shows the name of the program as created after creation of the main class (Figure 4.33). On selecting a menu to view full program, the full program is displayed as it was last saved (Figure 4.34). Figure 4.35 shows the code editor when the learner creates the header comment. On getting back to the main interface, the header comment is updated and main method is now activated (Figure 4.36).

On pressing the main method button, the default code for main method is revealed (Figure 4.36), and on pressing inside this expanded area the learner is shown some options to select (Figure 4.37). This problem requires display of output, hence the learner can select the System.out.println() option. This takes them to the code editor (Figure 4.38) and the learner can type what is required within the brackets of System.out.println(). On pressing the back button, the three completed sections are all green, as shown in Figure 4.39. The completed full program can now be viewed and seen as complete (Figure 4.40). To compile the program, the user selects the related menu after clicking on the phone menu button. On compilation, the output is shown in Figure 4.41.

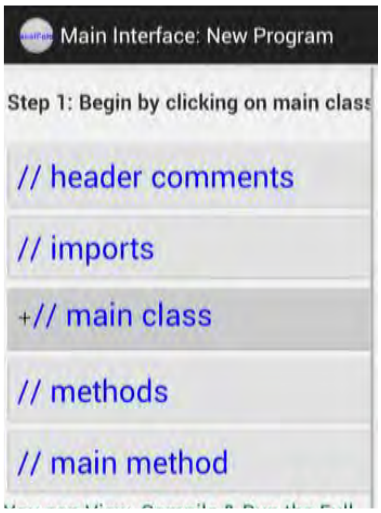


Figure 4.27: Main class active

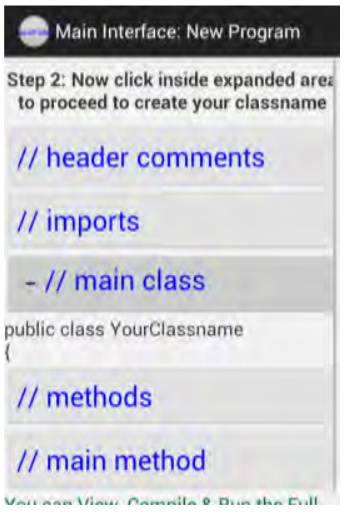


Figure 4.28: Main class clicked

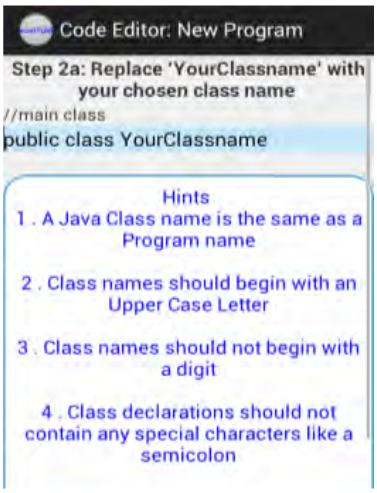


Figure 4.29: Editing main class

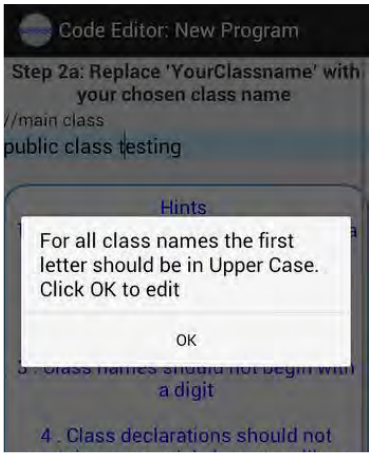


Figure 4.30: Error prompt

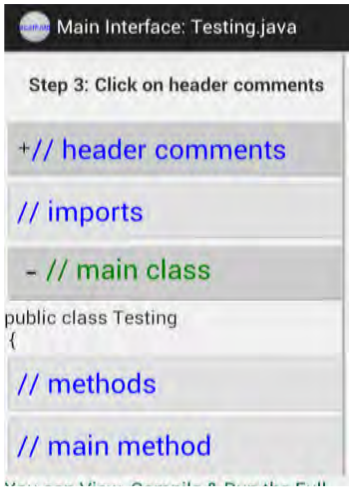


Figure 4.31: Header activated

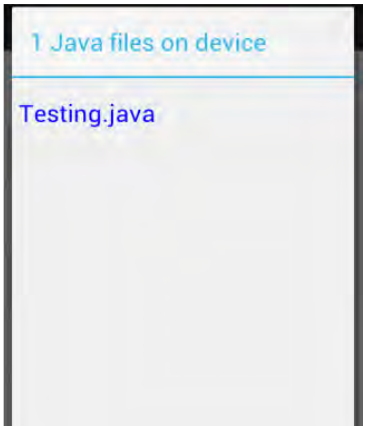


Figure 4.32: Saved on device

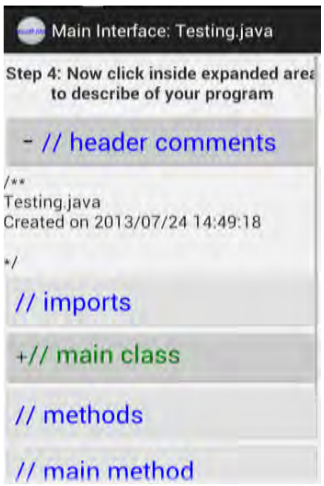


Figure 4.33: Header clicked

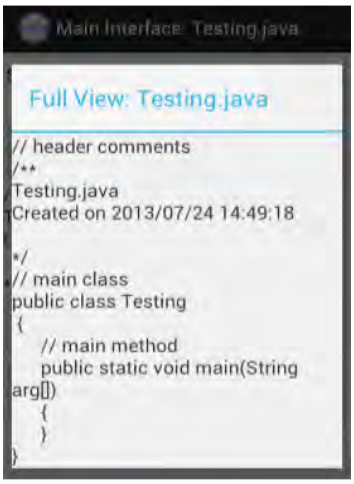


Figure 4.34: Full program

Figure 4.35: Creating header

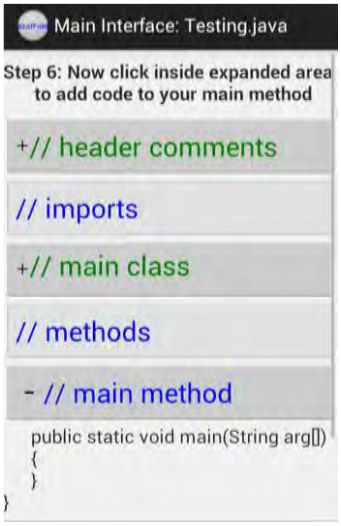


Figure 4.36: Main method clicked

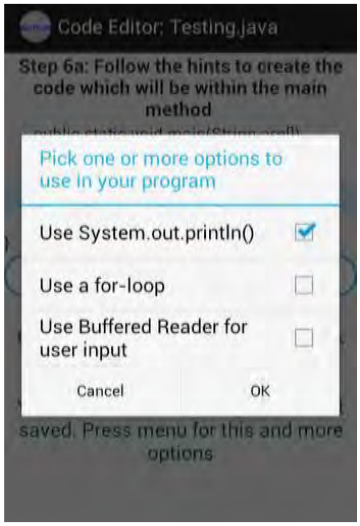


Figure 4.37: Default statements

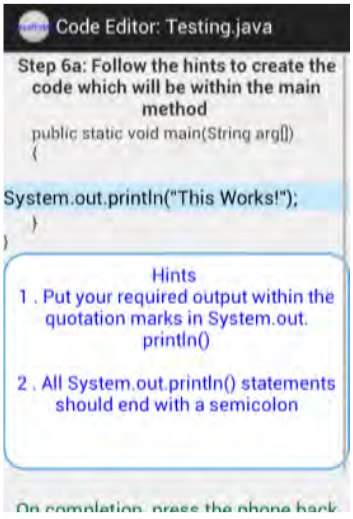


Figure 4.38: Edit main method

Figure 4.39: Completed program parts

Figure 4.40: Completed full program

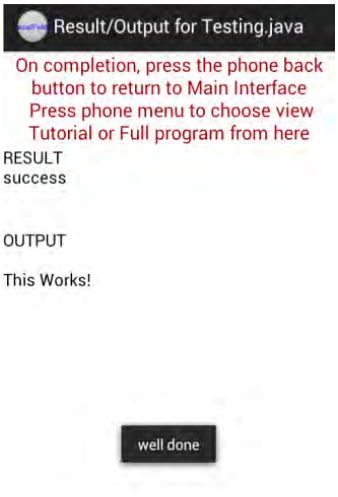


Figure 4.41: Output of program after compilation

4.7 Non-Scaffolded System Implementation

In order to make a comparison between the use of scaffolding techniques and use of a non-scaffolded mobile environment, a separate application was developed. This application had none of the scaffolding techniques that were designed in section 4.4. Figure 4.42 shows the resulting application, which had two interfaces, one showing instructions and the other where code could be typed. In order to maintain uniformity, this application was used for the sake of comparison with the scaffolded environment, as opposed to using one of the existing non-scaffolded mobile programming environments, such as SAND IDE. The non-scaffolded environment also used the ideone compiler for running and compiling programs.

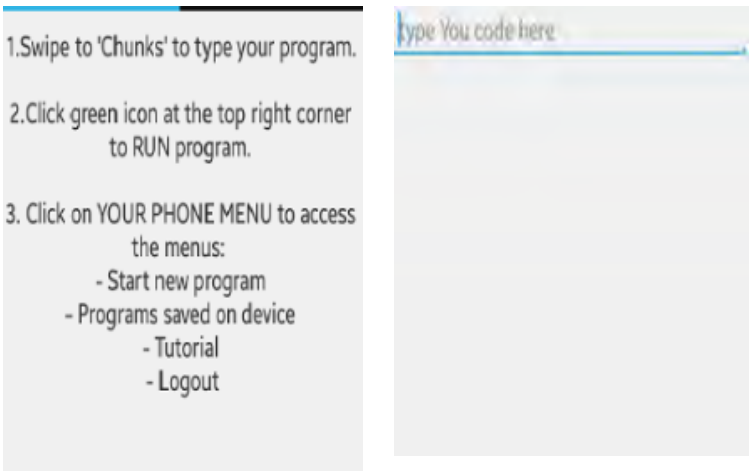


Figure 4.42: Interfaces for the non-scaffolded application

4.8 Chapter Summary

This chapter has illustrated how a six-level scaffolding framework has been used to select scaffolding techniques to address the learner challenges. The chapter has followed a learner-centered methodology where the learners' needs and limitations of mobile phones drove the choice of scaffolding techniques. Also, the chapter illustrated how the scaffolding techniques have been implemented on a mobile phone to scaffold the construction of Java programs. Therefore, this chapter has concretely shown a theoretic derivation of scaffolding techniques, and consequently their implementation on a mobile phone.

The use of the scaffolding framework has resulted in the choice of specific scaffolding techniques such as: providing a visual representation of a Java program by showing an overview of the program parts; enabling interaction with these parts using collapsible and expandable buttons and clickable parts; providing some default code; enabling completion of the program one part at a time while being able to view the full program; providing error prompts; providing hints and examples; and providing instructions and steps that support the use of the scaffolded environment. These scaffolding techniques were designed to address the three selected challenges cited by learners and also the small screen size and small keypad of mobile phones. Appendix A shows the other learner-cited challenges that were not illustrated in this chapter. A similar six-level approach was applied to these challenges and the process resulted in similar scaffolding techniques described in this chapter.

The chapter also presented an overview of the two prototypes and the use of ideone online compiler. Using an example, the chapter has shown how a simple program can be created using the scaffolding techniques. Finally, the chapter described a non-scaffolded environment that was designed by removing the scaffolding techniques. Using these prototypes, evaluation was conducted with learners of programming. The evaluation process is presented in the next chapter.

Chapter 5 Evaluation

The purpose of this study was to investigate which scaffolding techniques could support Java programming on a mobile phone, and to investigate the effect of using these scaffolding techniques to construct Java programs on a mobile phone. The design process led to the implementation of scaffolding techniques on a mobile programming environment (Mbogo et al. 2014; Mbogo et al. 2013), and a non-scaffolded mobile programming environment. To address the purpose of the study, learners of programming participated in experiments where they used the prototypes to construct Java programs.

Evaluation was conducted while learners constructed programs on the mobile environments. Evaluation models such as the CIAO model (Jones et al. 1999) have outlined that while evaluating educational technology one should consider data about learners' interaction with the software. Further, such evaluation is recognized in the micro and meso levels of the M3 evaluation framework, which examine individual activities of the technology users, and the learners' experience as a whole, respectively (Vavoula & Sharples 2009). In addition, a recommendation from both the CIAO and the M3 frameworks is consideration of learners' attitudes and outcomes. Learners' attitudes on the use of scaffolding techniques were measured by analyzing qualitative feedback. The outcomes were measured by analyzing log data from learners' interactions with the scaffolding techniques to construct programs.

In this chapter, the evaluation process used in this study is discussed by first describing the study participants. Thereafter, the data collection methods are described, followed by a discussion on the design of the experiments. The chapter concludes with a discussion on the criteria used to address the research questions and a summary of these criteria. The details of the experiments and the results are presented in Chapter 6.

5.1 Study Participants

Participants in the experiments were learners enrolled in an introduction to programming course taught using Java. Since the aim of the study was to support novice learners, such participants were deemed appropriate. Participation was voluntary. This means that learners participated in the study by choice and could withdraw at any time. In order to minimize the number of participants who would not turn up for the experiment sessions, two approaches were taken: (i) recruitment was conducted as close as possible to the time of the experiments; and (ii) participants were given incentives in the form of R50 per hour or provision of lunch.

Table 5.1 summarizes the number of learners who participated in the study per institution. A total of 182 learners from four universities participated in the study: 8 from University of Cape Town (UCT); 37 from University of Western Cape (UWC); 60 from Kenya Methodist University (KeMU); and 77 from Jomo Kenyatta University of Agriculture and Technology (JKUAT). These institutions were selected because there had been prior contact with the respective heads of departments and teachers of programming. The total number of learners depended on the availability of learners who volunteered to participate in the experiments and the total number of experiments that were carried out at that institution. For example, only one experiment session was conducted at UCT, while a total of three experiment sessions were conducted at KeMU. Table 5.1 shows the number of experiments that were conducted at each institution. The details of these experiments are discussed in Section 5.4.

Before conducting the experiments, ethical clearance was obtained from UCT (Appendix C1) and permission was sought to access learners from UCT (Appendix C2) and KeMU (Appendix C3). UWC and JKUAT recognized the ethical clearance from UCT and did not require a separate approval. The experiments took place at different times depending on the availability of learners of Java programming at the four institutions. For example, the programming course taught using Java at UWC was not offered during all the terms, so there was a need to wait until when such a course was offered. Further, the times also depended on the ability to travel to Kenya, for the Kenyan experiments. The experiments were conducted during these times: August 2013 at UCT and UWC; September 2013 at KeMU; June 2014 at UWC; July 2014 at KeMU and JKUAT; and October 2014 at KeMU and JKUAT.

Table 5.1: Total number of participants across the four institutions and the number of experiments conducted at each institution

Institution	Total number of Participants	Total number of experiment sessions	Experiment number	Total number of learners at each experiment
UCT	8	1	one	8
UWC	37	2	one	10
			two	27
KeMU	60	3	one	22
			two	14
			three	24
JKUAT	77	2	two	29
			three	48

5.2 Data Collection Methods

The research questions influenced the choice of the data collection methods. For example, in order to address the first research question, an analysis of the scaffolding techniques used to construct programs was required; this called for the use of computer logs. On the other hand, questionnaires were used to collect qualitative feedback from learners. The methods used were: electronic questionnaires; computer logs; and video and image recordings.

5.2.1 Electronic questionnaires

The electronic questionnaire method was used because it has the advantages of decreased cost, faster response times and increased response rates (Lazar & Preece 1999). Critical issues that must be addressed while using electronic questionnaires are: survey design, participant privacy and confidentiality, sampling and subject solicitation, distribution methods and response rates, and survey piloting (Andrews et al. 2010). These issues were addressed as follows:

i. LimeSurvey[8] was used to design the questionnaires. LimeSurvey is an open source online survey application. It is supported on multiple platforms and browsers, and automatically transferred the responses to a database that is hosted on a secure server at the department of Computer Science at UCT.
ii. The intent of the questionnaires was clearly outlined in the introduction, enabling well-informed participation and consent.
iii. The participants' privacy and confidentiality was ensured by not asking for personal information such as names or registration numbers.
iv. The respondents were learners in institutions of higher learning who had access to computers with Internet connections.
v. Questionnaires were activated on computers that were available in the rooms where the experiments took place.
vi. The first questionnaire was piloted with five learners at UCT.

Questionnaires were used throughout the study.

5.2.2 Computer logs

Computer logs can be used to yield information about learners' interaction with an application (Taylor 2006). Google Analytics[9] (GA) was used to collect data on learners' interactions with the mobile programming environment. GA is free and provides Application Programming Interface (API)

[8] http://www.limesurvey.org/

[9] http://www.google.co.za/analytics/

libraries that integrate easily with Eclipse and Android. Eclipse was used as the development environment to develop the application. The disadvantage of using GA is that it requires an Internet connection in order to send the data to the GA Web server. The experiments were conducted within the institutions' premises, where wireless connectivity was available. In cases where wireless connectivity was not available (as was the case at KeMU and JKUAT), participants were issued with airtime to cater for data costs. Computer logs were used after the first experiment.

5.2.3 Video and image recordings

The video and image recordings gave insight to some tacit information while learners interacted with the application. Not all participants' interactions were video recorded. Participants whose interactions were recorded on video were randomly selected. The video camera was close enough to capture the learners' interaction with the application, but not too close to interfere with the interaction. Video recordings were used only in the first experiment. Thereafter, computer logs were used to collect data on learners' interaction with the scaffolding techniques.

Pictures were taken while learners took part in the experiments. In the first experiment, computer logs were not used and so participants had to report the completion of each task. When a participant reported completion of a task, pictures were taken of the mobile application interface. In addition, pictures were taken of the groups of participant during all the experiments. The participants were asked for their consent before taking videos and pictures.

5.3 Internet-enabled mobile phones

The application was developed for the Android platform. Therefore, learners who did not own Android phones were issued with such phones during the experiment sessions. The majority of the phones issued were the Samsung Galaxy Pocket S5300 phones that run Android version 2.3 (Figure 5.1).

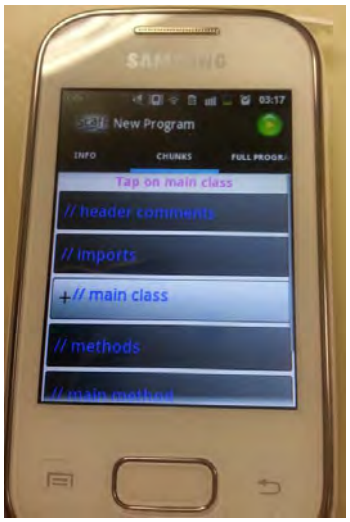


Figure 5.1: Samsung Galaxy Pocket S5300 used during the experiments

The Samsung Galaxy Pocket has a display size of 2.8 inches and contains 3GB of internal memory. The application was pre-installed on the phones and used during the experiments.

5.4 Experiment Design

Three experiments were conducted to address the two research questions. Table 5.2 shows a summary of which experiment was conducted at the four institutions, and a breakdown of the number of learners at each experiment. The number of experiments depended on the availability of learners, and the need to make stronger conclusions if an experiment indicated the need to collect more data.

In the first experiment, all the participants used the scaffolded environment and therefore only an experimental group was used. In the second and third experiments, the between-groups design was used, where participants were randomly split into the control and experimental groups and each group was exposed once to either the non-scaffolded environment (control) or the scaffolded environment (experimental). Participants in the control and experimental groups worked on the same programming tasks. The choice of the between-groups design countered any learning effect that would have occurred if learners were first exposed to the scaffolded environment and then to the non-scaffolded environment, using the same programming tasks. Table 5.2 shows which group was involved at each experiment.

The pre-test for this experiment was done by selecting learners who were at the same level of learning Java programming. The post-test was the collective measurements that compared performance between the control and experimental groups. In order to ensure non-contamination between the control and experimental groups, experiments were carried out at the same time, with the help of research assistants. Therefore, the second and third experiments were true experiments, where the features of a true experiment are (Cohen et al. 2007): (i) has one or more control groups; (ii) has one or more experimental groups; (iii) uses random allocation to control and experimental groups; (iv) contains pre-test of the groups to measure parity; (v) contains post-test of the groups to see the effect of the dependent variable; (vi) issues one or more interventions to the experimental group; (vii) observes isolation, control and manipulation of independent variables; and (viii) observes non-contamination between the control and experimental groups.

5.4.1 Programming tasks

During the entire study, five different sets of programming exercises were used: one set of similar exercises for the first experiments at UCT, UWC and KeMU; three different exercises for the second experiments at UWC, KeMU and JKUAT; and one set of similar exercises for the third experiments at KeMU and JKUAT.

Table 5.2: Number of experiments conducted at the four universities, the number of learners at each of the experiments, the groups involved in each experiment, and the data collection methods used at each experiment

Institution	Total number of Participants in all experiments	Total number of experiment sessions	Experiment number	Total number of learners at each experiment	Groups involved in the experiment	Data collection methods used
UCT	8	1	one	8	Experimental	Image/video recording Questionnaire
UWC	37	2	one	10	Experimental	
			two	27	Experimental and control	Image Logs Questionnaire
KeMU	60	3	one	22	Experimental	Image/video recording Questionnaire
			two	14	Experimental and control	Image Logs Questionnaire
			three	24		
JKUAT	77	2	two	29	Experimental and control	
			three	48		

In the first experiments, the exercises were similar because it was anticipated that results from the learners at KeMU (Kenya) would be different from results from the learners at UWC and UCT (South Africa) due to the different backgrounds. In the second experiments, the exercises were obtained from the different teachers of the courses in their respective institutions. In the third experiments, learners from both KeMU and JKUAT had covered similar topics in introduction to Java programming. Therefore, the exercises from the respective teachers were combined into one set. Despite the differences in the first, second and third sets of exercises, all the exercises covered introductory topics in Java. These tasks are presented in the next chapter that discusses the results.

5.4.2 Experiment procedure

At each experiment session, the procedure was as follows:

i. Participants were introduced to the purpose of the research and the experiment.

ii. Participants were guided through completion of the consent form (Appendix D).
iii. In the second and third experiments, participants were randomly divided into control and experimental groups.
iv. Participants were issued with Android phones containing the application.
v. Due to the use of the Internet for collecting computer logs and use of the ideone online compiler, participants were issued with airtime to cover data costs where there was no Wi-Fi.
vi. Participants were issued with printouts containing the programming tasks.
vii. During the experiment sessions, image/video and computer logs were used. Table 5.2 shows which data collection method was used in the different experiments.
viii. After the experiment sessions, participants were asked to fill out the online questionnaire.
ix. Participants returned the phones that were issued.

Following this experiment protocol, evaluation was conducted while learners interacted with the mobile programming environment. When considering data about learners' interaction, performance is evaluated because performance is all about what the user actually does in interacting with the product and consists of five types of metrics: task success; time-on-task; errors; efficiency; and learnability (Albert & Tullis 2008). A discussion follows on how these metrics and qualitative feedback were used as evaluation criteria in order to address each research question.

5.5 Criteria to Address the First Research Question

Which of the theoretically derived scaffolding techniques support programming on a mobile phone?

This research question led to four sub-questions:

i. Which scaffolding techniques were used to construct programs?
ii. How were scaffolding techniques used to construct programs?
iii. Which scaffolding techniques did learners find useful?
iv. What were the learners' experiences while using the scaffolding techniques?

Sub-questions (iii) and (iv) were subjective qualitative feedback.

5.5.1 Which scaffolding techniques were used to construct programs?

To address this sub-question, first, task success was measured by analyzing the level of completion of tasks. This means that each program was examined for the extent to which it was completed and if it produced the required output. A complete programming task is one that met all three criteria:

i. had all the required program parts completed;
ii. successfully compiled after completion of the required parts; and

iii. produced the required output.

Consequently, four metrics measured task success: (i) which tasks were attempted; (ii) which tasks were not attempted; (iii) which tasks were incomplete; and (iv) which tasks were completed. Incomplete tasks are tasks that failed to meet at least one of the criteria for completeness. Completed tasks met all the criteria for completeness. Attempted tasks are the combination of incomplete and completed tasks. Some tasks were not attempted. After measuring task success, analysis was conducted on which scaffolding techniques were used to construct the complete and incomplete programming tasks.

5.5.2 How were scaffolding techniques used to construct programs?

Measurement of how learners used scaffolding techniques involved an analysis of how scaffolding techniques were used to construct each program. This is called the "effects-with" evaluation (Quintana, Fretz, et al. 2000) and was defined as evaluation that looks at how learners work with the scaffolds in the software to do their work (Quintana et al. 2002a). Guided by the effects-with criteria, Table 5.3 shows a summary of criteria used to evaluate the scaffolding techniques designed in this study.

Initial use measured the first time a scaffolding technique was used. Reuse measured if a scaffold was used after its initial use. Therefore, use of a scaffolding technique was a measurement of both its initial use and reuse. Some scaffolds could be disabled automatically or by a user.

A faded out scaffold could be enabled (faded in) by a learner. Therefore, measurement was conducted on how the scaffolds were faded out and if they were faded in. Each attempted program was analyzed by following two steps: (i) extracting the sequence of steps that were followed to construct each program, in order to identify where a scaffolding technique was used; and (ii) where a scaffolding technique is used, evaluating it against the criteria in Table 5.3. Lastly, analysis was conducted on how scaffolding techniques were used differently over time (progression).

Table 5.3: Summary of criteria to evaluate use of scaffolding techniques

Criteria	Purpose
Use	Measurement of the initial use and reuse of the scaffolding technique.
Fading out	Measurement when a scaffold was disabled automatically or disabled by the learner.
Fading in	Measurement when a scaffold was enabled after fading.
Progression	How learners progressed through their work using scaffolding and whether they worked differently over time (Quintana, Fretz, et al. 2000).

5.5.3 Qualitative Feedback

Qualitative feedback was collected using self-reported data and by observing learners' experiences. Self-reported data was collected in two ways: (i) given a list of the scaffolding techniques, learners indicated the extent to which each feature supports the construction of programs on a mobile phone; and (ii) by learners reflectively indicating which scaffolding techniques they felt supports the construction of programs on a mobile phone. Learners' experiences were measured by recording learners' overall perceptions and observing their interaction with the scaffolded environment.

5.6 Criteria to Address the Second Research Question

What is the effect on learners of using the scaffolding techniques to construct Java programs on a mobile phone?

To address the second research question, learners were randomly divided into two groups: one group used a scaffolded mobile programming environment (experimental group); and the other group used a non-scaffolded mobile programming environment (control group). Therefore, the independent variable is the set of scaffolding techniques. The data from these two groups was analyzed to measure: task success; time-on-task; errors; and efficiency. Further, learnability was measured for only the experimental group in which learners used the scaffolded environment. Considering these metrics, this research led to sub-questions related to each metric. These sub-questions will be discussed in the relevant subsections.

The five metrics are the dependent variables. In manipulating the independent variable by providing some learners with a scaffolded environment and some learners with a non-scaffolded environment, the effects on the dependent variables were measured in order to test effectiveness of the scaffolding techniques.

The control and experimental groups were independent as each was subjected to one treatment (scaffolded or non-scaffolded environment). Therefore, the two-sample t-test was used to determine if the unknown means of the various metrics are different from each other (Elliott & Woodward 2007). In addition, t-tests are often used when only small samples are available ($n < 30$) (Harmon 2011). Since analysis was conducted per university, per experiment, the sample sizes in all the cases were less than 30.

5.6.1 Task Success

Following the definition in 5.5.1, task success was measured for all the attempted tasks in the experimental and control groups. This led to the first sub-question:

What is the effect of using the scaffolding techniques on task success?

To address this sub-question, task success results from the control group and the experimental group were compared. Some tasks could be attempted but not completed. Therefore, the hypotheses derived for task success for attempted but incomplete tasks were:

H_0: The mean number of attempted tasks in the experimental group is not larger than the mean number of attempted tasks in the control group.

H_1: The mean number of attempted tasks in the experimental group is larger than the mean number of attempted tasks in the control group.

Some tasks could be attempted and completed. Therefore, the hypotheses derived for task success for attempted and completed tasks were:

H_0: The mean number of completed tasks in the experimental group is not larger than the mean number of completed tasks in the control group.

H_1: The mean number of completed tasks in the experimental group is larger than the mean number of attempted tasks in the control group.

A one-tailed t-test was used to test these hypotheses.

5.6.2 Time-on-task

Time-on-task was the duration between the start and end of a program for both complete and incomplete programs. The end-time for complete programs referred to the first time the program compiled successfully and produced the desired output. The end-time for incomplete programs referred to the time the user quit working on the program. Data for time-on-task was measured by considering three criteria (Sauro & Lewis 2012): (i) task completion time for completed tasks; (ii) time until failure for incomplete tasks; (iii) and total time per user for both incomplete and completed tasks. Time-on-task was measured for all the attempted tasks in the experimental and control groups. Therefore, this led to the second sub-question:

What is the effect of using the scaffolding techniques on time-on-task?

To address this sub-question, time-on-task results between the control group and the experimental group were compared. Time-on-task represents either time on an incomplete task or time on a completed task. The hypotheses derived for time on completed tasks were:

H_0: The mean completion time in the experimental group is not less than the mean time on complete tasks in the control group.

H_1: The mean completion time in the experimental group is less than the mean time on complete tasks in the control group.

The hypotheses derived for time on incomplete tasks were:

H_0: The mean time on incomplete tasks in the experimental group is not less than the mean time on incomplete tasks in the control group.

H_1: The mean time on incomplete tasks in the experimental group is less than the mean time on incomplete tasks in the control group.

A one-tailed t-test was used to test these hypotheses.

5.6.3 Efficiency

The Common Industry Format for Usability Test Reports (NIST 2001) specifies efficiency as the ratio between task completion rate and the mean time-on-task. Task completion rate is the percentage of participants who completed each task. Mean time-on-task is the average time that was taken on each task. This calculation of efficiency specifies the percentage of users who were successful for every unit of time(NIST 2001). Such measurement of efficiency has been utilized in other studies such as one on the use of an adaptive user interface for service-oriented architectures (Senga 2010). Task completion rate and mean time-on-task was measured for all the attempted tasks in the experimental and control groups. This led to the third sub-question:

What is the effect of using the scaffolding techniques on the ratio between task completion rate and mean time-on-task?

To address this sub-question, task completion rates and mean time-on-task results between the control groups and the experimental groups were compared.

5.6.4 Errors

Two types of errors were evaluated: (i) the number of run-time errors for all the programs in the control and experimental groups; and (ii) errors that triggered scaffolding techniques that offered support for error detection, only for the experimental group. This led to the fourth sub-question:

What is the effect of using the scaffolding techniques on the number of errors?

To address this sub-question, the number of errors between the control groups and the experimental groups were compared. The hypotheses that were derived for errors were:

H0: The mean number of run-time errors encountered in the experimental group is not lower than the number of run-time errors encountered in the control group.

H1: The mean number of run-time errors encountered in the experimental group is lower than the number of run-time errors encountered in the control group.

A one-tailed t-test was used to test these hypotheses.

5.6.5 Learnability

The data from time-on task was used to evaluate learnability. A comparison was made between time-on-task from one task to the next. This analysis considered only the experimental group because the aim was to investigate the learnability of the scaffolded environment. This led to the fifth sub-question:

What is the effect of using the scaffolding techniques on time-on-task over time?

5.7 Summary of Criteria to Address Research Questions

Table 5.4 at the end of this chapter shows a combined overall picture of the number and distribution of the experiments, and the evaluation criteria for each experiment. Not all evaluation criteria were addressed in the first experiment. However, by the end of the third experiment, all metrics had been collectively measured. For example, the first experiment did not measure time-on-task, but the second and third experiments measured time-on-task (alongside all other metrics) in the experimental and the control groups. A summary of the two research questions, their sub-questions and the criteria that were derived to address them is presented next.

Which of the theoretically derived scaffolding techniques support construction of Java programs on a mobile phone?

To address this research question, the following sub-questions are posed:

i. Which scaffolding techniques were used to construct programs?
ii. How were scaffolding techniques used to construct programs?
iii. Which scaffolding techniques did learners find useful?
iv. What were the learners' experiences while using the scaffolding techniques?

To address these sub-questions, three metrics were measured: (i) task success; (ii) which scaffolding techniques were used to construct the complete and incomplete programming tasks; (iii) how the scaffolding techniques were used, considering their use, fading, and progression; and (iv) qualitative feedback considering ratings of the desirability of scaffolding techniques, learners' reflections on the use of scaffolding techniques and learners' experiences while using the scaffolding techniques.

What is the effect of using the scaffolding techniques to construct Java programs on a mobile phone?

To address this research question, the following sub-questions are posed:

i. What is the effect of using the scaffolding techniques on task success?
ii. What is the effect of using the scaffolding techniques on time-on-task?
iii. What is the effect of using the scaffolding techniques on the ratio between task completion rate and mean time-on-task?
iv. What is the effect of using the scaffolding techniques on the number of errors?

v. What is the effect of using the scaffolding techniques on time-on-task over time?

To address these sub-questions, five metrics were measured: (i) task success; (ii) time-on-task; (iii) ratio between task completion rate and mean-time-on task, which calculates the efficiency; (iv) errors; and (v) time-on-task over time, which calculates learnability.

5.8 Chapter Summary

182 learners from four universities participated in three experiments. Participation in the experiments was voluntary and learners signed consent forms prior to the start of each experiment session. Learners were issued with phones to use and they used the pre-installed application to complete programming tasks during the experiments. Questionnaires, computer logs and image and video recordings were used to collect data. The first experiment consisted of only experimental groups, and the last two experiments consisted of control groups and experimental groups. This chapter has discussed the evaluation criteria that were derived in order to address the two research questions: (i) which scaffolding techniques support Java programming on a mobile phone; and (ii) the effect of using the scaffolding techniques to construct Java programs on a mobile phone. These criteria are: task success; time-on-task; errors; efficiency; learnability; qualitative feedback; and the use of scaffolding techniques. The results obtained from collecting data are described in the next chapter.

Table 5.4: Table showing number of experiments, number of learners at each of the experiments, groups involved in each and the evaluation criteria addressed at each experiment

Institution	Total number of Participants	Total number of experiment sessions	Experiment number	Total number of learners at each experiment	Groups involved in the experiment	Data collection methods used	Evaluation criteria addressed
UCT	8	1	one	8	Experimental	Image/video recording Questionnaire	Task success Qualitative feedback
UWC	37	2	one	10	Experimental		
			two	27	Experimental and control	Image Logs Questionnaire	Task success, Time-on-task, Errors, Efficiency, Learnability, Qualitative feedback, use of scaffolding techniques
KeMU	60	3	one	22	Experimental	Image/video recording Questionnaire	Task success Qualitative feedback
			two	14	Experimental and control	Image Logs Questionnaire	Task success, Time-on-task, Errors, Efficiency, Learnability, Qualitative feedback, use of scaffolding techniques
			three	24	Experimental and control		
JKUAT	77	2	two	29	Experimental and control		
			three	48			

Chapter 6 Results and Discussion

Data was collected while learners interacted with the scaffolded and non-scaffolded mobile programming environments. This chapter discusses the results and analyses of these data as per the evaluation metrics used to address the research questions, namely task success, time-on-task, errors, efficiency, learnability, use of scaffolding techniques, and qualitative feedback. Appendix I contains the raw data from learners' verbatim feedback. First, the following section presents the participants who took part in the study and a review of how they participated in the experiments.

6.1 Participants and Experiments

182 learners from four institutions participated in three experiments: 111 learners in experimental groups; and 71 learners in control groups. There were more learners in the experimental groups because the first experiment did not have a control group. Table 6.1 shows the distribution of the learners in the experimental and control groups at each experiment, across the four participating institutions. Each experiment session involved an introduction to the purpose of the research with learners signing consent forms, learners tackling the programming tasks, and completion of a post-experiment questionnaire.

Table 6.1: Distribution of learners in the control and experimental groups across three experiments at four institutions

Experiment	Institution	Number of learners in experimental groups	Number of learners in control groups
one	UWC	10	-
	UCT	8	-
	KeMU	22	-
two	UWC	14	13
	KeMU	7	7
	JKUAT	13	16
three	KeMU	13	11
	JKUAT	24	24

6.1.1 First Experiment

40 learners participated in the first experiment: 8 from UCT; 10 from UWC; and 22 from KeMU. At UCT and UWC, 17 of the learners studied Computer Science and one learner studied Electrical and Computer Engineering; all were at Bachelors level. At UCT, the learners participated in three 1-hour

long experiment sessions in groups of three, two and three learners, respectively. At UWC, all 10 learners participated in a single experiment session during a 1-hour lunch break. Figure 6.1 shows some participants in the session at UCT. Figure 6.2 shows some participants in the session at UWC. Learners attempted the programming tasks in Figure 6.3 using the first prototype of the mobile application. At the end of the experiment, learners completed the questionnaire in Appendix E1. The questionnaire collected demographic information and user feedback.

The final session of the first experiment was conducted at KeMU with 22 learners. All the learners studied Computer Information Systems at Bachelors level. The 22 learners were taking a course in 'Introduction to Object Oriented Programming using Java, and participated in a two-week class session. Figure 6.4 shows some participants in the class session at KeMU. During the class sessions, learners were taught topics on Java syntax, Loops, Input and Output using Scanner and BufferedReader, and Classes. Learners were required to use only the scaffolded environment to complete programming exercises and not use any PC IDEs. At the end of the two-week class session, learners attempted the programming exercises in Figure 6.3. Thereafter, some of the learners completed the questionnaire in Appendix E1.

The first experiment at UCT, UWC and KeMU measured the number of tasks completed, learners' perceptions of using the scaffolding techniques to construct programs on a mobile phone, and general usability of the application.

Figure 6.1: First Experiment session at UCT

Figure 6.2: First Experiment session at UWC

1. a.) Write a program called Output that prints the words 'This is Java';
2. a.) Write a program that uses a for-loop to print the odd numbers from 1 to 20;
   b.) Write a program that calculates the area of a room that is 10 meters wide and 25 meters long; and
3. Write a program that requires input of your name and outputs it in the format 'Your name is X'.

Figure 6.3: Programming task attempted by learners in the first experiment

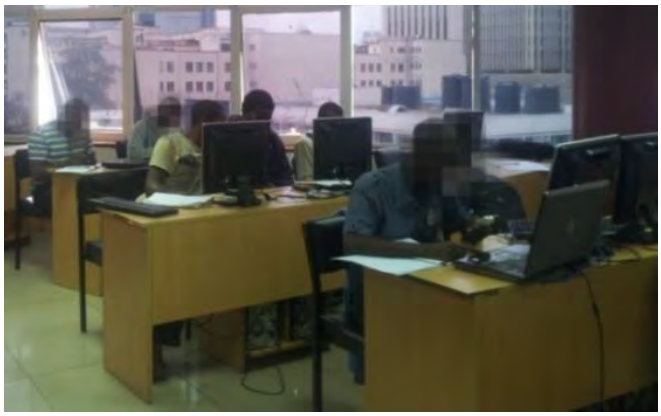


Figure 6.4: Class session at KeMU during the first experiment

6.1.2 Second Experiment

The second experiment was conducted at UWC, KeMU and JKUAT. 70 learners participated in the experiment: 34 in the experimental groups; and 36 in the control groups. The distribution of the number of learners in the three universities is as shown in Table 6.1. All the learners at UWC were studying towards a Postgraduate Diploma in Software Development. All the learners at KeMU were studying towards a Bachelor's degree in either Computer Information Systems or in Business Information Technology. All the learners at JKUAT were studying towards a Bachelor's degree in Information Technology. Figure 6.5 and Figure 6.6 show a section of some of the learners during the experiments at KeMU and UWC, respectively.

At KeMU and JKUAT, learners took part in 2-hour experiment sessions. At UWC, learners took part in a 1-hour experiment session. The difference in time was dependent on how long the groups of learners were available. The programming tasks attempted by learners in the three universities are shown in Figure 6.7. Learners used the second prototype of the application. At the end of the experiment, all the learners completed the questionnaire in Appendix E2, which collected demographic information and user feedback. This second experiment measured task success, time-on-task, errors, efficiency, learnability, use of scaffolding techniques, and learners' perceptions.

6.1.3 Third Experiment

The third experiment was conducted at KeMU and JKUAT with a total of 72 learners: 37 learners in the experimental groups; and 35 learners in the control groups. The distribution of the number of learners in the two universities is as shown in Table 6.1. All the learners at KeMU were studying towards a Bachelor's degree in either Computer Information Systems or in Business Information Technology. All the learners at JKUAT were studying towards a Bachelor's degree in Information Technology.

Figure 6.5: Second Experiment session at KeMU

Figure 6.6: Second Experiment session at UWC

Programming Task for UWC group in Experiment 2

1. Write a program that calculates the total cost of an item that is R159.72 and incurs a VAT of 14%.
2. Write a program that uses a for-loop to calculate the sum of the numbers from 1 to 50 and displays the sum and average.
3. Write a program that uses a method **name()** to print out your name.
4. Write a program that uses the Scanner input to ask for the user's name and age, and prints

   "Hello " + name " your age is "+ age;
5. Write a program that uses a method **input()** to ask for height and width of a rectangle, and calculate and display the area using height x width.
6. Write a program that determines if a number input by a user is odd or even.

Programming Task for KeMU group in Experiment 2

1. Write a program that initialises x to 10 and prints out its double value.
2. Use the appropriate control structures to print out the first 10 numbers.
3. Write a program that accepts two numbers as input and calculates the average.
4. Overload a method to print one and two integer values. Call these methods from the main method to output the number 34, and 12 and 24, respectively.
5. Write a program that creates a class that contains the constructor below:

   Item(int id, String title) { }

Programming Task for JKUAT group in Experiment 2

1. Write a program that output 'Scaffolding at JKUAT'.
2. Write a program that computes the sum and average of the number 1-20.
3. Write a program that captures and displays the ages of two students.
4. Write a program that uses a method to capture two integers and outputs their sum.
5. Write a program that initialises default values of name and age in a constructor and outputs these in a main class.

Figure 6.7: Programming tasks attempted by learners in the second Experiment at UWC, KeMU and JKUAT

Figure 6.8 and Figure 6.9 show a section of some of the learners during the experiments at KeMU and JKUAT, respectively. Learners took part in 2-hour experimental sessions at both KeMU and JKUAT. Figure 6.10 shows the programming tasks attempted by learners in this experiment. Learners used the second prototype of the application. At the end of the session, learners in the experimental groups completed the questionnaire in Appendix E2, while learners in the control groups completed the questionnaire in Appendix E3. These questionnaires collected demographic information and user feedback. This third experiment measured task success, time-on-task, errors, efficiency, learnability, use of scaffolding techniques, and learners' perceptions.

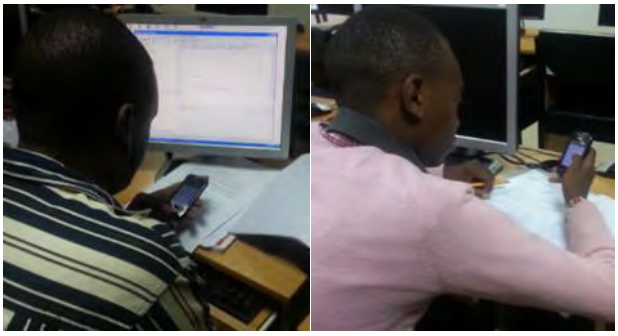


Figure 6.8: Third Experiment session at KeMU

Figure 6.9: Third Experiment session at JKUAT

1. Write a program that initialises x to 10 and prints out its double value. Save this program as XValue.java
2. Using a for-loop print the first 10 natural numbers. Save this program as Natural.java
3. Write a program that accepts input from the user and displays this as
   "Your input is " + input. Save this program as Natural.java
4. Write a program that uses a method **input()** to capture and display the names of two students. Save this program as MethSt.java
5. Write a program that creates two classes. The second class contains the constructor below. Access this constructor from the main class

```
Output()
{
        System.out.println("Constructor called");
}
```

   Save this program as Constructor.java

6. Write a program that uses a for-loop within a method avg() to calculate the sum of the numbers 20-100 and displays the sum. Call this method from the main method. Save this program as AvgMeth.java

Figure 6.10: Programming tasks attempted by learners in the third Experiment

6.2 Task Success

This section discusses the task success results from the three experiments, and highlights some of the issues that affected task completion based on observations and user feedback. For each experiment, results are presented first, followed by a discussion. In the first experiment, task completion was manually recorded and observations were made using video and image recordings. In the second and third experiments, computer logs were used to record task completion.

6.2.1 First Experiment

6.2.1.1 Results: Task Success

Table 6.2 shows the number of learners who completed each task at UCT, UWC and KeMU in the first experiment. At UWC and UCT, only two learners completed the third exercise that required the use of the BufferedReader class to accept user input. At KeMU, no learner completed the third task.

In the post-experiment questionnaire, learners were asked to indicate the extent to which they agreed that the scaffolding techniques support the construction of programs on the mobile phone. All the 18 learners at UCT and UWC completed this questionnaire. 10 learners at KeMU completed this questionnaire. Table 6.3 shows how learners at UWC and UCT rated the different scaffolding techniques and Table 6.4 shows how learners at KeMU rated the different scaffolding techniques. The last column shows a combined value of agree and strongly agree. Due to the small number of learners who completed the questionnaire at KeMU, Table 6.4 shows results in numbers and not percentages. The scaffolding techniques with the highest values in the last column were perceived to most effectively support constructions of programs on a mobile phone.

Presentation of the program in chunks received a high rating from learners at UCT, UWC and KeMU. Availability of hints had a high rating among UCT and UWC learners, with a slightly lower rating among KeMU learners. Despite some learners appreciating the error prompts and provision of default code, they both received lower ratings from both groups in comparison to the rest of the features. Steps, dialog prompts, examples and viewing of the full program had almost similar desirability preferences from both groups.

Table 6.2: Number of learners who completed each task at UCT, UWC and KeMU in the first experiment

Task	Learners who completed the tasks at UWC and UCT (out of 18)	Learners who completed the tasks at KeMU (out of 22)
1	18	14
2	12	16
3	2	0

Table 6.3: How UCT and UWC learners rated the different scaffolding techniques in terms of desirability to support construction of programs on a mobile phone

Scaffolding features	Strongly Disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree	Combination of Agree & Strongly Agree
Presentation in chunks	0%	12%	0%	63%	25%	88%
Completion part at a time	0%	13%	6%	38%	43%	81%
Steps to interact with application	0%	0%	12%	63%	25%	88%
Availability of hints	0%	0%	6%	38%	56%	94%
Error prompts	13%	12%	6%	44%	25%	69%
Dialog prompt of options e.g 'System.out.println()',	13%	0%	0%	31%	56%	87%
Provision of default code	0%	19%	19%	31%	31%	62%
Provision of examples	0%	6%	18%	38%	38%	76%
View of full program at any time	0%	0%	19%	38%	43%	81%

Table 6.4: How KeMU learners rated the different scaffolding techniques in terms of desirability to support construction of programs on a mobile phone

Scaffolding features	Strongly Disagree	Disagree	Neither agree Nor disagree	Agree	Strongly agree	Combination of Agree & Strongly Agree
Presentation in chunks	0	0	0	5	5	10
Completing a chunk at a time	0	0	1	5	4	9
Steps	0	0	2	3	5	8
Availability of hints	0	2	1	2	5	7
Error prompts	0	1	2	2	5	7
Dialog prompt of options	0	0	1	4	5	9
Provision of default code	1	0	2	2	5	7
Provision of examples	1	1	1	3	4	7
View full program at any time	1	0	0	3	6	9

6.2.1.2 Discussion: Task Success

Results from the first experiment indicate that learners could complete programming tasks using the scaffolding techniques. For the third task, learners at UCT and UWC indicated that they had not learnt the use of the BufferedReader but had learnt the use of the Scanner class for input. Therefore, this affected their ability to complete the third task. Similarly, learners at KeMU indicated that they would

prefer to use the Scanner class since they found the Scanner class simpler for input than the BufferedReader. The preference by learners to use the Scanner class over the BufferedReader indicates that the choice of the latter in the design was inappropriate. Further, this indicates that even while providing scaffolding techniques on mobile phones, it is important to keep the gap between what is learnt in the classroom and outside the classroom as small as possible, and this is encouraged by many learning theories that stress the principle of starting where the students are at (Carter 2010).

Some learners completed a full program within the main class chunk where only the class name is required (for example, in Figure 6.11). On pressing the back button to go back to the main interface, a prompt appeared indicating that the class declaration required only one line of code. On the other hand, some learners deleted the provided default code and then typed their own code from scratch, often leading to errors. Figure 6.12 shows inappropriate code within the main class written after deletion of the default code 'public class Yourclassname'. The learner was to replace only 'Yourclassname' with the required class name. These observations indicate that the application needed improvement to provide immediate prohibition on writing code that is not required for the given chunk. Further, additional scaffolding was required to prevent editing of default code, especially in the main class chunk.

Despite provision of a dialog box that provided some default statements to use within the main method and the method chunks (Figure 6.13), some learners opted to ignore the prompt and type the statements on their own. A commonly occurring instance was in preselecting 'System.out.println()' where the learner was required to write the output inside the 'println()' brackets. However, some learners opted out of the dialog box by pressing 'Cancel' and typed the statement from scratch (for example, in Figure 6.14). This sometimes led to incorrect completion of such code. This observation suggests that additional scaffolding was needed that provides an alert on how to re-enable the dialog box, should it be required.

The video recordings showed that learners hardly scrolled to view information that was not readily visible on the screen. Figure 6.15 shows a video screenshot of a learner at the main interface of the application. This learner continued to work on the visible interface and hardly scrolled up or down to view instructions that were below the last visible tab. In several instances, learners kept clicking on a non-active button, while the instruction on what to do next was at the bottom of the screen, which would have been visible upon scrolling downward. Further, a challenge observed was the soft keypad that covered nearly half of the screen while typing (Figure 6.16). This blocked some of the instructions and the hints that were placed on the lower half of the screen, and therefore some learners missed these. Indeed, feedback given by some of the learners stated that, 'The instructions

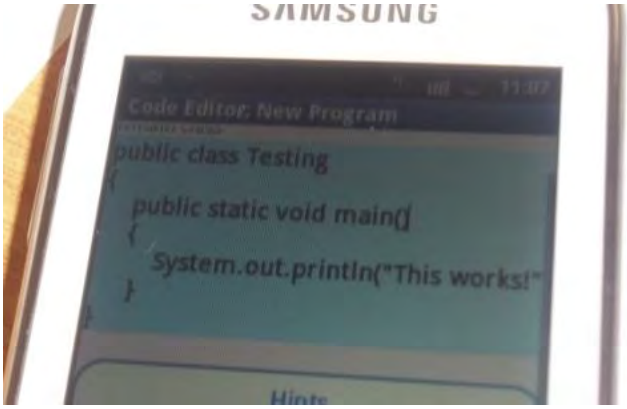


Figure 6.11: Full program written within the main class chunk where only the class name is required

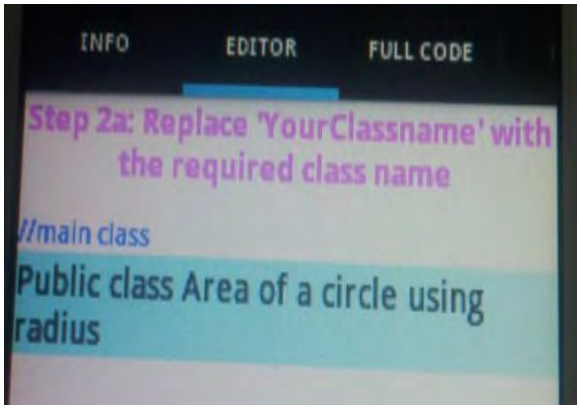


Figure 6.12: Inappropriate completion of the main class

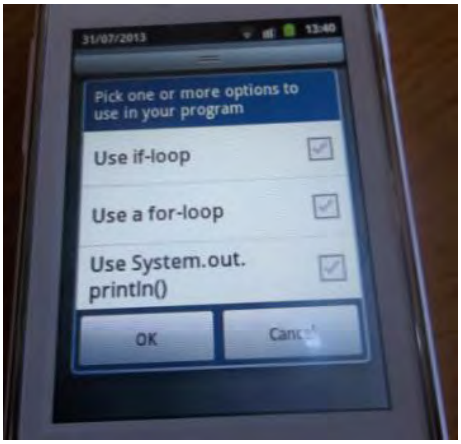


Figure 6.13: Dialog box showing default statements

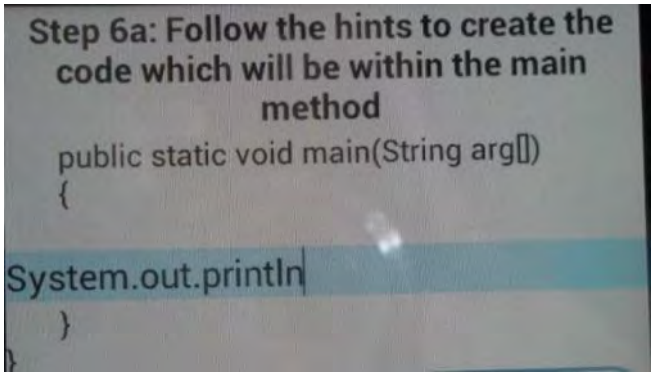


Figure 6.14: Learner typing statement from scratch

Figure 6.15: Video screenshot of a learner at the main interface of the application

Figure 6.16: Video screenshot showing soft keypad covering half the screen

were hidden and I didn't know where to look to get the next one. I suggest using a tabbed interface and not a list view.' This sentiment is supported by a study that suggested that scrolling can be reduced by placing navigational features in the fixed place near the top of a presented resource, and by placing key information at the top (Jones et al. 1999).

Learners rated the desirability of the scaffolding techniques to support construction of programs on a mobile phone. The highly rated scaffolding techniques are: (i) the program overview that presents a layout of the program; (ii) completing one chunk of a program at a time; (iii) the ability to view the full program while working on the individual chunks; (v) dialog prompts that provide default statements that can be reused; and (vi) steps that guide the user on how to interact with the application.

Despite the challenges that affected the completion of programs, the scaffolding techniques supported the construction of Java programs on a mobile phone. Indeed, below are some of the verbatim remarks by learners on their reflections on using the scaffolded environment to create programs:

'The main interface is simple and direct'

'Very easy to use especially with assistance of the hints and the examples'

'It simplifies the idea of programming as one does not have to keep on remembering the basic codes which are already in the program.'

'The fact that it has steps and guidelines. It's hard for a new user to have a hard time using it.'

Further, learners recommended that the link to run the program should be more accessible, as opposed to accessing it through a menu.

The feedback obtained from the first experiment was implemented on the first prototype. Appendix F shows screenshots of the second prototype with these modifications. The modifications were in three forms. First, to minimize text on the screen, several modifications were implemented: (i) separate tabs at the main interface that display instructions, the program overview and the full program; (ii) separate tabs at the editor to display instructions, the coding screen and the full program; (iii) a cancellable header dialog box for creating header comments with a related menu that could enable it; and (iv) links to hints and examples via a menu that opened these on separate screens. Second, some additions to the interface were implemented: (i) a run button was created at the top of the main interface for easy access; (ii) a button to enable creation of another class for programs that required more than one class; and (iii) one-time instructions to indicate that a chunk could be exited without being created. Third, some modifications to scaffolding techniques were implemented: (i) the main class' default text, 'public class', was disabled to prevent editing in the first program and enabled in the second program but could be enabled or disabled via a menu; and (ii) use of the Scanner class with a dialog box to enable user input. The modified prototype was used in the second and third experiments.

Although the findings from the first experiment were encouraging and useful, they contained certain limitations that required further research. Firstly, the programming exercises used were simple and therefore presented a limitation in the extent to which the application could be used to support tasks that were more difficult. Secondly, the number of participants in the evaluation was small and some key feedback could have been missed. Thirdly, since only an experimental group was used, the effect of using the scaffolding techniques to construct Java programs was not evaluated. Consequently, a second experiment was conducted.

6.2.2 Second Experiment

6.2.2.1 Results: Task success

Due to a technical challenge, the logs from KeMU's second experiment session were not recorded. However, the number of tasks that were completed was recorded manually and the learners completed the questionnaire at the end of the session. For this reason, KeMU's data for the second experiment was analyzed to measure only task success and qualitative feedback.

To recap, the hypotheses derived for attempted tasks were:

H_0: The mean number of attempted tasks in the experimental group is not larger than the mean number of attempted tasks in the control group. This is the first null hypothesis.

H_1: The mean number of attempted tasks in the experimental group is larger than the mean number of attempted tasks in the control group. This is the first alternate hypothesis.

The hypotheses derived for completed tasks were:

H_0: The mean number of completed tasks in the experimental group is not larger than the mean number of completed tasks in the control group. This is the second null hypothesis.

H_1: The mean number of completed tasks in the experimental group is larger than the mean number of attempted tasks in the control group. This is the second alternate hypothesis.

Table 6.5 shows the number of tasks attempted and completed in the experimental groups at KeMU, UWC and UCT. Table 6.6 shows the number of tasks attempted and completed in the control groups at KeMU, UWC and UCT. The raw data for the number of tasks attempted and completed per user at KeMU, UWC and JKUAT are shown in Appendices G1 to G3, respectively. An independent sample t-test was conducted to compare the number of attempted tasks for the experimental groups and the number of attempted tasks for the control groups. Similarly, an independent sample t-test was conducted to compare the number of completed tasks for the experimental groups and the number of completed tasks for the control groups. Table 6.7 shows the statistical results for attempted and completed tasks in the second experiment.

Table 6.5: Number of learners who attempted and completed each task in the Experimental groups at KeMU, UWC and JKUAT in the second Experiment

	KeMU		UWC		JKUAT	
	Attempted	**Completed**	**Attempted**	**Completed**	**Attempted**	**Completed**
Task 1	7	6	14	12	13	9
Task 2	7	4	10	7	10	5
Task 3	4	1	6	3	5	1
Task 4	0	0	2	0	2	1
Task 5	0	0	0	0	1	0

Table 6.6: Number of learners who attempted and completed each task in the Control groups at KeMU, UWC and JKUAT in the second Experiment

	KeMU		UWC		JKUAT	
	Attempted	**Completed**	**Attempted**	**Completed**	**Attempted**	**Completed**
Task 1	7	3	12	7	11	9
Task 2	7	1	6	2	14	11
Task 3	2	1	1	0	12	2
Task 4	0	0	1	0	2	1
Task 5	0	0	0	0	1	0

Table 6.7: Statistical task success results for attempted and completed tasks in the second Experiment

		Attempted Tasks		Completed Tasks	
Institution	**Statistical Metric**	**Experimental Group**	**Control Group**	**Experimental Group**	**Control Group**
KeMU	***M***	2.57	2.29	1.57	0.71
	SD	0.53	0.49	0.53	0.75
	t	t (12) = 1.04		t (11) = 2.44	
	p	**0.16**		**0.02**	
UWC	***M***	2.29	1.54	1.57	0.69
	SD	1.07	0.88	1.02	0.63
	t	t (25) = 1.99		t (22) = 2.72	
	p	**0.03**		**0.006**	
JKUAT	***M***	2.38	2.50	1.23	1.44
	SD	1.04	-0.33	1.17	0.89
	t	t (22) = 1.04		t (22) = 0.52	
	p	**0.37**		**0.30**	

At KeMU, there was no significant difference between the mean number of attempted tasks in the experimental group and the mean number of attempted tasks in the control group. With a *p*-value of 0.16, the first null hypothesis cannot be rejected. Therefore, the mean number of attempted tasks in the experimental group is not larger than the mean number of attempted tasks in the control group.

However, there was a significant difference between the mean number of completed tasks in the experimental group at KeMU and the mean number of completed tasks in the control. With a *p*-value of 0.02, the second null hypothesis is rejected in favour of the second alternate hypothesis. Therefore, the mean number of completed tasks in the experimental group is larger than the mean number of completed tasks in the control group.

The learners at KeMU were not able to attempt the last two tasks and they indicated that they struggled with topics of methods, classes and constructors in the classroom, considering that for most of them this was the first time to learn programming using Java.

At UWC, there was a significant difference between the mean number of attempted tasks in the experimental group and the mean number of attempted tasks in the control group. With a *p*-value of 0.03, the first null hypothesis is rejected in favour of the first alternate hypothesis. Therefore, the mean number of attempted tasks in the experimental group is larger than the mean number of attempted tasks in the control group.

Similarly, there was a significant difference between the mean number of completed tasks in the experimental group at UWC and the mean number of completed tasks in the control group. With a *p*-value of 0.006, the second null hypothesis is rejected in favour of the second alternate hypothesis. Therefore, the mean number of completed tasks in the experimental group is larger than the mean number of completed tasks in the control group. Further, some learners in the experimental group at UWC were able to complete the third task, while no learner in the control group was able to complete this task. Lastly, no learner was able to attempt the last program, perhaps due to the time constraint of the experiment session being in just 1 hour.

At JKUAT, there was no significant difference between the mean number of attempted tasks in the experimental group and the mean number of attempted tasks in the control group. With a *p*-value of 0.37, the first null hypothesis cannot be rejected. Therefore, the mean number of attempted tasks in the experimental group is not larger than the mean number of attempted tasks in the control group.

Similarly, there was no significant difference between the mean number of completed tasks in the experimental group and the mean number of completed tasks in the control group. With a *p*-value of 0.30, the second null hypothesis cannot be rejected. Therefore, the mean number of completed tasks in the experimental group is not larger than the mean number of completed tasks in the control group.

6.2.2.2 Discussion: Task Success in the second Experiment

Of the three experiment sessions at UWC, KeMU and JKUAT, one resulted in a significantly higher number of attempted tasks in the experimental group than in the control group, and two resulted in significantly higher number of completed tasks in the experimental groups than in the control groups. Further, some learners at UWC's experimental group were able to complete the third task while no learner in the control group completed the same task. These results indicate that the scaffolding techniques enabled completion of more programming tasks than the non-scaffolded environment.

A further analysis was conducted to understand the results at JKUAT. It was noted that learners in the control group accessed previously attempted programs that were stored on the mobile phone, and reloaded them to the interface to edit them. This could be because learners found it cumbersome to type each program from scratch on the small interface of the mobile phone. It could also be attributed to how leaners construct programs on a PC by copying old programs to the programming environment and editing them to suit a new program.

These results warranted further study where learners in both the control and experimental groups could write the programming tasks from scratch, and hence provide a uniform baseline for both groups. Further, in order to understand why learners were not able to attempt all tasks, the post-experiment questionnaire was redesigned to include a relevant question. In addition, since the results from KeMU were not used for the entire analysis, there was a need to conduct additional experiments in order to strengthen the conclusions. Consequently, a third experiment was conducted.

6.2.3 Third Experiment

6.2.3.1 Results: Task success

In this experiment, explicit instructions were issued to learners to write all programs from scratch. Examination of the logs revealed that learners in both groups followed this instruction, which eliminated the bias of one group simply editing previously completed programs.

Table 6.8 shows the number of learners who attempted and completed each task in the experimental groups at KeMU and JKUAT. Table 6.9 shows the number of learners who attempted and completed each task in the control groups at KeMU and JKUAT. The raw data for the number of tasks attempted and completed per user at KeMU and JKUAT are shown in Appendices G4 and G5, respectively. An independent sample t-test was conducted to compare the number of attempted tasks for the experimental groups and the number of attempted tasks for the control groups. Similarly, an independent sample t-test was conducted to compare the number of completed tasks for the experimental groups and the number of completed tasks for the control groups. Table 6.10 presents the statistical results for attempted and completed tasks in the third experiment.

Table 6.8: Number of learners who attempted and completed tasks in the Experimental groups at KeMU and JKUAT in the third Experiment

	KeMU		JKUAT	
	Attempted	**Completed**	**Attempted**	**Completed**
Task 1	13	9	24	18
Task 2	11	8	19	17
Task 3	7	5	20	12
Task 4	1	0	12	7
Task 5	0	0	6	3
Task 6	0	0	5	3

Table 6.9: Number of learners who attempted and completed each task in the Control groups at KeMU and JKUAT in the third Experiment

	KeMU		JKUAT	
	Attempted	**Completed**	**Attempted**	**Completed**
Task 1	11	3	24	9
Task 2	8	1	14	8
Task 3	1	0	11	4
Task 4	0	0	4	0
Task 5	0	0	2	0
Task 6	0	0	2	0

Table 6.10: Statistical task success results for attempted and completed tasks in the third Experiment

		Attempted Tasks		Completed tasks	
Institution	**Statistical Metric**	**Experimental Group**	**Control Group**	**Experimental Group**	**Control Group**
KeMU	***M***	2.46	1.82	1.69	0.36
	SD	0.97	0.60	1.03	0.50
	t	t (20) = 1.8		t (18) = 4.10	
	p	**0.03**		**0.0003**	
JKUAT	***M***	3.58	2.36	2.50	0.86
	SD	1.56	1.41	1.87	1.19
	t	t (46) = 2.82		t (39) = 3.59	
	p	**0.004**		**0.0004**	

At KeMU, there was a significant difference between the mean number of attempted tasks in the experimental group the mean number of attempted tasks in the control group. With a p-value of 0.03, the first null hypothesis is rejected in favor of the first alternate hypothesis. Therefore, the mean number of attempted tasks in the experimental group is larger than the mean number of attempted tasks in the control group. Similarly, there was a significant difference between the mean number of

completed tasks in the experimental group at KeMU and the mean number of completed tasks in the control group. With a *p*-value of 0.0003, the second null hypothesis is rejected in favor of the second alternate hypothesis. Therefore, the mean number of completed tasks in the experimental group is larger than the mean number of completed tasks in the control group.

At JKUAT, there was a significant difference between the mean number of attempted tasks in the experimental group and the mean number of attempted tasks in the control group. With a *p*-value of 0.004, the first null hypothesis is rejected in favor of the first alternate hypothesis. Therefore, the mean number of attempted tasks in the experimental group is larger than the mean number of attempted tasks in the control group.

Similarly, there was a significant difference between the mean number of completed tasks in the experimental group than in the control group. With a *p*-value of 0.0004, the second null hypothesis is rejected in favor of the second alternate hypothesis. Therefore, the mean number of completed tasks in the experimental group is larger than the mean number of attempted tasks in the control group.

6.2.3.2 Discussion: Task Success in the third Experiment

The two experiment sessions at KeMU and JKUAT both resulted in a significantly higher number of attempted tasks in the experimental group than in the control group. Similarly, both experiment sessions resulted in a significantly higher number of completed tasks in the experimental groups than in the control groups. The results from both KeMU and JKUAT indicate that the scaffolding techniques enabled learners to attempt and complete more programming tasks than the non-scaffolded environment.

At KeMU, only one learner from both groups was able to attempt any of the last three tasks. At JKUAT, fewer learners were able to attempt the last three tasks than the first three. At the end of the experiment session, learners were asked to indicate reasons why they could not attempt all the tasks. Collectively, the reasons that the learners gave are:

> 'time could not allow', 'the tasks were a bit challenging for me', 'I have very limited Java knowledge', 'I came late to the session so I had limited time to attempt all.'

These reasons indicate that with more time and with sufficient programming background, learners may be able to attempt, and perhaps complete, more programming tasks using the scaffolding techniques.

6.2.4 Summary of Task Success Results from all the Experiments

Learners in the first experiment were able to complete tasks using the scaffolding techniques. Learners indicated their most desirable scaffolding techniques as: the program overview that presents a program in chunks, completing one chunk of a program at a time, the ability to view the full program while working on the individual chunks, provision of steps that enable the user to interact with the

application, dialog prompts that provide default statements that can be reused, and steps that guide the user on how to interact with the application.

Learners' experiences and feedback indicated additional scaffolding techniques that could support programming on a mobile phone and meet learners' needs: (i) disabling of keywords in the first few programs; (ii) use of the Scanner class for input; and (iii) use of tabs, dialogs and menu links that open separate screens for additional scaffolds such as hints and examples. These were implemented on a second prototype.

Table 6.11 shows the consolidated statistical task success results from the second and third experiments. Of the five experiment sessions in the second and third experiments, three resulted in a significantly higher number of attempted tasks in the experimental groups than in the control groups, and four resulted in a significantly higher number of completed tasks in the experimental groups than in the control groups.

Collectively, the results for task success indicate that the theoretically-derived scaffolding techniques enable learners to attempt and complete more programming tasks on a mobile phone than when using a non-scaffolded environment.

Table 6.11: **Statistical task success results in the second and third Experiments for attempted and completed tasks in Experimental and Control groups**

		Second Experiment				**Third Experiment**			
		Attempted Tasks		**Completed Tasks**		**Attempted Tasks**		**Completed Tasks**	
Insti-tution	**Statistical Metric**	**Experi-mental**	**Control**	**Experi-mental**	**Control**	**Experi-mental**	**Control**	**Experi-mental**	**Control**
KeMU	***M***	2.57	2.29	1.57	0.71	2.46	1.82	1.69	0.36
	SD	0.53	0.49	0.53	0.75	0.97	0.60	1.03	0.50
	t	t (12) = 1.04		t (11) = 2.44		t (20) = 1.8		t (18) = 4.10	
	p	**0.16**		**0.02**		**0.03**		**0.0003**	
UWC	***M***	2.29	1.54	1.57	0.69				
	SD	1.07	0.88	1.02	0.63				
	t	t (25) = 1.99		t (22) = 2.72					
	p	**0.03**		**0.006**					
JKUAT	***M***	2.38	2.50	1.23	1.44	3.58	2.36	2.50	0.86
	SD	1.04	-0.33	1.17	0.89	1.56	1.41	1.87	1.19
	t	t (22) = 1.04		t (22) = 0.52		t (46) = 2.82		t (39) = 3.59	
	p	**0.37**		**0.30**		**0.004**		**0.0004**	

6.3 Time-on-task

Time-on-task was measured in the second and third experiments in four ways: (i) time on incomplete tasks; (ii) time on complete tasks; (iii) total time on tasks; and (iv) comparison of times on complete tasks from one task to another. For each experiment, results are presented first, followed by a discussion. In all the experiments, computer logs were used to record time-on-task.

To recap, the derived hypotheses for complete tasks were:

H_0: The mean completion time in the experimental group is not less than the mean completion time in the control group. This is the first null hypothesis.

H_1: The mean completion time in the experimental group is less than the mean completion time in the control group. This is the first alternate hypothesis.

The derived hypotheses for incomplete tasks were:

H_0: The mean time on incomplete tasks in the experimental group is not less than the mean time on incomplete tasks in the control group. This is the second null hypothesis.

H_1: The mean time on incomplete tasks in the experimental group is less than the mean time on incomplete tasks in the control group. This is the second alternate hypothesis.

An independent sample t-test was conducted to compare the completion time for the experimental groups and the completion time for the control groups. Similarly, an independent sample t-test was conducted to compare the time on incomplete tasks for the experimental groups and the time on incomplete tasks for the control groups.

6.3.1 Second Experiment

6.3.1.1 Results: Time-on-Task

Figure 6.17 shows the time-on-task distributions for experimental and control groups at UWC.

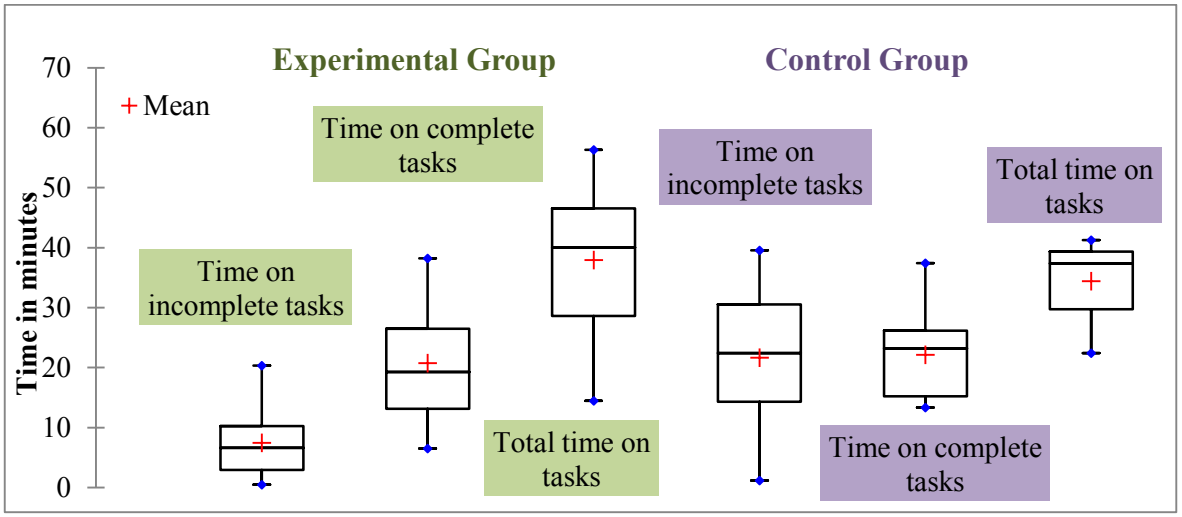


Figure 6.17: Box plots showing time-on-task for incomplete tasks, completed tasks and total time for Experimental and Control group at UWC, Experiment 2

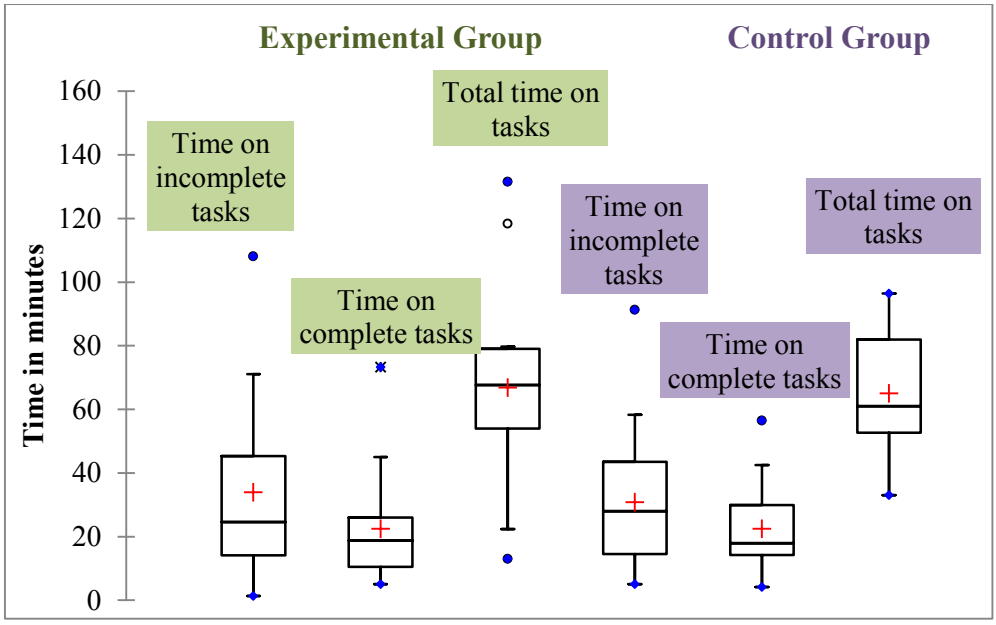


Figure 6.18: Box plots showing time-on-task distribution for incomplete tasks, completed tasks and total time on task for Experimental and Control groups at JKUAT, Experiment 2

Table 6.12: Statistical time-on-task results for all complete and incomplete tasks in the second Experiment

		Completed tasks		Incomplete tasks	
Institution	**Statistical Metric**	**Experimental**	**Control**	**Experimental**	**Control**
UWC	***M***	20.76	22.18	7.51	21.70
	SD	9.99	8.05	6.34	12.74
	t	$t(18) = 0.41$		$t(15) = -3.27$	
	p	**0.34**		**0.003**	
JKUAT	***M***	22.46	22.44	34.00	30.86
	SD	17.77	13.00	28.27	21.74
	t	$t(26) = 0.004$		$t(26) = 0.34$	
	p	**0.49**		**0.37**	

The Raw data for UWC is in Appendix H1. Figure 6.18 shows the time-on-task distributions for experimental and control groups at JKUAT (Raw data in Appendix H2). Table 6.12 shows the statistical results for all complete and incomplete tasks in the second experiment.

There was no significant difference in mean completion time between the experimental group and the control group at UWC. With a *p*-value of 0.34, the first null hypothesis cannot be rejected. Therefore, the mean completion time in the experimental group is not less than the mean completion

time in the control group. There was a significant difference between the mean time on incomplete tasks in the experimental group at UWC and the mean time on incomplete tasks in the control group. With a *p*-value of 0.003, the second null hypothesis is rejected in favor of the second alternate hypothesis. Therefore, the mean time on incomplete tasks in the experimental group is less than the mean time on incomplete tasks in the control group.

There was no significant difference in mean completion time between the experimental group and the mean completion time in control group at JKUAT. With a *p*-value of 0.49, the first null hypothesis cannot be rejected. Therefore, the mean completion time in the experimental group is not less than the mean completion time in the control group.

The experimental group at JKUAT had an outlier who completed tasks in longer times than normal. To determine whether the outlier influenced results for completion rates, the analysis was conducted twice, with the outlier and without the outlier. Both analyses concluded that there was no significant difference in in mean time on completed tasks between the experimental and control groups. Both *p*-values were above a significance level of 0.05 ($p = 0.49$ with outliers and $p = 0.32$ without outliers).

There was no significant difference between the mean time on incomplete tasks in the experimental group at JKUAT and the mean time on incomplete tasks in the control group. With a *p*-value of 0.37, the second null hypothesis cannot be rejected. Therefore, the mean time on incomplete tasks in the experimental group is not less than the mean time on incomplete tasks in the control group.

Figure 6.19 shows the time-on-task for each of the completed tasks in the experimental and control groups at UWC. These three tasks are considered because they were the ones completed by more than one learner in either of the groups. Figure 6.20 shows the time distributions for the first two

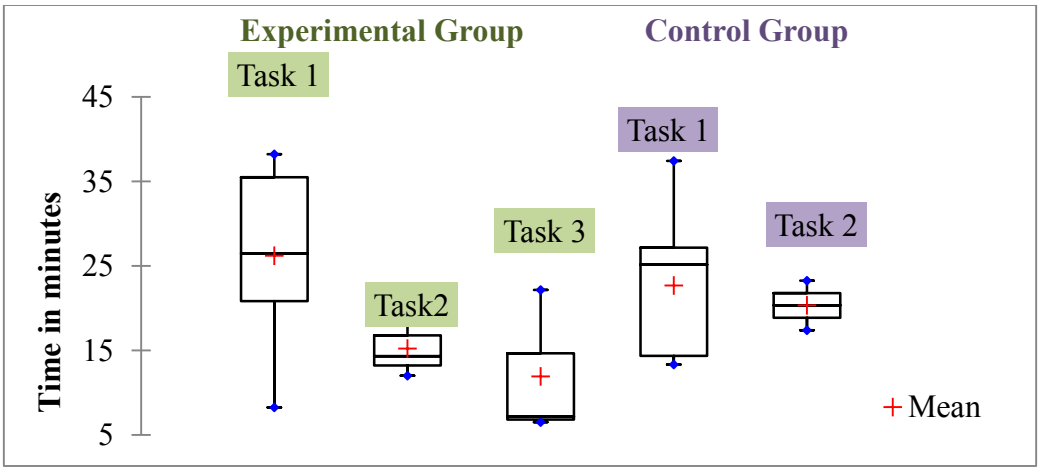


Figure 6.19: Box plot showing time on completed tasks per-task in the Experimental and Control group at UWC, Experiment 2

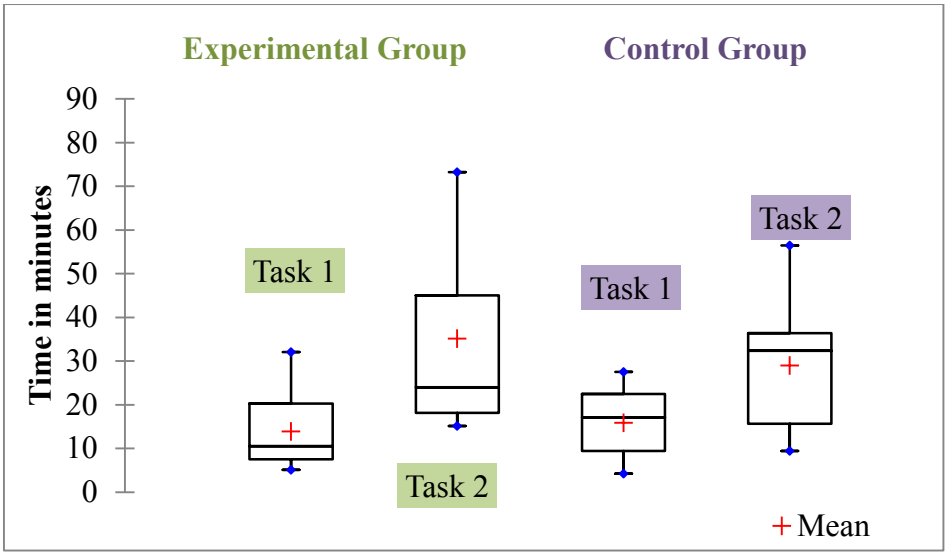


Figure 6.20: Box plot showing time on completed tasks per-task for Experimental and Control group at JKUAT, Experiment 2

Table 6.13: Statistical time-on-task results per completed task in the second Experiment

		Task 1		Task 2		Experimental		Control	
Institution	**Statistical Metric**	**Experimental**	**Control**	**Experimental**	**Control**	**Task 1**	**Task 2**	**Task 1**	**Task 2**
UWC	***M***	26.2	22.71	15.61	20.33	26.2	15.61	22.71	20.33
	SD	9.90	9.07	2.99	4.12	9.90	2.99	9.07	4.12
	t	$t(14) = 0.78$		$t(1) = -1.63$		$t\ (14) = 3.57$		$t\ (4) = 0.53$	
	p	**0.22**		**0.17**		**0.002**		**0.31**	
JKUAT	***M***	13.92	15.86	35.11	28.96	13.92	35.11	15.86	28.96
	SD	8.90	8.25	24.31	14.89	8.90	24.31	8.25	14.89
	t	$t(16) = -0.48$		$t(5) = 0.52$		$t\ (5) = -1.88$		$t\ (16) = -2.49$	
	p	**0.32**		**0.31**		**0.06**		**0.01**	

completed tasks in the experimental and control groups at JKUAT. These two tasks are considered because they were the ones completed by more than one learner in both groups. Table 6.13 shows the statistical results per completed task at UWC and JKUAT. In this table, the first two tasks are considered because they were the ones completed in both the control and experimental groups at UWC and JKUAT.

There was no significant difference in mean completion time for the first two tasks in both groups at UWC. For example, there was no significant difference in mean completion time for the first task in the experimental group and the first task in the control group. Similarly, there was no significant difference in mean completion time for the second task in the experimental group and the second task

in the control group. With both *p*-values > 0.05, the first null hypothesis cannot be rejected. Therefore, the mean completion time per task in the experimental group is not less than the mean completion time per task in the control group.

At UWC, learners in the experimental group spent a significantly shorter time on the second task than the first task. For example, there was a significant difference in mean completion time on the second task ($M = 15.61$, $SD = 2.99$) in comparison to the first task ($M = 26.2$, $SD = 9.90$), $t\ (14) = 3.57$, $p = 0.002$. On the other hand, the control group showed a non-significant difference in mean completion time in the second task ($M = 20.33$, $SD = 4.12$) in comparison to the first task ($M = 22.71$, $SD = 9.07$), $t\ (4) = 0.53$, $p = 0.31$. Therefore, the mean completion time for subsequent tasks after the first in the experimental group is less than the mean completion time for subsequent tasks after the first in the control group.

At JKUAT, there was no significant difference in the mean completion time for the first two tasks in both groups. For example, there was no significant difference in mean completion time for the first task between the experimental group and the first task in the control group. Similarly, there was no significant difference in mean completion time for the second task in the experimental group and the second task in the control group. With both *p*-values > 0.05, the first null hypothesis cannot be rejected. Therefore, the mean completion time per task in the experimental group is not less than the mean completion time per task in the control group.

At JKUAT, there was no significant difference between the mean completion time in the first task in the experimental group ($M = 13.92$, $SD = 8.90$) and the mean completion time in the second task in the experimental group ($M = 35.11$, $SD = 24.31$), $t\ (5) = -1.88$, $p = 0.06$. On the other hand, there was a significant difference between the mean completion time in the first task in the control group ($M = 15.86$, $SD = 8.25$) and the mean completion time in the second task in the control group ($M = 28.96$, $SD = 14.89$) $t\ (16) = -2.49$, $p = 0.01$. Therefore, the mean completion time for subsequent tasks after the first in the experimental group is not less than the mean completion time for subsequent tasks after the first in the control group.

6.3.1.2 Discussion: Time-on-Task in the second Experiment

Results from UWC and JKUAT indicate that the mean completion time in the experimental group is not less than the mean completion time in the control group. This is supported by results that indicate that the mean completion time per task in the experimental group is not less than the mean completion time per task in the control group. This shows that the scaffolding techniques did not enable faster completion times than the non-scaffolded environment. Further, as reported in the results for task success for JKUAT, the learners in the control group edited previously completed programs as opposed

to starting programs from scratch. This shows that for the second experiment, learners in the control group had an advantage over learners in the experimental group.

Results from UWC indicate that the mean time on incomplete tasks in the experimental group is less than the mean time on incomplete tasks in the control group. This shows that learners using the scaffolding techniques were able to reach failure states quicker and could move on to other tasks, as opposed to learners in the control group who spent longer on unsuccessful tasks. However, results from JKUAT indicate that the mean time on incomplete tasks in the experimental group is not less than the mean time on incomplete tasks in the control group. This shows that the scaffolding techniques did not enable learners to reach failure states quicker than the non-scaffolded environment.

Lastly, results from UWC indicate that learners in the experimental group spent significantly shorter times in subsequent tasks after the first task. In comparison, learners in the control group did not show this trend. This indicates the learnability of the scaffolded environment. However, results from JKUAT indicate that there was no significant difference between the mean completion time in the first task in the experimental group and subsequent tasks. On the other hand, learners in the control group took a significantly longer time on the second task than on the first task. This shows that the scaffolding techniques did not enable faster completion times in subsequent tasks after the first.

6.3.2 Third Experiment

6.3.2.1 Results: Time-on-Task

Figure 6.21 shows the time-on-task distributions for experimental and control groups at KeMU including all incomplete tasks (Raw data in Appendix H3). Figure 6.22 shows the time-on-task

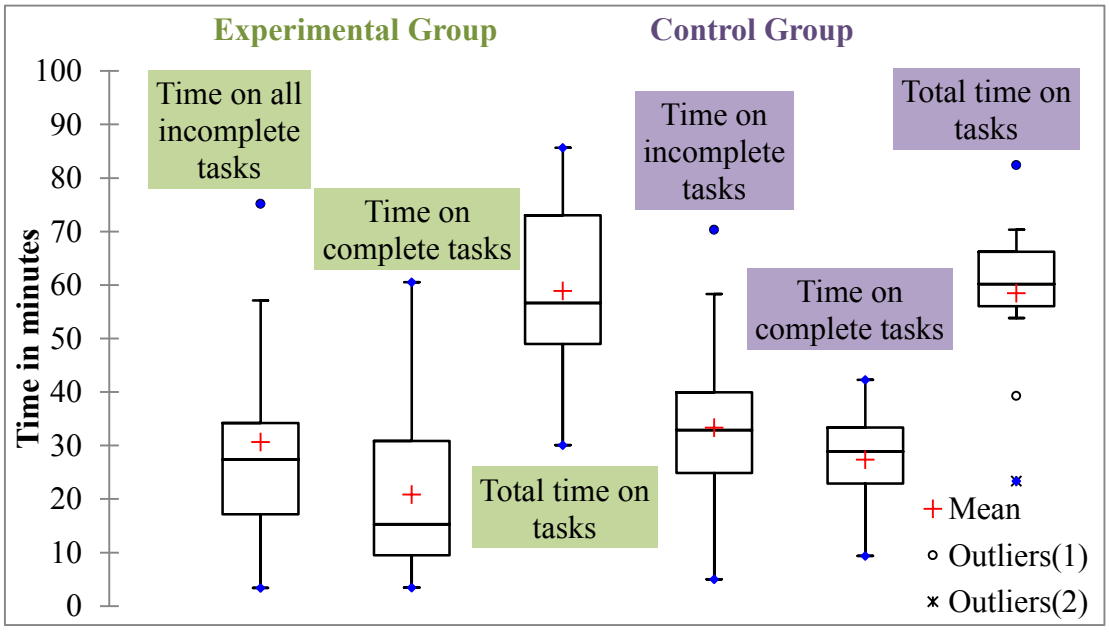


Figure 6.21: Box plots showing time-on-task distribution for incomplete tasks, completed tasks and total time on task for Experimental and Control groups at KeMU, Experiment 3

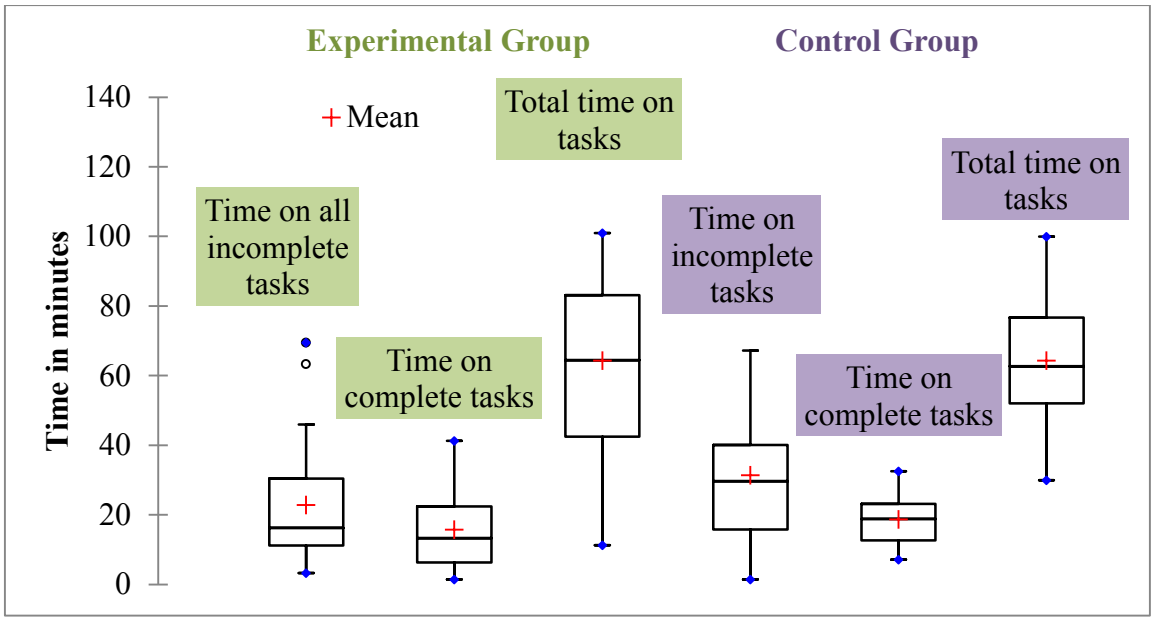


Figure 6.22: Box plots showing time-on-task distribution for all incomplete tasks, completed tasks and total time on task for Experimental and Control groups of JKUAT, Experiment 3

Table 6.14: Statistical time-on-task results for all complete and incomplete tasks in the third Experiment

		Completed tasks		Incomplete tasks	
Institution	**Statistical Metric**	**Experimental**	**Control**	**Experimental**	**Control**
KeMU	***M***	20.88	27.36	30.65	33.39
	SD	15.01	13.59	21.41	16.77
	t	t (4) = 0.86		t (16) = -3.44	
	p	**0.22**		**0.37**	
JKUAT	***M***	15.82	18.75	22.84	31.39
	SD	11.15	7.51	17.66	19.92
	t	t (52) = 1.34		t (57) = -1.78	
	p	**0.09**		**0.04**	

distributions for experimental and control groups at JKUAT, including all incomplete tasks (Raw data in Appendix H4). Table 6.14 shows the statistical results for all complete and incomplete tasks in the third experiment.

There was no significant difference in mean completion time between the experimental group at KeMU and the control group. With a p-value of 0.22, the first null hypothesis cannot be rejected.

Therefore, the mean completion time in the experimental group is not less than the mean completion time in the control group.

There was no significant difference between the mean time on incomplete tasks in the experimental group at KeMU and the mean time on incomplete tasks in the control group. With a *p*-value of 0.37, the second null hypothesis cannot be rejected. Therefore, the mean time on incomplete tasks in the experimental group is not faster than the mean time on incomplete tasks in the control group.

There were two kinds of incomplete tasks: those that had all parts completed but contained errors; and those that had only some parts completed. An additional analysis was conducted on the data from KeMU to examine if there was a significant difference on time of incomplete programs between the two types. Both analyses concluded that there was no significant difference in mean time on incomplete tasks between the experimental and control groups for the two types of incomplete programs.

There was no significant difference in mean completion time at JKUAT between the experimental group and the control group. With a *p*-value of 0.09, the first null hypothesis cannot be rejected. Therefore, the mean completion time in the experimental group is not faster than the mean completion time in the control group.

There was a significant difference between the mean time on all incomplete tasks in the experimental group at JKUAT and the mean time on all incomplete tasks in the control group. With a *p*-value of 0.04, the second null hypothesis is rejected in favor of the second alternate hypothesis. Therefore, the mean time on incomplete tasks in the experimental group is faster than the mean time on incomplete tasks in the control group. A further analysis was conducted with only full incomplete tasks on the data from JKUAT. The analysis concluded that there was also no significant difference in mean time on incomplete tasks between the experimental and control groups for the full incomplete programs.

Figure 6.23 shows the time-on-task for each of the completed tasks in the experimental and control groups at KeMU. Figure 6.24 shows the time-on-task for each of the completed tasks in the experimental and control groups at JKUAT. Table 6.15 shows the statistical results per completed task at KeMU and JKUAT in the third experiment.

There was no significant difference between the mean completion time for the first task in the experimental group at KeMU and the mean completion time for first task in the control group. With a *p*-value of 0.40, the first null hypothesis cannot be rejected. Because only one learner completed the second task in the control group at KeMU, no further statistical analysis could be performed on the second task.

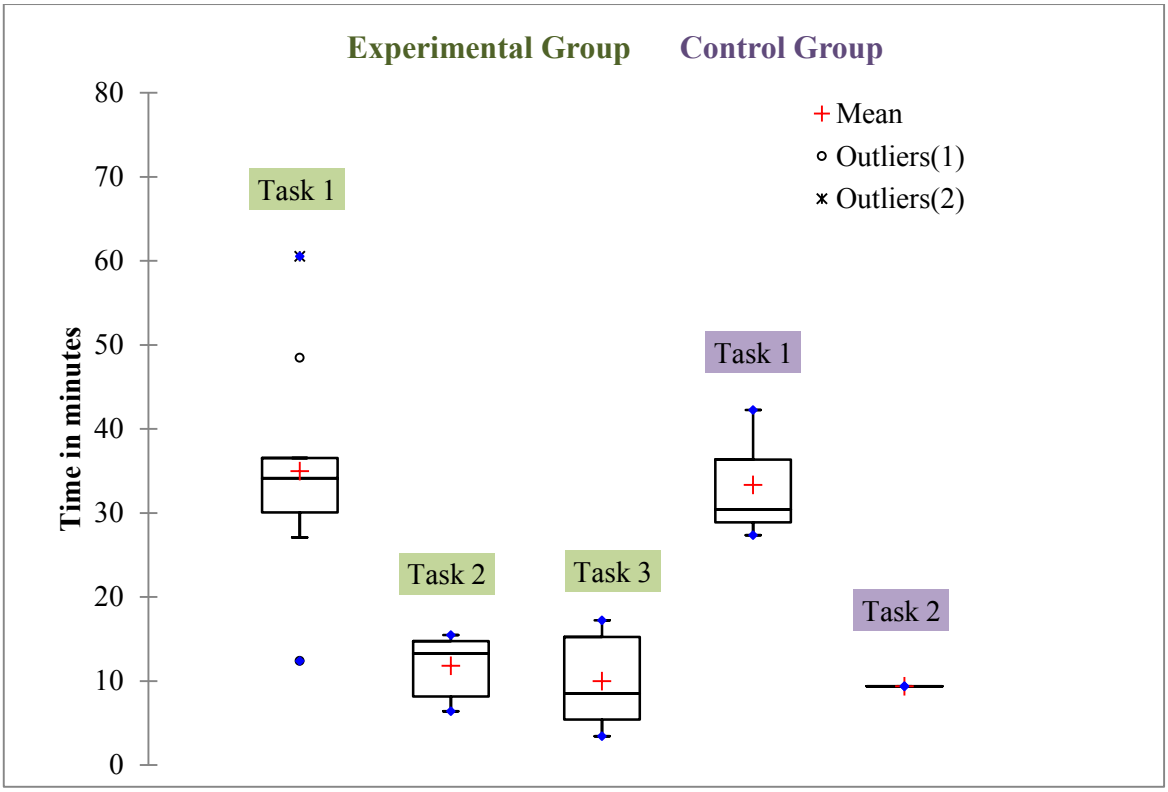


Figure 6.23: Box plot showing task completion rates across completed tasks for Experimental and Control groups at KeMU, Experiment 3

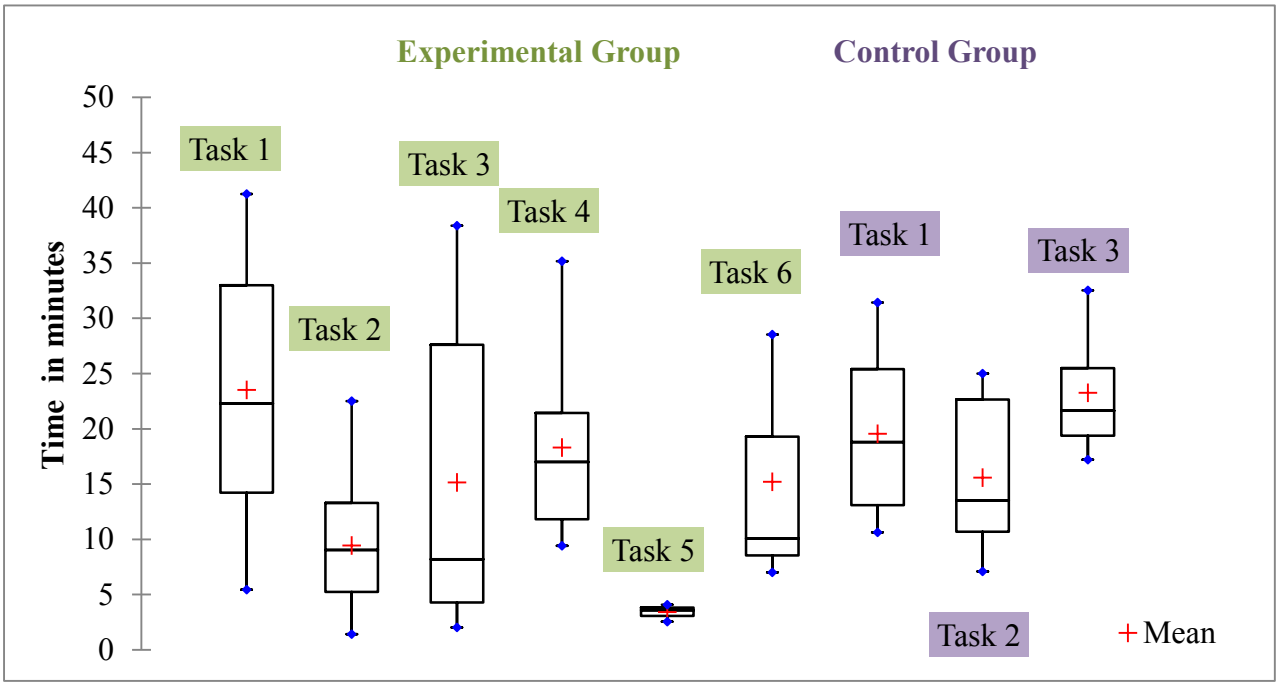


Figure 6.24: Box plot showing task completion rates across completed tasks for Experimental and Control groups at JKUAT, Experiment 3

Table 6.15: Statistical time-on-task results per completed task in the third Experiment

		Task 1		Task 2		Task 3	
Institution	**Statistical Metric**	**Experimental**	**Control**	**Experimental**	**Control**	**Experimental**	**Control**
KeMU	***M***	34.99	33.35	11.83	9.39	9.97	-
	SD	13.47	7.86	3.45	0	5.40	-
	t	t (6) = 0.26		t(8) = 1.87		-	
	p	**0.40**		-		-	
JKUAT	***M***	23.53	19.56	9.42	15.56	15.16	24.25
	SD	10.69	7.79	5.59	6.99	13.00	5.66
	t	t (21) = 1.09		t (11) = -2.18		t (12) = -1.93	
	p	**0.14**		**0.03**		**0.04**	

There were outliers in the mean completion time for the first task. To determine whether the outliers influenced results for the first task, the analysis was conducted twice, with outliers and without outliers. The second analysis (without outliers) concluded there was still no significant difference in mean completion time on the first task between the experimental and control groups.

Figure 6.24 shows the time distributions for only the completed tasks in the experimental and control groups at JKUAT. There was no significant difference in the mean completion time in the first task in the experimental group and the mean completion time in the first task in control group. With a p- value of 0.14, the first null hypothesis cannot be rejected. Therefore, the mean completion time for the first task in the experimental group is less than the mean completion time for the first task in the control group.

However, there was a significant difference in the mean completion time in the second task in the experimental group and the mean completion time in the second task in the control group. Similarly, there was a significant difference in the mean completion time in the third task in the experimental group and the third task in the control group. With both p-values < 0.05 in the second and third tasks, the first null hypothesis is rejected for these tasks in favor of the alternate hypothesis. Therefore, the mean completion time for the second task in the experimental group is less than the mean completion time for the second task in the control group. Similarly, the mean completion time for the third task in the experimental group is less than the mean completion time for the third task in the control group.

6.3.2.2 Discussion: Time-on-Task in third Experiment

Results from KeMU and JKUAT indicate that the mean completion time in the experimental group is not less than the mean completion time in the control group. This is supported by results from KeMU that indicate that the mean completion time per task in the experimental group is not less than the mean completion time per task in the control group. This shows that the scaffolding techniques did not enable faster completion times than the non-scaffolded environment.

However, results from JKUAT indicate that the mean completion times for the second and third tasks in the experimental group are less than the mean completion time for the second and third tasks in the control group. These results indicate that after the initial familiarization with a new environment, learners using the scaffolding techniques were able to complete tasks significantly faster than learners using the non-scaffolded environment. This indicates the learnability of the scaffolded environment.

Lastly, results indicate that the mean time on incomplete tasks in the experimental group is not less than the mean time on incomplete tasks in the control group. This shows that the scaffolding techniques did not enable learners to reach failure states quicker than the non-scaffolded environment.

6.3.3 Summary of Time-on-Task Results

Table 6.16 shows the consolidated statistical time-on-task results in the second and third experiments.

Table 6.16: Statistical time-on-task results in the second and third Experiments for attempted and completed tasks in Experimental and Control groups

		Second Experiment				**Third Experiment**			
		Completed tasks		**Incomplete tasks**		**Completed tasks**		**Incomplete tasks**	
Insti-tution	**Statistical Metric**	**Experi-mental**	**Control**	**Experi-mental**	**Control**	**Experi-mental**	**Control**	**Experi-mental**	**Control**
UWC	***M***	20.76	22.18	7.51	21.70				
	SD	9.99	8.05	6.34	12.74				
	t	$t(18) = 0.41$		$t(15) = -3.27$					
	p	**0.34**		**0.003**					
KeMU	***M***					20.88	27.36	30.65	33.39
	SD					15.01	13.59	21.41	16.77
	t					$t(4) = 0.86$		$t(16) = -3.44$	
	p					**0.22**		**0.37**	
JKUAT	***M***	22.46	22.44	34.00	30.86	15.82	18.75	22.84	31.39
	SD	17.77	13.00	28.27	21.74	11.15	7.51	17.66	19.92
	t	$t(26) = 0.004$		$t(26) = 0.34$		$t(52) = 1.34$		$t(57) = -1.78$	
	p	**0.49**		**0.37**		**0.09**		**0.04**	

In all the four experiment sessions, the mean completion time in the experimental group was not less than the mean completion time in the control group. This was supported by results from UWC, JKUAT and KeMU that indicated that that the mean completion time per task in the experimental group is not less than the mean completion time per task in the control group. These results indicate that the scaffolding techniques did not enable faster completion times than the non-scaffolded environment.

Results from the second experiment at UWC and the third experiment at JKUAT indicated that learners using the scaffolding techniques may reach failure states quicker than those who used the non-scaffolded environment.

Lastly, results from the second experiment at UWC and the third experiment at JKUAT indicate that after the initial familiarization with a new environment, learners using the scaffolding techniques are able to complete tasks significantly faster than learners using the non-scaffolded environment. This also indicates the learnability of the scaffolded environment.

6.4 Efficiency

Efficiency was measured by calculating the ratio of task completion rate and the mean time-on-task, where task completion rate is the percentage of participants who completed each task. Mean time-on-task is the average time spent on all tasks, complete and incomplete. Therefore, for each of the four experiment sessions that contained experimental and control groups, the number of completed tasks and the mean time on all tasks were used to calculate efficiency for each task.

Table 6.17 and Table 6.18 show the efficiency calculations for UWC and JKUAT in the second experiment. Table 6.19 and Table 6.20 show the efficiency calculations for KeMU and JKUAT in the third experiment. The efficiency ratio was calculated for all the tasks completed by at least one learner, in both control and experimental groups. The dashes in the tables indicate where there was no learner who completed the task. For example, to calculate the efficiency for the first task in the second experiment at UWC, the number of learners who completed the tasks in the control group was 7 out of 12 while that in the experimental group was 12 out of 14. The completion rates are 58% for control group and 85% for experimental group. The mean time for the first task (including incomplete and complete attempts) in the control and experimental group was 26.20 minutes and 23.93 minutes, respectively. Therefore, the efficiency ratios for the first task for the control group and the experimental group are 2.21 and 3.55 respectively. This shows that learners in the experimental group were more efficient in completing the first task than learners in the control group.

Table 6.17: Task completion rate, Average task time and Efficiency calculations for UWC, Experiment 2

	Experimental Group			Control Group		
	Completion rate %	**Mean task time on all tasks**	**Efficiency**	**Completion rate %**	**Mean task time on all tasks**	**Efficiency**
Task 1	85	23.93	3.55	58	26.20	2.21
Task 2	70	13.45	5.20	33	19.57	1.69
Task 3	50	9.12	5.48	-	-	-

Table 6.18: Task completion rate, Average task time and Efficiency calculations for JKUAT, Experiment 2

	Experimental Group			Control Group		
	Completion rate %	**Mean task time on all tasks**	**Efficiency**	**Completion rate %**	**Mean task time on all tasks**	**Efficiency**
Task 1	69	18.75	3.68	82	16.55	4.95
Task 2	50	45.61	1.09	79	31.97	2.47
Task 3	20	17.49	1.14	17	30.07	0.57
Task 4	50	31.35	1.59	50	22.00	2.27

Table 6.19: Task completion rate, Average task time and Efficiency calculations for KeMU, Experiment 3

	Experimental Group			Control Group		
	Completion rate %	**Mean task time on all tasks**	**Efficiency**	**Completion rate %**	**Mean task time on all tasks**	**Efficiency**
Task 1	69	36.29	1.90	27	37.96	0.71
Task 2	72	13.43	5.36	13	27.32	0.47
Task 3	71	11.69	6.07	-	-	-

Table 6.20: Task completion rate, Average task time and Efficiency calculations for JKUAT, Experiment 3

	Experimental Group			Control Group		
	Completion rate %	**Mean task time on all tasks**	**Efficiency**	**Completion rate %**	**Mean task time on all tasks**	**Efficiency**
Task 1	75	29.17	2.57	38	34.92	1.09
Task 2	89	9.40	9.47	57	21.00	2.71
Task 3	60	16.73	3.59	36	20.27	1.78
Task 4	58	18.86	3.08	-	-	-
Task 5	50	5.91	8.46	-	-	-
Task 6	60	15.47	3.87	-	-	-

The results show that, apart from the second experiment at JKUAT, the ratios between task completion rate and the mean time-on-task in the experimental groups are higher than the ratios between task completion rate and the mean time-on-task in the control groups. Therefore, the efficiency ratio is higher in all these experimental groups than in the control groups.

The results from the second experiment at JKUAT could be explained by learners in the control group completing more tasks. As was explained in the results for task success, this was attributed to learners in the control group editing previously completed programs as opposed to starting them from scratch.

These results indicate that the scaffolding techniques enabled learners to complete programming tasks more efficiently than the non-scaffolded environment.

6.5 Errors

Errors were measured by investigating the number of run-time errors for all the programs in the control and experimental group and the errors that triggered scaffolding techniques that offered support for error detection, only for the experimental group.

To recap, the derived hypotheses were:

H_0: The mean number of run-time errors encountered in the experimental group is not lower than the mean number of run-time errors encountered in the control group. This is the null hypothesis.

H_1: The mean number of run-time errors encountered in the experimental group is lower than the mean number of run-time errors encountered in the control group. This is the alternate hypothesis.

An independent sample t-test was conducted to compare the number of run-time errors in the experimental group and the number of run-time errors in the control group.

Results from the second and third experiments are presented first, followed by a discussion.

6.5.1 Second Experiment

Table 6.21 shows the statistical results on the mean number of errors for all tasks, first task and second tasks in the second experiment. The first analysis was conducted on the mean number of errors for all the tasks. There was a significant difference between the mean number of run-time errors encountered on all the tasks in the experimental group at UWC and the mean number of run-time errors encountered on all the tasks in the control group. With a p-value of 0.0004, the null hypothesis is rejected in favor of the alternate hypothesis. Therefore, the mean number of run-time errors encountered in the experimental group at UWC is lower than the mean number of run-time errors encountered in the control group.

On the contrary, there was no significant difference between the mean number of run-time errors encountered on all the tasks in the experimental group at JKUAT and the mean number of run-time errors encountered on all the tasks in the control group. With a p-value of 0.41, the null hypothesis cannot be rejected. Therefore, the mean number of run-time errors encountered in the experimental group at JKUAT is not lower than the mean number of run-time errors encountered in the control group.

A second analysis was conducted on the mean number of run-time errors per task, as shown in Table 6.22. There was a significant difference between the mean number of run-time errors encountered on the first task in the experimental group at UWC and the mean number of run-time

Table 6.21: Statistical results on the mean number of errors for all tasks, first task, and second task at UWC and JKUAT in the second experiment

		All tasks		**Task 1**		**Task2**		**Task 3**	
Institution	**Statistical Metric**	**Experimental**	**Control**	**Experimental**	**Control**	**Expeimental**	**Control**	**Experimental**	**Control**
UWC	***M***	1.93	6.41	1	7.61	3	3		
	SD	1.43	4.38	0	4.33	1.41	2.64		
	t	$t(20) = -3.97$		$t(12) = -5.50$		$t(3) = -5.50$			
	p	**0.0004**		**$p = 0.00006$**		**$p = 0.05$**			
JKUAT	***M***	5.5	5.11	4	3.55	7.57	5.66	3	5.66
	SD	5.70	3.61	3.60	2.00	7.36	4.37	2.82	3,91
	t	$t(17) = 0.23$		$t(2) = 0.20$		$t(9) = 0.62$		$t(2) = -1.16$	
	p	**0.41**		**0.42**		**0.27**		**0.18**	

errors encountered on the first task in the control group. With a *p*-value of 0.00006, the null hypothesis is rejected in favor of the alternate hypothesis. Therefore, the mean number of run-time errors encountered in the experimental group at UWC is lower than the mean number of run-time errors encountered in the control group.

However, there was no significant difference between the mean number of run-time errors encountered in the second task in the experimental group at UWC and the mean number of run-time errors encountered on the second task in the control group. With a *p*-value of 0.05, the null hypothesis cannot be rejected. Therefore, the mean number of run-time errors encountered in the experimental group at UWC is not lower than the mean number of run-time errors encountered in the control group. Statistical analysis was not performed on the third and fourth tasks because these had only one learner each attempting these tasks in the control group.

At JKUAT, there was no significant difference between the mean number of run-time errors encountered in the first three tasks in the experimental group and the mean number of run-time errors encountered in the first three tasks in the control group. With all p-values > 0.05, the null hypothesis cannot be rejected for these tasks. Therefore, the mean number of run-time errors encountered in the experimental group is not lower than the mean number of run-time errors encountered in the control group. Statistical analysis was not performed on the fourth and fifth tasks because these had only one learner with errors each in the experimental group.

Table 6.22 shows that most of the error prompts were encountered in the first three programs. A further analysis was conducted on UWC's and JKUAT's experimental group data to investigate which parts of the programs that the error prompts occurred. Appendices K1 and K2 contain the raw data that was used to conduct this analysis. The results revealed that most of the error prompts occurred in the main class chunk. Examples of the error prompts displayed to the learners are when the main class does not begin with an upper case letter (Figure 6.25 in italics) and some in the main method where a learner did not correctly complete the for-loop declaration (Figure 6.26 in italics).

Additional analysis on the data from the second experiment revealed that learners in the control group had syntactical errors that could be reduced by scaffolding techniques found in the scaffolded environment. For example, Figure 6.27 shows a program of a learner in the control group in which the keywords 'String' and 'System' were written with a lower case 's' (in bold). In the scaffolded environment, a scaffolding technique that provides default statements such as 'System.out.println()' reduces the occurrence of such syntax errors. It was noted that none of the programs written by learners in the control group contained header comments (as can be seen from Figure 6.27); this is as opposed to the scaffolded environment that guides the learner to create header comments.

Table 6.22: Mean number of run-time errors and scaffolded errors in attempted tasks (per task) at UWC and JKUAT Second Experiment

	UWC			JKUAT		
	Average number of errors per learner		Average number of error prompts per learner	Average number of errors per learner		Average number of error prompts per learner
	Experimental	Control	Experimental	Experimental	Control	Experimental
Program 1	1	8	1	4	4	2
Program 2	3	3	1	8	6	2
Program 3	3	2	2	3	6	1
Program 4	3	2	-	4	5	-
Program 5	-	-	-	2	7	-

Main Class Button Pre Main Class Child
Started at Basic Interface
Editor
Main class Error classname does not begin with an upper case

Figure 6.25: Error prompt showing incorrect creation of the main class

Main Method Button Pre Main Method Child
Editor
System.out.println selected from statement dialog
for-loop selected from statement dialog
Main Method Error: A for loop syntax doesnt have two commas within the declaration

Figure 6.26: Error prompt showing incorrect completion of the for-loop

```
import java.util.Scanner; import java.util NoSuchElementExeption; class
Compute{ public static void main(string[]args){ int num = 1, sum = 0, avg; for
(num = 1;num< 21; num ++) { sum+= num; } avg=sum/20;
system.out.println("The sum is "+ sum); system.out.println("The average is"+
avg); } }23/07/2014 16:43:18:621
```

Figure 6.27 A program showing the Keywords 'String' and 'System' written in lower case 's' (in bold)

6.5.2 Third Experiment

For the third experiment, JKUAT is used to illustrate the results on errors since it had the highest number of participants in both the control and the experimental groups. Table 6.23 shows the statistical results on the mean number of errors for all tasks, first, second and third tasks in the third experiment at JKUAT.

There was a significant difference between the mean number of run-time errors encountered in all the tasks in the experimental group at JKUAT and the mean number of run-time errors encountered on all the tasks in the control group. With a p-value of 0.0003, the null hypothesis is rejected in favor of the alternate hypothesis. Therefore, the mean number of run-time errors encountered in the experimental group at is lower than the mean number of run-time errors encountered in the control group.

Table 6.23 shows that the average number of run-time errors encountered per task in the non-scaffolded environment is significantly higher than the average number of run-time errors encountered in the scaffolded environment. For example, there was a significant difference between the mean number of run-time errors encountered in the first, second and third tasks in the experimental group and the mean number of run-time errors encountered in these tasks in the control group. With *p*-values < 0.05, the null hypothesis is rejected in favor of the alternate hypothesis. Therefore, the mean number of run-time errors encountered in the experimental group for these tasks is lower than the mean number of run-time errors encountered in the control group.

A further analysis was conducted on JKUAT's experimental group data to investigate where most of the error prompts occurred. Appendix K3 shows the raw data that was used to conduct this analysis. Table 6.24 shows that most of the error prompts were encountered in the first program, at two error prompts on average per learner. The additional analysis revealed that most of the error prompts were encountered within the main class chunk. Examples of the error prompts displayed to

Table 6.23: Statistical results on the mean number of errors for all tasks, first, second and third tasks at JKUAT in the third experiment

		All tasks		**Task 1**		**Task2**		**Task 3**	
	Statistical metric	**Experimental**	**Control**	**Experimental**	**Control**	**Experimental**	**Control**	**Experimental**	**Control**
JKUAT	***M***	1.78	5.02	2.05	5.83	1.6	3.83	1.75	7
	SD	1.08	5.39	1.16	7.03	0.91	3.15	1.30	3.42
	t	t (40) = -3.64		t (18) = -2.24		t (14) = -2.28		t (4) = -3.97	
	p	**0.0003**		**0.018.**		**0.019**		**0.008**	

Table 6.24: Average number of run-time errors and scaffolded errors in attempted tasks in control and experimental groups, JKUAT Third Experiments

	JKUAT		
	Average number of errors per learner		Average number of error prompts per learner
	Experimental	Control	Experimental
Program 1	2	6	2
Program 2	2	4	1
Program 3	2	6	1
Program 4	1	5	1
Program 5	1	1	1
Program 6	1	1	1

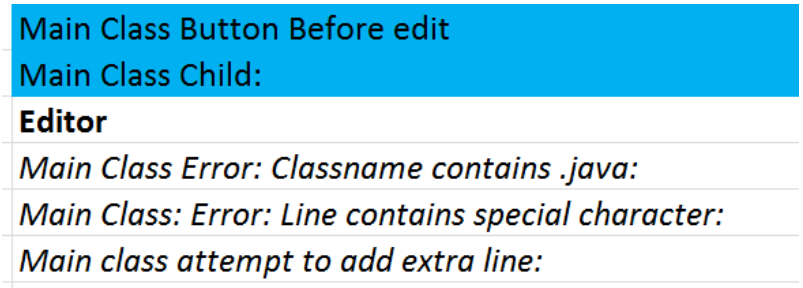


Figure 6.28: Error prompts encountered within the main class in italics

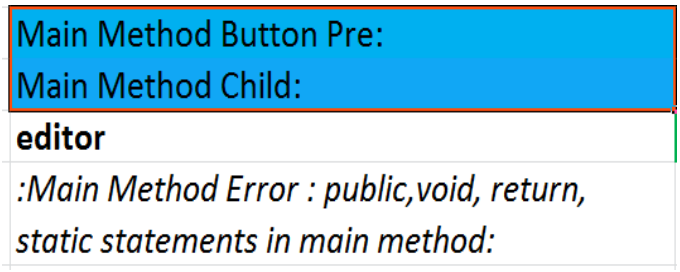


Figure 6.29: Error prompts encountered within the main method in italics

the learners are the main class containing special characters (Figure 6.28 in italics) and some in the main method where a learner wrote public, void or return statement within the main method (Figure 6.29 in italics).

6.5.3 Discussion: Error Results from the Second and Third Experiments

Table 6.25 shows the error results from the second and third experiments at UWC and JKUAT on all tasks and the first three tasks. Of the three experiment sessions, two resulted in a significantly lower mean number of errors across all the tasks in the experimental group than in the control group. Further, the first task at UWC (second experiment) and the first three tasks at JKUAT (third experiment) resulted in a significantly lower mean number of errors in the experimental group than in the control

group. The results indicate that scaffolding techniques may lead to fewer run-time errors. Further, additional analyses indicate that the scaffolding techniques may capture some syntactical errors that a non-scaffolded environment may not.

Table 6.25: Statistical error results from the second and third Experiments at UWC and JKUAT across all tasks and the first three tasks

		All tasks		**Task 1**		**Task2**		**Task 3**	
	Statistical metric	**Experimental**	**Control**	**Experimental**	**Control**	**Experimental**	**Control**	**Experimental**	**Control**
UWC (Exp 2)	***M***	1.93	6.41	1	7.61	3	3		
	SD	1.43	4.38	0	4.33	1.41	2.64		
	t	$t(20) = -3.97$		$t(12) = -5.50$		$t(3) = -5.50$			
	p	**0.0004**		**$p = 0.00006$**		**$p = 0.05$**			
JKUAT (Exp 2)	***M***	5.5	5.11	4	3.55	7.57	5.66	3	5.66
	SD	5.70	3.61	3.60	2.00	7.36	4.37	2.82	3.91
	t	$t(17) = 0.23$		$t(2) = 0.20$		$t(9) = 0.62$		$t(2) = -1.16$	
	p	**0.41**		**0.42**		**0.27**		**0.18**	
JKUAT (Exp 3)	***M***	1.78	5.02	2.05	5.83	1.6	3.83	1.75	7
	SD	1.08	5.39	1.16	7.03	0.91	3.15	1.30	3.42
	t	$t(40) = -3.64$		$t(18) = -2.24$		$t(14) = -2.28$		$t(4) = -3.97$	
	p	**0.0003**		**0.018**		**0.019**		**0.008**	

6.6 Scaffolding Techniques Used

This section and the next will present which and how scaffolding techniques were used. To organize the discussion on which scaffolding techniques were used, the three criteria mentioned in the evaluation chapter (Section 5.5.2) will be used, namely use (initial and reuse), fading of the scaffolds if any, and how the scaffolding was used from one program to another (progression). Verbatim feedback is used to illustrate some of the results. In some of the graphs, UWC-2 means the second experiment at UWC, KeMU-3 means the third experiment at KeMU, and so on.

As was described in Chapter 4, the scaffolded environment provided three kinds of scaffolding techniques: (i) scaffolding that was static and had to be used to complete a program; (ii) scaffolding that was automatically provided but could be cancelled or faded over time; and (iii) scaffolding that was not automatically activated and the learner needed to initiate its use. This section discusses the use of scaffolding techniques based on these three categories.

6.6.1 Use of Static Scaffolding

Static scaffolding was provided using two techniques: (i) a program overview at the main interface; and (ii) editing of a program one chunk at a time at the editor. The program overview offered a structure that provided a layout of the program and restricted the construction of a program in a certain order. The editing screen enabled construction of the program only one part at a time. The program overview and the editing screen were used to navigate between the program parts and edit them, respectively. Consequently, these two scaffolding techniques were mostly used to create the programs. Figure 6.30 shows a comparison of the use of static scaffolding techniques in complete and incomplete programs across the four experiment sessions in the second and third experiments.

Figure 6.30 reveals that there was variation in use of the static scaffolding across the experiments. For example, in the second experiments at UWC and JKUAT, learners who completed programs edited the program chunks more than the learners who did not complete programs. Whereas in the third experiment at KeMU, learners who did not complete programs edited the program chunks more than the learners who completed programs. This variation in use could be because learners had to interact with the static scaffolds to construct the programs, whether or not they completed the programs successfully.

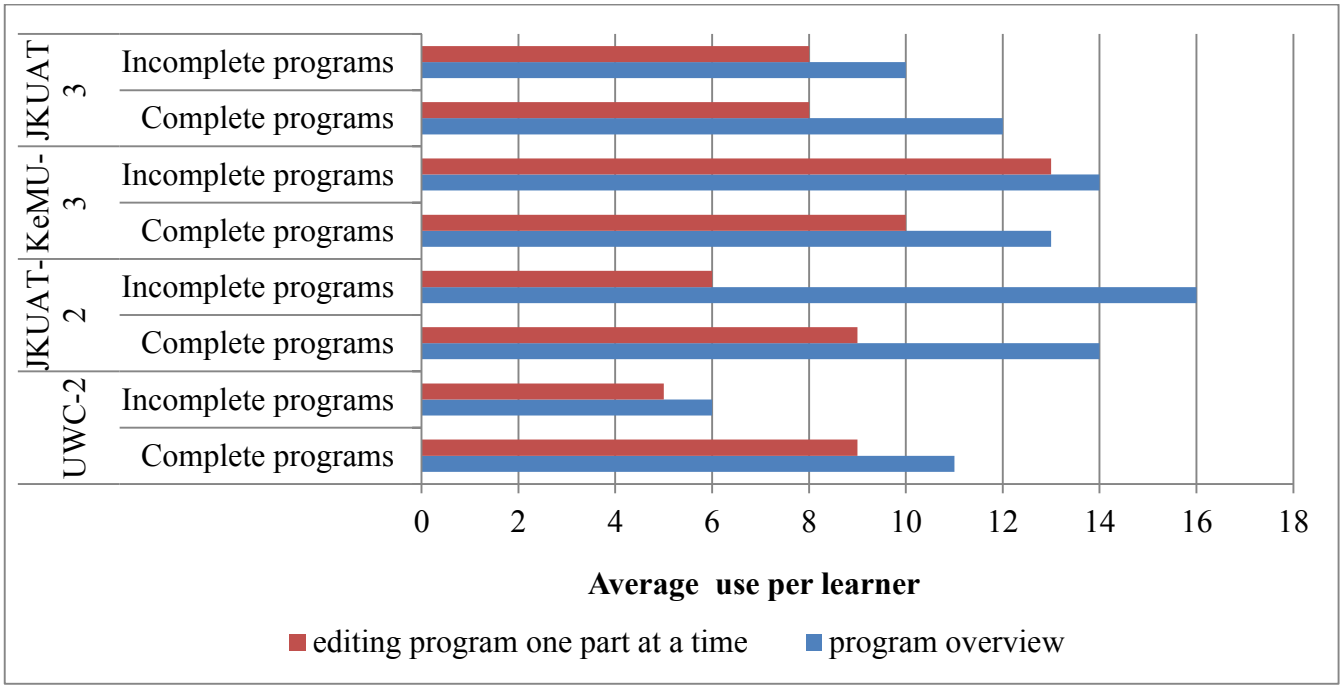


Figure 6.30: Comparison of use of static scaffolding techniques between incomplete and complete programs at UWC, KeMU and JKUAT in Experiments 2 and 3

Further analysis revealed that the static scaffolds support correct construction of programs on a mobile phone. Figure 6.31 is used to explain this point. The lines indicate the sequence of interaction in this section of the learner's logs. The learner started by clicking on the main class button (line 1) and then clicked on the main class' expanded view (main class child) that links to the editor. While at the editor, the learner attempted to add an extra line while creating the class name. The learner correctly created the class name and returned to the main interface that contained the program overview (line 6 and 7), and again clicked the main class button and its child to go back to the editor. At the editor, the learner attempted to add an extra line (line 11) and then deleted the class name that was previously created, which then restored the main class to the default text (line 12 and 13). Thereafter, the learner returned to the editor to edit the main class chunk three times, including two more attempts at adding extra code. The learner eventually proceeded to create the header comment as shown in line 37. This example has shown that editing a program one part at a time, while providing some restrictions, enabled the learner to work correctly on only that program part. After the header chunk was unlocked, the restricted interface enabled the learner to proceed to the next part.

Additional analysis was conducted on the use of static scaffolding across the different tasks. The results from the third experiment at JKUAT are used to illustrate this because it is the group where the most number of tasks were attempted and completed. Figure 6.32 shows the progression of use of static scaffolding from the first program to the sixth program in the third experiment at JKUAT. Learners used the static scaffolding nearly two times less in the second program than in the first. The reduced use of the static scaffolding in the second program could be due to learners having familiarized themselves with the interface, than when they encountered it for the first time in the first program. These results indicate that the static scaffolding was mostly used in the first program than in subsequent programs for both incomplete and complete programs. Some of the programs that were completed in the fourth task were constructed at the advanced interface. This explains the increased use of static scaffolding since learners encountered this interface for the first time. Further, all the tasks that were completed in the sixth program were completed within the advanced interface. These tasks required the construction of a method in addition to the main class, header and main method. This explains the increased use of static scaffolds at the sixth program.

Line 1	Main Class Button before first edit
Line 2	Main Class Child to editor
Line 3	**Editor Instructions at mainclass**
Line 4	**Editor**
Line 5	*attempt to add extra line at main class*
Line 6	**Program overview**
Line 7	YourClassnam.java created
Line 8	Main Class Button Post edit
Line 9	Main Class Child to editor
Line 10	**Editor**
Line 11	*attempt to add extra line at main class*
Line 12	*Classname deleted*
Line 13	*Main class restored to default value*
Line 14	**Program overview**
Line 15	Main Class Button before first edit
Line 16	Main Class Child to editor
Line 17	**Editor**
Line 18	**Editor Full Program**
Line 19	**Editor**
Line 20	**Program overview**
Line 21	Main Class Button Post edit
Line 22	Main Class Child to editor
Line 23	**Editor**
Line 24	**Program overview**
Line 25	Main Class Button Post edit
Line 26	Main Class Child to editor
Line 27	**Editor**
Line 28	Classname edited
Line 29	*attempt to add extra line at main class*
Line 30	**Program overview**
Line 31	Main Class Button Post edit
Line 32	Main Class Child to editor
Line 33	**Editor**
Line 34	*attempt to add extra line at main class*
Line 35	**Editor Full Program**
Line 36	**Program overview**
Line 37	Header Button Pre
Line 38	Header Child

Figure 6.31: A section of a learner's logs showing several attempts at adding an extra line within the main class chunk

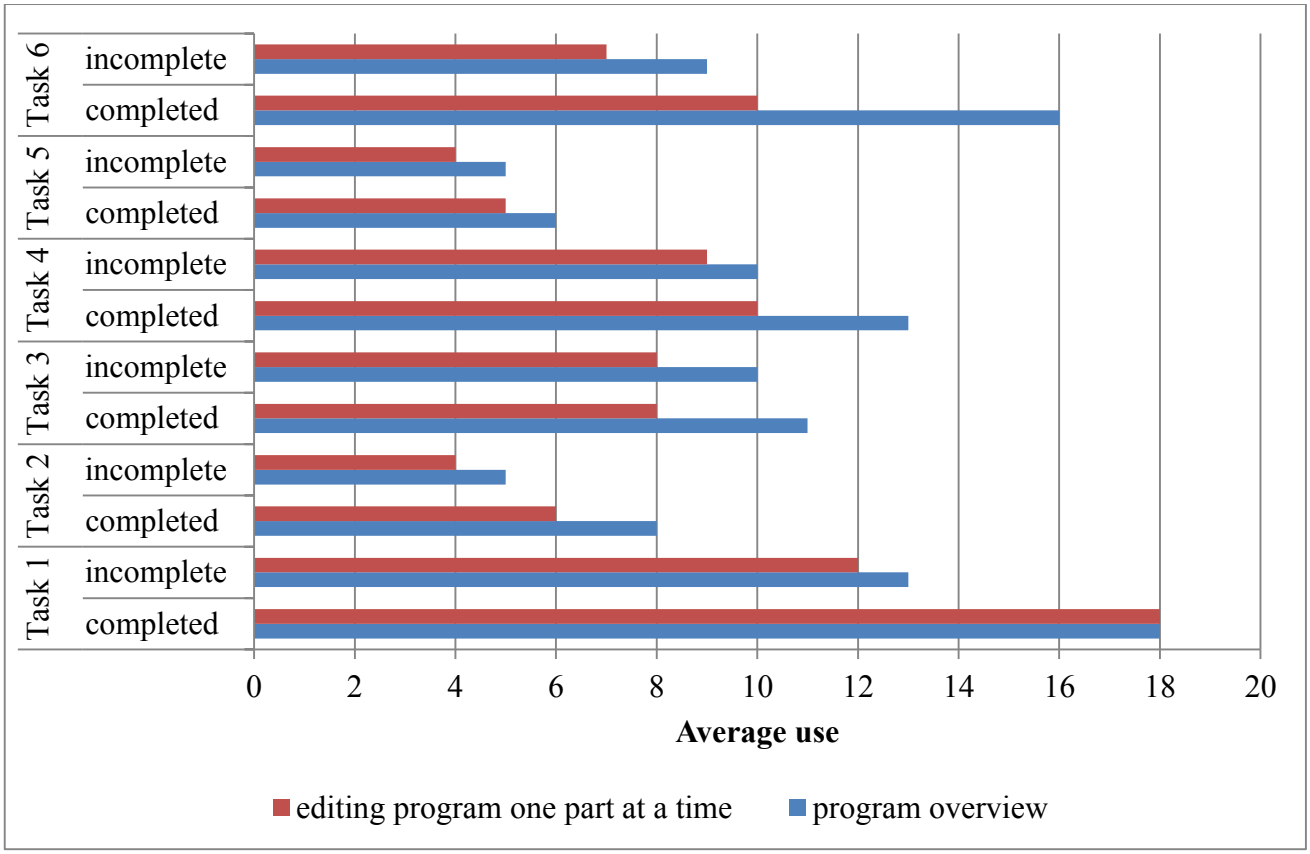


Figure 6.32: Progression of use of static scaffolding techniques in incomplete and complete programs at JKUAT Experiment 3

Further evidence that static scaffolding supported construction of programs on a mobile phone was observed by how learners edited programs after they encountered run-time errors. After learners encountered run-time errors, they were able to go directly to the program chunk that contained the erroneous code in order to edit it.

Importantly, learners found the two static scaffolding techniques useful as evidenced by the verbatim feedback:

'I really enjoyed the program, because it has made my life easy. It is structured; there is a tab for methods, a tab for main, a tab for classes, a tab for documentation. And it allows you to go through them by order.'

'The application divides the program or code into sections then one can then track and write the code properly by following the sections.'

'It is well constructed in that, it clearly states on where to start first."

"The sections are well laid out.'

'The separate segments of program are useful.'

'How the codes are divide into chunks making the application easier to use.'

'The chunks made it easier to construct the program'

In summary, the results show that by guiding the learner to create the program in a certain order, the restricted interface enabled correct construction of a program. Further, by editing one part at a time while checking that the correct code for that part is created, learners were guided towards correct completion of code. The learner's positive feedback on the use of these scaffolding techniques further indicates their usefulness in supporting construction of programs on a mobile phone.

6.6.2 Use of Automatic Scaffolding

Automatic scaffolding was provided using seven techniques: (i) main interface instructions that were automatically displayed the first time the main interface was arrived at; (ii) steps instructions that were automatically displayed in the first two programs to guide the learner on which button to click at the program overview; (iii) editor instructions that were automatically displayed at a tab the first time the editor was used; (iv) the header dialog that was automatically displayed in the first two programs while creating the header chunk; (v) the statement dialog that was automatically displayed in the first two programs and provided default statements to use; (vi) the automatic restriction of the keywords within the main class in the first program; and (vii) error prompts that were automatically displayed when some syntactical errors in the program were encountered. It is worth noting that after the initial automatic provision of these scaffolding techniques, the learner has to initiate their use except for error prompts that are always automatic.

The third experiment at JKUAT will be used to illustrate the use of the automatic scaffolds because it had the highest number of learners in the experiment group. Figure 6.33 shows the average use of automatic scaffolding in incomplete and complete programs at JKUAT, in the third experiment.

The graph reveals that the average use for main interface instructions, editor instructions and statement dialog was the same at twice per program. This indicates that after the initial automatic provision, they were used at least once more. Further, feedback from a learner indicated that they found 'the instructions on which parts of the interface to begin with' useful. Learners who completed programs used the header dialogs more than learners who did not complete programs. Learners found it useful to 'assist with the writing of the comments'. The statement dialog was used twice per program on average.

It was noted that some learners who cancelled the initial automatic provision of the statement dialog did not edit the program chunk and instead, exited the editor interface. For example, Figure 6.34 shows a sequence of program creation showing the statement dialog cancelled twice (in italics), and thereafter the learner went back to the main interface without editing the main method. The learner then enabled the statement dialog on the third attempt (in red). This shows that the statement dialog is useful at least at the initial point when a program chunk is created for the first time.

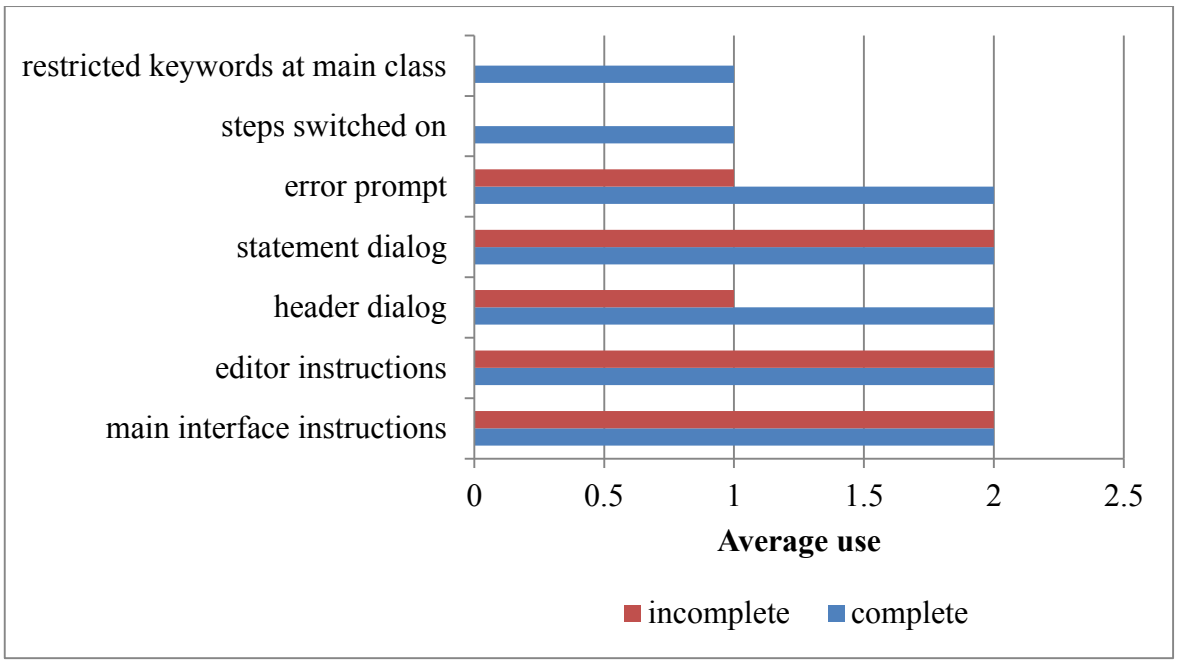


Figure 6.33: Use of automatic scaffolding techniques in incomplete and complete programs at JKUAT Experiment 3

…..
Editor
Statements Dialog Cancelled at main method
Main method not edited
Program overview at main interface
Main Method Button Clicked
Main Method child button clicked to editor
Editor
Statements Dialog Cancelled at main method
Main method not edited
Program overview at main interface
Main Method Button Clicked
Main Method child button clicked to editor
Editor
Statements Dialog Cancelled at main method
main method Statements dialog Enabled
….

Figure 6.34: Sequence of program creation showing the statement dialog cancelled twice while creating the main method, and then enabled on the third attempt

Learners who completed programs encountered more error prompts than learners who did not complete programs. This indicates that the error prompts were effective scaffolding techniques that guided the learner on correct program completion. Further, learners found the error prompts useful as

evidenced by feedback such as, 'the error handling is accurate in pinpointing errors' and learners appreciated 'its ability to detect and in most cases correct errors'.

After the step instructions automatically faded, learners who completed programs switched them on once on average. Similarly, learners who completed programs enabled the restricted keywords at the main class once on average. This indicates that both were effective scaffolding techniques to guide the learner on the use of the interface for the step instructions, and to enable correct completion of the main class for the restricted keywords.

In order to understand the progression of use of automatic scaffolding, analysis was conducted on how they were used on all the tasks. Figure 6.35 shows the progression of use of automatic scaffolding from the first program to the sixth program in the third experiment at JKUAT. In the first program, all the automatic scaffolding was provided by default, except the error prompts. Some, like the statement and header dialogs, were also provided by default in the second program. The use of these scaffolding after the first two programs (except for error prompts) were purely user-initiated. The graph in Figure 6.35 shows that the automatic scaffolding was used mostly in the first three programs than in the last three programs. This is especially so for the main interface instructions that seemed not to be needed until at the sixth task when all the learners were working at the advanced interface. This shows that the main interface instructions were useful for learners when they encountered a new interface and needed information on how to use it.

The header dialog was used to complete tasks until the third program. After this, learners opted to use the provided text boxes to create header comments. This could mean two things: (i) the header dialog provided sufficient support and guidance within the first three programs and learners knew what to do thereafter; and (ii) that the provided text boxes effectively enabled construction of the header comments on the mobile phone's small screen. Both reasons support the propositions that: (i) provision of the header dialog meets learners' needs and once it fades, they are able to continue on the task without it; and (ii) enabling construction of a program one part at a time supports construction of programs on a mobile phone.

The statement dialog was used in all the programs, both complete and incomplete. This indicates that providing default text that learners can reuse supports construction of programs. In fact, the statement dialog was one of the most preferred scaffolding techniques as evidenced by the feedback:

'The statements dialog really makes work easier…', 'It helped that some of the system's code (e.g. for loop, system.out) were already created.', 'The features of this application which were helpful was the fact that the statements were there already.', 'Preset statement helped in typing.'

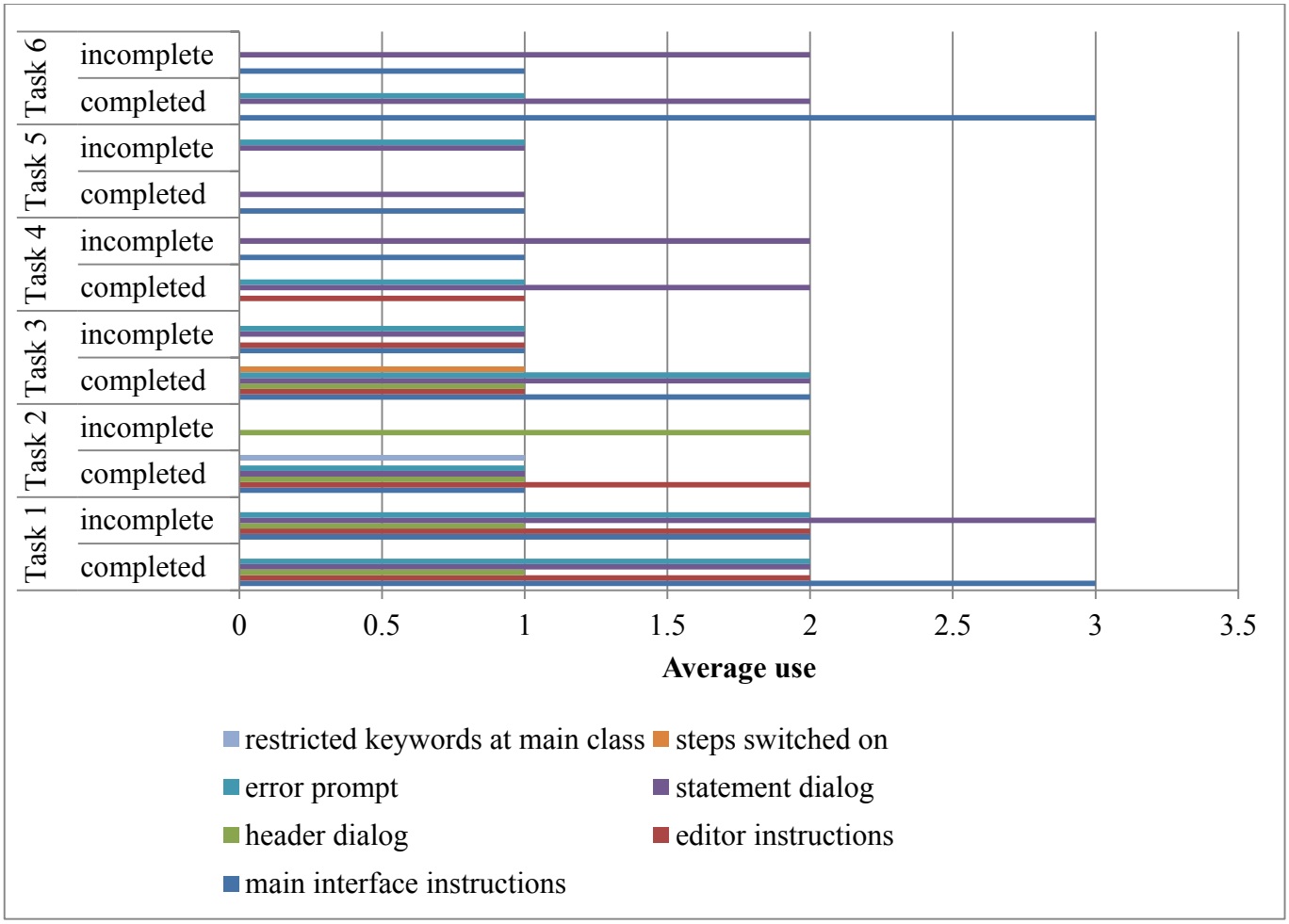


Figure 6.35: Progression of use of automatic scaffolding techniques in incomplete and complete programs at JKUAT Experiment 3

Finally, learners who completed programs in the second and third tasks enabled the restricted keywords at the main class, and enabled the steps instructions; these were not enabled thereafter. This could mean that learners had already understood how to navigate the interface and no longer needed the steps instructions. Further, it shows that the restricted keywords in the first three programs provided guidance on correct construction of the class name, and learners proceeded to create correct programs without these restrictions.

6.6.3 User-initiated Scaffolding Techniques

User-initiated scaffolding was provided using three techniques: (i) view of the full program; (ii) examples; and (iii) hints. These were not automatically provided and required a user to swipe to the interface to view the full program, and to click on provided menus to access examples and hints. Figure 6.36 shows the use of these scaffolds in incomplete and complete programs across the four experiment sessions.

The graph shows that learners who completed programs used all the three user-initiated scaffolding techniques, at all the four institutions. In three out of the four institutions, learners who completed programs viewed the full program more than those who did not complete programs. Learners could view the full program from two points: at the editor while working on the program parts, and at the main interface. Further analysis revealed that learners viewed the full program at three instances: before creating any program parts; during creation of the program chunks; and after they completed the program. The results show that learners who completed programs viewed the full program more during creation of the program parts than learners who did not complete programs. Figure 6.37 and Figure 6.38 are used to illustrate this.

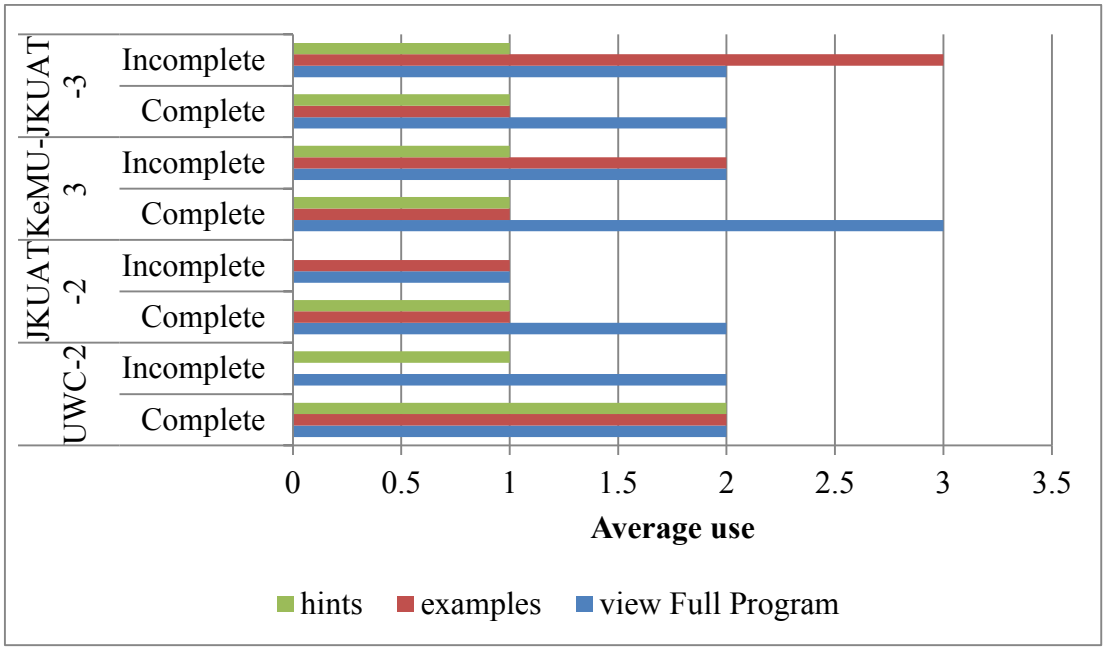


Figure 6.36 : Use of user-initiated scaffolding techniques in all programs across the four experiment sessions

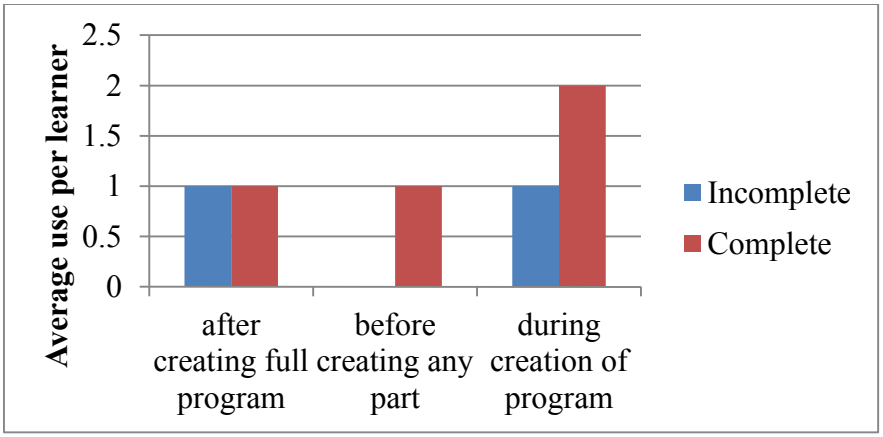


Figure 6.37: Graph showing when the full program was viewed and the average view per learner at UWC, Experiment 2

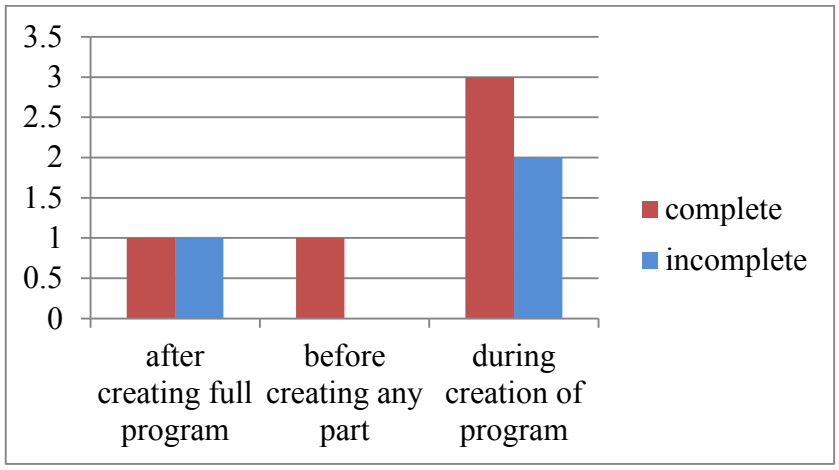


Figure.6.38: Graph showing when the full program was viewed and the average view per learner at JKUAT, Experiment 3

Figure 6.37 shows when learners at the second experiment at UWC viewed the full program. The results show that learners who completed programs viewed the full program at all three points, with more instances during creation of the program. Figure 6.38 shows when learners at the third experiment at JKUAT viewed the full program. The results show that learners who completed programs viewed the full program at all three points, with more instances during creation of the program. Both examples indicate that viewing the full program, while working on the program parts, potentially supports successful construction of programs on a mobile phone.

6.6.4 Summary of Results on which Scaffolding Techniques were used

All the scaffolding techniques were used at least once on average. The static scaffolding techniques enabled learners to correctly create programs, with evidence that the learners found them useful. There was varied use of the automatic scaffolding across the incomplete and complete programs. However, the results indicate that scaffolding techniques such as the statement dialog were always used throughout all programs. Further, learners viewed the instructions when they encountered a new interface. In addition, when learners cancelled the use of a scaffolding technique, such as the header dialog, they tended to enable it again in order to use it to create the header comments. This indicates that after the initial automatic provision of such scaffolding, learners still found these techniques useful in constructing program parts. User-initiated scaffolding was mostly utilized in viewing the full program. Results indicate that learners who completed programs viewed the full program more during creation of the program parts. This indicates that viewing the full program while working on the different chunks is an effective scaffolding technique for constructing programs on a mobile phone. These results have shown that learners were able to correctly construct programs using the three types

of scaffolding techniques. Importantly, the verbatim feedback indicate that learners found these three types of scaffolding techniques useful to support construction of Java programs on a mobile phone.

6.7 How the Scaffolding Techniques were used to Create Programs

The previous section discussed *which* scaffolding techniques were used to attempt and complete programs. This section examines *how* these scaffolding techniques were used to create programs. In order to understand how learners used scaffolding, several characteristics were identified based on learners' behavior during creation of the programs.

6.7.1 Time-based Outliers

While analyzing data for the time-on-tasks, it was observed that some learners spent significantly longer times on tasks than the rest of the group. An analysis was conducted in order to understand how such learners used scaffolding techniques to create programs.

The most amount of time spent on a program by a time-based outlier was 1 hour and 48 minutes. The least was 48 minutes. The common pattern displayed by the time-based outliers was that they initially created the main class, header and main method correctly and then spent a significant amount of time correcting run-time errors, mostly by editing the main method. Further, some of the outliers repeatedly edited the main class (at least 5 times) before they proceeded to create the header. For the learners who spent time repeatedly editing the main class, they encountered several error prompts. Such error prompts included notifications on use of a lower case to start the class name and a notification when they attempted to add extra code (other than the class name) within the main class. In addition, while repeatedly editing the main class, one of the outliers viewed examples severally. The learners who spent more than one hour on a program viewed the full program at least nine times while editing the main method.

Of the six time-based outliers, four eventually completed the programs correctly. This shows that despite the long length of time that these learners spent on the program, the scaffolding techniques that they used (examples, full program and error prompts) supported them to eventually successfully complete the program.

6.7.2 Learners who attempted to edit a chunk repeatedly before proceeding to the next one

While creating a program part for the first time, some learners repeatedly went back to the editor on the same program chunk, before proceeding to the next one. It was observed that most of the repeated visits happened in the main class within the first program. For example, seven learners in the second experiment at UWC exhibited this characteristic, with six of them within the main class and only one within the main method; all in the first program. Further, it was observed that most of the

learners who repeatedly edited a program chunk in the first program did not display this behavior in subsequent programs. This shows that learners could have been familiarizing themselves with the new interface in the first program.

Four scenarios were observed on how learners treated the code within the editor when they repeatedly worked on the same program chunk: (i) no code is created and the default code is restored when the learner goes back to the main interface; (ii) code is created and on repeated visits left un-edited; (iii) code is deleted, which restores the chunk to the default code; and (iv) code is edited on repeated visits.

A common characteristic that was displayed by most of the learners who made repeated visits on a single chunk, is that to get out of this 'loop' they enabled a scaffolding technique that they could use to construct that part of the program. For example, a learner who repeatedly went back to the main method without editing it first continually cancelled the statement dialog. This learner eventually enabled the statement dialog and used one of the default statements to create code within the main method.

In contrast, there were learners who initially worked on each program chunk just once or made at most two attempts before proceeding to the next chunk. The common characteristic among such learners is that they mostly used only the static scaffolding techniques and the provided automatic scaffolding techniques to create the programs with very little use of user-enabled scaffolding. This is further evidence that the static and automatic scaffolding techniques support construction of programs on a mobile phone.

6.7.3 Learners who cancelled the use of scaffolding techniques

It was observed that several learners cancelled some provided scaffolding techniques. For instance, when a statement dialog was used and the for-loop or the Scanner option was selected for the first time, a suggestion to view an example was provided. It was observed that several learners opted not to view these examples. For example, of the 24 learners in the third experiments at JKUAT's experimental group, 18 of them cancelled the use of one scaffolding technique or another, with 11 of these learners rejecting a suggestion to view an example. However, it was noted that all of these cancelations occurred when learners were in the third program or above. This suggests that at this stage of creating programs, learners may not have required extra support such as viewing of examples, but found it sufficient to use the static scaffolding to create programs.

6.7.4 Learners who unlocked the advanced interface

After creating three successful programs, learners could unlock the advanced interface that provided an unrestricted interface on which a program could be created in any order, starting with the main

class. After unlocking the advanced interface, a learner could continue working on this interface or switch back to the basic restricted interface. All the learners who unlocked the advanced interface continued to create programs on this interface. JKUAT's experimental group from the third experiment is used for illustration because it had the highest number of participants who created programs on the advanced interface. Of the 24 learners in the experimental group, eight unlocked the advanced interface. 13 programs were attempted on this interface, with nine of them successfully completed. This shows that the advanced interface enabled construction of successful programs.

In order to understand how learners created the program in this unrestricted interface, in comparison to the restricted interface, an analysis was conducted on the sequence of program creation that learners followed. To recap, the basic interface order of program creation was: main class; header; main method; and then imports; other classes and methods could be created in any order. Table 6.26 shows a summary of the sequence of program creation per chunk in the advanced interface for the learners at JKUAT's third experiment.

This table shows that, after creation of the main class, all the learners created the header chunk. This is similar to the order that was provided in the basic interface. The learners opted to still follow

Table 6.26: Summary of the sequence of program creation in the advanced interface by learners at JKUAT, Experiment 3

Learner	Programming Task and status	Sequence of program creation per chunk
User3	Program 5, Completed	main class, header, other class, main method
User11	Program4, Completed Program 5, Completed Program6, Completed	Main class, header, main method main class, header, Other class, Main method main class, header, main method, method
User12	Program 4, Completed Program 5, Completed	main class, header, imports, other class, main method main class, header, method, main method
User13	Program, Incomplete	main class, header , method, main method
User19	At program4, Completed Program 5, Completed Program 6, Incomplete	main class, header, method, main method main class, header, method, main method main class, header, other class, method
User20	Program4, Completed	main class, header, main method, method
User21	Program 4, Incomplete	main class, header, other class
User22	Program 4, Incomplete	main class, header, main method

this order in the unrestricted interface. Thereafter, for the third part of the sequence, four of the learners worked on the additional class, four worked on the main method, and four worked on the method. Only one worked on the imports chunk. These results indicate three things: (i) all the learners working on the header chunk after the main class is an indication that the basic interface provided an effective guidance that learners followed in the advanced interface; (ii) that some learners still followed the order of the basic interface and constructed the main class after constructing the header also indicates that the basic interface provided an effective guidance; and (iii) learners who worked on the additional class and the methods before proceeding to the main method showed that the advanced interface offered sufficient flexibility that enabled learners to construct programs in any order.

6.7.5 Summary of how scaffolding techniques were used

Time-based outliers viewed the full program more than the learners who constructed programs within the normal time. Such learners also encountered error prompts that guided them towards correct construction of programs. Learners who initially worked on program chunks repeatedly before moving on to the next one did this mostly within the main class, and in the first program. In subsequent programs, most learners did not exhibit this characteristic. Such learners who worked repeatedly on program chunks enabled scaffolds such as the statement dialog; such use enabled them to correctly create the program chunk and move on the next one. Learners cancelled the use of provided scaffolds mostly from the third program onwards, indicating that at this stage most of the learners did not need additional scaffolding. Lastly, some learners who constructed programs at the advanced interface displayed a sequence of program creation that was similar to the one provided at the basic interface.

In summary, these results show that despite the different characteristics exhibited by learners while creating programs, the provided scaffolding techniques enabled the learners to navigate in the scaffolded environment, to get out of repeated construction of chunks, and to create the program with some flexibility at the advanced level.

6.8 Chapter Summary

Three experiments were conducted with a total of 182 learners of programming from four institutions. In all the experiments, learners constructed programming tasks and completed questionnaires at the end of the sessions. In addition, a video recording was taken in the first experiment, and image recordings were used in all the experiments.

The first experiment was conducted at UWC, UCT and KeMU. Results from these sessions indicate that the scaffolding techniques enabled completion of programming tasks. Learners experienced some challenges that were discussed to explain the outcome of the results. Further,

feedback from the learners in this experiment was implemented on a second prototype before using it in the second and third experiments.

The second experiment was conducted at UWC, JKUAT and KeMU. Of the three experiment sessions, one resulted in a significantly higher number of attempted tasks in the experimental group than in the control group, and two resulted in a significantly higher number of completed tasks in the experimental groups than in the control groups. The last experiment was conducted at JKUAT and KeMU. Both experiment sessions resulted in a significantly higher number of attempted and completed tasks in the experimental group than in the control group.

The results indicate that the mean time on complete tasks in the experimental group was not less than the mean time on complete tasks in the control group. This shows that the scaffolding techniques did not enable faster completion times than the non-scaffolded environment. Results from the second experiment at UWC and the third experiment at JKUAT indicated that learners using the scaffolding techniques may reach failure states quicker than those who used the non-scaffolded environment. Results from the second experiment at UWC, the third experiment at KeMU and the third experiment at JKUAT indicate that after the initial familiarization with a new environment, learners using the scaffolding techniques are able to complete tasks significantly faster than learners using the non-scaffolded environment.

Results from the second experiments (except from JKUAT) and the third experiments indicate that learners using the scaffolding techniques were able to complete the programs more efficiently than those using the non-scaffolded environment.

Results from the second experiment at JKUAT indicate that the number of run-time errors encountered in the experimental group is not lower than the number of run-time errors encountered in the control group. In contrast, results from the second experiment at UWC and the third experiment at JKUAT indicate that the number of run-time errors encountered in the experimental group is lower than the number of run-time errors encountered in the control group. These results show that the scaffolding techniques may lead to lower run-time errors. Further, the results indicated that the scaffolding techniques capture some syntactical errors that a non-scaffolded environment may not.

The chapter used verbatim feedback from learners to illustrate some of the results. The verbatim feedback indicated that learners found the scaffolding techniques useful to support construction of Java programs on a mobile phone.

Lastly, this chapter has discussed which and how scaffolding techniques were used to construct programs. The results indicate that learners appreciated the use of static scaffolding. Further, results indicate that the mostly used automatic scaffolding was the statement dialog. In addition, results indicate that the mostly used user-enabled scaffolding was the full program. Results show that despite

the different characteristics exhibited by learners while creating programs, the provided scaffolding techniques enabled the learners to navigate in the scaffolded environment in both the basic and advanced level.

The next chapter presents a synthesis of how these empirical findings have addressed the two research questions, and then concludes the thesis.

Chapter 7 Conclusion

The proposition of this research was that programming environments on mobile phones could include scaffolding techniques that are specifically designed for mobile phones, and designed based on learners' needs. To address this proposition, two research questions were posed:

Which of the theoretically-derived scaffolding techniques support construction of Java programs on a mobile phone?

What is the effect on learners of using the scaffolding techniques to construct Java programs on a mobile phone?

This chapter begins with a synthesis of how the empirical findings addressed the research questions. Thereafter, a discussion follows on the implications of the study. Finally, the chapter discusses the limitations of the study and recommendations for future research.

7.1 Synthesis of Empirical Findings

7.1.1 Which of the theoretically-derived scaffolding techniques support construction of Java programs on a mobile phone?

The findings indicated that all the theoretically-derived scaffolding techniques were used at least once. However, some of the scaffolding techniques showed more frequent use than others and were highly rated by learners. First, the program overview and constructing a program one chunk at a time enabled effective support and guidance towards correct creation of programs. Learners also rated these techniques as most useful. Secondly, the statement dialog was used at least once to construct all programs, even after the first two programs where learners had to initiate its use. In addition, it was one of the most preferred scaffolding techniques by learners. Third, most learners viewed the full program while working on program chunks. In addition, even learners who took the longest times to work on programs viewed the full program frequently. Fourth, the high frequency of error prompts experienced in the first programs indicated that these are useful to capture basic syntactical errors.

7.1.2 What is the effect of using the scaffolding techniques to construct Java programs on a mobile phone?

The synthesis in this section is presented as per the sub-questions that were posed to address the second research question.

What is the effect of using the scaffolding techniques on task success?

Scaffolding techniques enable learners to attempt and complete more programming tasks than a non-scaffolded environment.

What is the effect of using the scaffolding techniques on time-on-task?

The scaffolding techniques do not enable faster average task completion times than a non-scaffolded environment. However, after the initial familiarization with the scaffolded environment, the scaffolding techniques may enable faster completion of tasks than a non-scaffolded environment. Further, the scaffolding techniques may enable learners to reach failure states quicker than those who use a non-scaffolded environment.

What is the effect of using the scaffolding techniques on the ratio between task completion rate and mean time-on-task?

The scaffolding techniques result in a higher ratio between task completion rate and mean time-on-task. This means that learners using the scaffolding techniques are able to complete the tasks more efficiently that those using a non-scaffolded environment.

What is the effect of using the scaffolding techniques on the number of errors?

The scaffolding techniques may lead to fewer run-time errors. Further, the scaffolding techniques capture some syntactical errors that a non-scaffolded environment may not.

What is the effect of using the scaffolding techniques on time-on-task over time?

Learners using the scaffolding techniques spend shorter times in subsequent tasks after the previous tasks.

7.2 Implications of the Study

7.2.1 Theory of constructivism

Constructivism formed the underlying theoretical framework for this research since it embodies the principles of learning by doing and scaffolding. The focus was on designing support for programming environments on devices with limited capabilities, such as mobile phones. The question then is how should constructivism be applied to the design of programming environments on such devices?

Desktop IDEs provide complex environments where a large amount of information is exposed to the learner at the same time, because this is possible on such large screens. This also means that it is possible to provide support to the learner all in one place without the learner having to leave the screen. Further, the learner has to often remember how to navigate through the complex interface, in addition to working on the task at hand. However, providing all the functionality in one place does not work well on small screens. In addition, the intention for small screens is often to provide the user with a simple interface enough such that they can focus on the task at hand. One technique that was used in this study to address the small screen is the static scaffolding technique of completing a program one

part at a time. For example, a learner is presented with only the main class chunk to work on. This way, the learner is able to focus on only the small part of the main class and correctly create it before working on the next small part. Such an approach provides atomic simplicity. Constructivism underlies the principle of atomic simplicity, while enabling active interaction with the content at hand. This shows that constructivism can be applied to the design of programming environments on such limited devices.

Section 2.1 indicated that one of the major arguments against the constructivist approach is that learners are expected to construct new knowledge with minimal guidance. Such an approach may be problematic because evidence has shown that novice learners may struggle to build skills if they are not provided with strong guidance while creating new knowledge (Kirschner et al. 2006). This criticism was discussed by Guzdial (2015), where he posed the question: how then should programming be taught considering that the emphasis has been to learn programming by constructing programs? This study provides two possible answers.

First, the scaffolded environment developed in this study was to be used in addition to the classroom learning; it was not intended to be used on its own. The expectation was that the skills gained from the programming class would be applied when using the scaffolded environment. In the first experiment, learners preferred to use as input a class library they had learnt in the classroom. Further, in the second experiment at KeMU, learners could not attempt two of the tasks since they struggled to understand the related topics in the classroom. Therefore, the first answer to Guzdial's question is: programming can be taught by supporting learners to construct programs on their own, alongside active class teaching that could have other checks to track learners' progress and skill acquisition. A combination of extra support for construction of programs and active instruction could prove more fruitful in teaching programing, than learners applying their programming knowledge alone. It would be a mistake to assume that instruction should exclusively focus on application (Kirschner et al. 2006).

Secondly, the scaffolding techniques designed in this study provide strong guides for the construction of programs. Some recommended techniques that could overcome the criticized unguided nature of constructivism are examples and process worksheets (Kirschner et al. 2006). In this study, these were provided in the form of default code, examples, hints, and a guided process to follow in creating a program. Further, two types of static scaffolding were provided that never faded: the program overview, and completing a program one part at a time. The provision of static scaffolding ensured that there was always support available that addressed the limitations of mobile phones and learners' needs. In addition, the two static scaffolds were among the scaffolding techniques that were highly rated by learners. This leads to the second answer to Gudzial's question: programming can be supported by providing some static scaffolding techniques that are always present to support

construction of programs, and additional scaffolding techniques that provide strongly guided learning.

Thus, the contributions of this study are two recommendations on how to apply the constructivist theory when designing mobile programming environments: (i) the mobile programming environment should be provided in addition to active classroom learning, not as the only platform of constructing programs; and (ii) the mobile programming environment should provide some static scaffolding techniques that never fade, which address the limitations of mobile phones and guide the learner on correct program creation.

7.2.2 Design process

Chapter 4 presented a detailed design process that led to the selection of scaffolding techniques that could support construction of Java programs on a mobile phone. This design process was guided by limitations of mobile phones, challenges faced by learners of programming, and theoretic scaffolding guidelines recommended by several researchers. The challenges faced by learners of programming were specifically elicited for this study. However, these challenges could be applicable to most learners of programming. Further, the two limitations of mobile phones that were considered are standard limitations that present challenges in using most mobile phones. Therefore, this study provides a strong theory-based scaffolding framework that could be used to design mobile programing environments to support construction of programs in other object oriented languages.

The design of some of the scaffolding techniques was influenced by standard Java coding guidelines. For example, the order of the program layout on the main interface was influenced by how a typical Java program would be ordered. All programming languages have coding guidelines. Therefore, the selection of such a scaffolding technique could be replicated when designing for other languages by following their respective coding guidelines.

Two prototypes were designed in this study, the second of which contained modifications from feedback by learners in the first experiment. This follows the learner-centered design process, which is highly recommended when designing for novice learners. Apart from the addition of a header dialog, a chunk for another class, and use of the Scanner class instead of the BufferedReader, the designed scaffolding techniques in the second prototype remained the same as the first prototype. Further, there were some interface related changes, such as provision of a run button and use of tabs, but these did not affect the scaffolding techniques that were designed in the first prototype. Therefore, learners in the first experiment still benefited from the use of the designed scaffolding techniques, as evidenced by a majority of these learners completing the first two tasks. Thus, both prototypes supported the learners to construct Java programs on the mobile phone. This shows that the learner-centered design process can be applied to the design of programming environments on the mobile phone.

Thus, the design contributions of this study are: (i) a theoretic scaffolding framework that could be applied to the design of other mobile programing environments; (ii) a selection process of scaffolding techniques that could apply coding guidelines in other languages; and (iii) a learner-centered design process that includes initial requirements from learners and subsequent feedback used to modify a prototype.

7.2.3 Novel scaffolding techniques and fading mechanisms

The three types of scaffolding techniques that were designed in this study provided a novel way to support construction of Java programs on a mobile phone. The positive feedback from learners indicated that such scaffolding techniques could address the limitations of mobile phones and also meet learners' needs. Some of the scaffolding techniques, such as provision of default code, exist in most of the current IDEs and may be argued as not novel. However for this study, the findings highlighted two things related to provision of default code that could be applied to the design of other mobile programming environments: (i) if the default code is programming keywords, these could be restricted from being edited; and (ii) if the default code can be edited it could be at an advanced level after the learner has gone past the 'beginner' stage.

Some of the scaffolding techniques, like examples and hints, were not as frequently used or as highly rated by learners as the ones described above. In a reputedly difficult programming language such as Java this was surprising. Perhaps the provision of default code and a strongly guided interface minimized the need to view the examples and hints. The design of examples and hints can still be experimented with in different ways. One way is to reuse the learners' successfully created programs as future examples.

One of the characteristics of scaffolding is the fading of scaffolds as the learner acquires skills. This study implemented four fading approaches: (i) Fading of automatically provided steps and instructions after the first two programs; (ii) fading of automatically provided header dialog and statement dialog after the first two programs; (iii) fading of the restricted keywords in the main class after the first two programs; and (iv) fading of the restricted interface after three successful programs. After fading, these scaffolding techniques could be enabled by the user if they wished to. After the fading of the instructions, most of the learners enabled them when they reached the advanced interface. This shows that instructions could be designed to be automatically provided whenever learners encounter a new interface. Learners who reached the advanced interface after creating three successful programs continued to work on that interface. This is a good indication that fading a restricted interface after three successful programs could be used as a design technique for other mobile programming environments. Further, some applications, such as TouchDevelop, provide prompts that guide a user

on where to click in order to create code in the first program. Thereafter, the prompts fade and the user is asked to attempt to create the code on their own. Such fading mechanisms and the ones applied in this study could be extended elsewhere. Nevertheless, there is still room to conduct research on suitable fading models that could be applied to mobile programming environments that use languages such as Java.

7.2.4 Understanding how learners use scaffolding techniques

Results in Section 6.7 provided novel information that could be used to further design scaffolding techniques to support programming on a mobile phone. These results showed various characteristics exhibited by learners as they used the scaffolding techniques.

The findings indicated that the learners who spent a long time on tasks did so while repeatedly correcting code in the main method. Therefore, the design of mobile programming environments could provide more support on creation of the main method. Further, since such learners encountered error prompts that guided them towards correct program creation, automatic prompts with hints could be provided at a certain point when a learners spends a significantly long time on a program.

Some learners repeatedly went back and forth on the same chunk and eventually got out of this 'loop' by enabling a scaffolding technique such as the statement dialog. Therefore, the design of mobile programming environments could provide automatic scaffolds to learners who appear to be moving back and forth on the same chunk without proceeding to the next.

Analysis of how learners used the scaffolds indicates that learners cancelled the use of automatically provided scaffolds mostly from the third program onwards. This gives an indication that automatic fading of scaffolds after two or three successful programs could be appropriate when designing for mobile programming environments.

7.2.5 Contribution to the field of ICT4D

Information and communication technologies for development (ICT4D) is the name given to a range of activity which considers how electronic technologies can be used towards socio-economic development of developing communities worldwide (Donner & Toyama 2009). In this study, the ICT techniques are the designed scaffolding techniques and the Development aspect is in the contribution towards enhancing a skill in a complex subject such as programming.

In developing countries where there may not be a large capital outlay to acquire new equipment for learners, such as desktops and laptops, the solution could be to use the devices that the learners already have and design applications that consider both the limitations of the available devices and learners' needs. This study has shown that this is possible. The prototype developed in this study could be used in future studies that seek to understand the long-term impact of the use of mobile phones in

learning complex subjects such as programming, in the context of a developing country. This is especially because, non-formal learning efforts are a viable means of delivering non-formal learning in a developing country via a smartphone (Jobe 2014).

ICT4D research not only focuses on the rural poor but also on the urban poor (Chepken et al. 2012), who may experience resource constrains. In addition, research indicates there is a gap in studies that consider users who live in urban areas, with a lot more studies conducted with the rural poor (Chepken et al. 2012). The learners who participated in this research were all from universities that were located in urban areas in developing countries, thus representing urban users who nevertheless may be in resource constrained environments. However, even learners who are not necessarily in resource-constrained environments may sometimes find themselves in situations where they may not be able to use a desktop or a laptop. Therefore, this study contributes towards research that provides solutions to the urban poor or those who find themselves in resource-constrained situations while in urban areas.

Mobile for Development research should be conducted using sound conceptual foundation, proven theories, conceptual frameworks or models (Duncombe 2010). The development of the scaffolding techniques in this research were based on a rigorous process using existing scaffolding guidelines. Further, the six-level scaffolding framework used in this study can easily be replicated to design scaffolding techniques that support programming on a mobile phone, in other programming languages.

Lastly, there is a tendency to portray mobile phones as an end, rather than a means to specific social improvements (Burrell 2010). This study has emphasized the fact that the mobile phone can be used as a vehicle for delivering education in resource-constrained environment. Importantly, this study has shown that mobile applications for learning complex subjects that require a practical approach, can be specifically designed to address the limitations of mobile phones and also meet learners' needs.

7.3 Limitations of Research

In this research, the emphasis was on providing scaffolding techniques intended to be used by learners who were just beginning to learn programming using Java. Therefore, they were not used to create complex or high-level programs such as those that develop graphical user interfaces (GUIs). Hence, the simplicity of the programs used in the study may be limiting. However, early success in simple programs allows learners to build both self-confidence and their programming routine, which helps them to transition towards seeing more than simple syntax (Vihavainen et al. 2013).

The choice of Android as an implementation platform means that only specific phones could be used during the experiments. Further, this means that users of other platforms cannot use the

application. Further, there are other limitations of mobile phones, such as limited memory, that were not considered. This study focused on the limitations of small keypads and small screens.

There seemed to be minimal research conducted that provides explicit models on fading mechanisms, especially for mobile programming environments. The fading mechanisms implemented on the scaffolded environment were based on some existing programming environments and some on learners' feedback. This means that the fading mechanisms designed in Chapter 4 may not be exhaustive.

Finally, this research did not evaluate the long-term learning impact of the use of the scaffolding techniques on the eventual performance of students in their programming course, say at the end of the term. This was not evaluated because learners were already exposed to other learning resources and tools for programming and it would have been difficult to determine whether the use of the scaffolding techniques is what directly influenced their eventual success or failure in programming. Nevertheless, given more time and resources, such a long-term study is possible and it is part of future work.

7.4 Opportunities for Future Work

7.1.1 Extension of the system

The application developed in this research is a proof-of-concept prototype that addresses the use of scaffolding techniques to support construction of Java programs on a mobile phone. Future work could add to the system. Possible additions include: the use of syntax coloring; application of automatic code indentation; increasing the complexity of the programs that can be constructed by introducing more scaffolding techniques; development for other mobile platforms apart from Android; and enabling users to store their programs on the cloud directly from the applications, should they wish to. In addition, the error prompts that were used to check for syntax errors were by no means exhaustive. Future work could extend the implementation of error prompts following a more extensive consideration of possible syntax errors. Further, the hints and examples that were provided were based on existing standard coding guidelines. Future work could enable the use of successfully created programs in the system to be used as future examples. With such additions, the application could become more than a tool for novice learners of programming, and be useful to more advanced learners as well.

7.1.2 Additional experiments

Once the above extensions have been implemented in the system, additional experiments could be conducted. Further, in this study, the participants were learners of programing enrolled in introductory programming courses. If extended to suit use by advanced users, such users could be involved in the experiments.

7.1.3 Evaluation with other existing tools

Since the focus of this research was on testing the effect of the scaffolding techniques, evaluation was conducted without comparisons with other available tools. A specific non-scaffolded environment was designed for this study. Other methods of evaluation could be to compare the use of the application developed in this study with the use of existing mobile programing environments such as SAND IDE. Further, the effectiveness of the scaffolding techniques could also be tested by comparing its use with a desktop environment.

7.1.4 Model on fading of scaffolding

Another way that the system could be extended is to implement a more elaborate mechanism for fading scaffolds. There seems to be scarce literature that present elaborate models on when to reduce or remove the level of support as learners progress in working on the task, especially for mobile programming environments. For example, should they stop receiving hints on the second program? On the third? Should they always receive examples in the first program? This presents an opportunity for further work because one key characteristic of scaffolding is fading. In addition, this prototype can be used to conduct experiments specifically targeted at understanding how and when learners prefer to fade scaffolds. Such data could be used to design models on fading of scaffolding in mobile programming environments.

7.1.5 Use of the system to teach a class

In this study, the researcher worked with learners who were already using other tools in their respective programming classes. However, a controlled longitudinal study where learners use the scaffolded environment over an extended period of time is possible. This can be carried out by teaching a class where learners use the mobile programming environment as one of the main resources. This way, it would be possible to test the long-term impact of the scaffolding techniques on the learners' programming skills.

REFERENCES

Ackermann, E., 2001. Piaget's constructivism, Papert's constructionism: What's the difference. *Future of learning group publication*, 5(3), p.438. Available at: http://learning.media.mit.edu/content/publications/EA.Piaget _ Papert.pdf [Accessed June 9, 2014].

Ackermann, E.K., 1996. Perspective-Taking and object Construction. In *Constuctionism in Practice: Designing, Thinking, and Learning in a Digital World*. pp. 25–37. Available at: http://web.media.mit.edu/~edith/publications/1996-persp.taking.pdf [Accessed March 11, 2014].

Adipat, B. & Zhang, D., 2005. *Interface Design for Mobile Applications*, Available at: http://aisel.aisnet.org/amcis2005/494 [Accessed January 19, 2015].

Ajayi, A.O. et al., 2011. A Low Cost Course Information Syndication System. *Journal of Information Technology Education*, 10. Available at: http://jite.org/documents/Vol10/JITEv10IIPp105-118Ajayi947.pdf [Accessed January 15, 2015].

Albert, W. & Tullis, T., 2008. *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*, Morgan Kaufmann.

Alonso-Ríos, D. et al., 2014. A user study on tailoring GUIs for smartphones. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14*. New York, New York, USA: ACM Press, pp. 186–192. Available at: http://dl.acm.org/citation.cfm?id=2554850.2555085 [Accessed August 13, 2014].

Anderson, N. & Gegg-Harrison, T., 2013. Learning computer science in the "comfort zone of proximal development." In *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*. Denver, Colorado: ACM Press, p. 495. Available at: http://dl.acm.org/citation.cfm?id=2445196.2445344 [Accessed February 19, 2014].

Andrews, D., Nonnecke, B. & Preece, J., 2010. Electronic Survey Methodology: A Case Study in Reaching Hard-to-Involve Internet Users. Available at: http://www.tandfonline.com.ezproxy.uct.ac.za/doi/abs/10.1207/S15327590IJHC1602_04#.U15KMVWSzmM [Accessed April 28, 2014].

Apiola, M. & Tedre, M., 2011. Towards a contextualized pedagogy for programming education in Tanzania. In *IEEE Africon '11*. IEEE, pp. 1–6. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6072010 [Accessed May 31, 2014].

Apiola, M., Tedre, M. & Oroma, J.O., 2011. Improving programming education in Tanzania: Teachers' and students' perceptions. In *2011 Frontiers in Education Conference (FIE)*. IEEE, pp. F3G–1–F3G–7. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6142787 [Accessed May 31, 2014].

Athreya, B. et al., 2012. End-user programmers on the loose: A study of programming on the phone for the phone. In *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, pp. 75–82. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6344486 [Accessed December 29, 2014].

Azadmanesh, M.R. et al., 2014. Mobile vs. Desktop Programming Projects. In *Proceedings of the 2nd Workshop on Programming for Mobile & Touch - PROMOTO '14*. New York, New York, USA: ACM Press, pp. 25–28. Available at: http://dl.acm.org/citation.cfm?id=2688471.2688479 [Accessed January 14, 2015].

Azorín, J.M. & Cameron, R., 2010. The application of mixed methods in organisational research: A literature review. *Electronic Journal of Business Research Methods*, 8(2), pp.95–105.

Bati, T.B., Gelderblom, H. & van Biljon, J., 2014. A blended learning approach for teaching computer programming: design for large classes in Sub-Saharan Africa. *Computer Science Education*, 24(1), pp.71–99. Available at: http://www.tandfonline.com/doi/full/10.1080/08993408.2014.897850#tabModule [Accessed January 14, 2015].

Ben-Ari, M., 1998. Constructivism in computer science education. *ACM SIGCSE Bulletin*, 30(1), pp.257–261. Available at: http://dl.acm.org/citation.cfm?id=274790.274308 [Accessed June 9, 2014].

Ben-Bassat Levy, R., Ben-Ari, M. & Uronen, P.A., 2003. The Jeliot 2000 program animation system. *Computers & Education*, 40(1), pp.1–15. Available at: http://www.sciencedirect.com/science/article/pii/S0360131502000763 [Accessed January 15, 2015].

Bennedsen, J. & Caspersen, M.E., 2007. Failure rates in introductory programming. *ACM SIGCSE Bulletin*, 39(2), p.32. Available at: http://dl.acm.org/citation.cfm?id=1272848.1272879 [Accessed March 22, 2014].

Black, A.P. et al., 2013. Seeking grace. In *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*. New York, New York, USA: ACM Press, p. 129. Available at: http://dl.acm.org/citation.cfm?id=2445196.2445240 [Accessed January 7, 2015].

Bowen, H. & Goldstein, P., 2010. *Radio , Mobile Phones Stand Out in Africa ' s Media Communication Landscape*,

Boyer, N.R., Langevin, S. & Gaspar, A., 2008. Self direction & constructivism in programming education. In *Proceedings of the 9th ACM SIGITE conference on Information technology education - SIGITE '08*. New York, New York, USA: ACM Press, p. 89. Available at: http://dl.acm.org/citation.cfm?id=1414558.1414585 [Accessed March 24, 2014].

Bruner, J.S., 1966. *Toward a Theory of Instruction*, Harvard University Press. Available at: http://books.google.com/books?hl=en&lr=&id=F_d96D9FmbUC&pgis=1 [Accessed June 9, 2014].

Burrell, J., 2010. Evaluating Shared Access: social equality and the circulation of mobile phones in rural Uganda. *Journal of Computer-Mediated Communication*, 15(2), pp.230–250. Available at: http://doi.wiley.com/10.1111/j.1083-6101.2010.01518.x [Accessed April 6, 2015].

Butgereit, L., 2012. Dr Math. *International Journal of Mobile and Blended Learning*, 4(2), pp.15–29. Available at: http://www.igi-global.com/article/math-mobile-scaffolding-environment/65084 [Accessed January 23, 2015].

Carter, J., 2010. The Problems of Teaching Programming: Do They Change with Time? In *11th Annual Conference of the Subject Centre for Information and Computer Sciences*. Durham University, pp. 6–10.

Chatley, R., 2001. *Java for Beginners*, Available at: http://chatley.com/kenya/reports/finalreport.pdf [Accessed January 15, 2015].

Chatley, R. & Timbul, T., 2005. KenyaEclipse. *ACM SIGSOFT Software Engineering Notes*, 30(5), p.245. Available at: http://dl.acm.org/citation.cfm?id=1095430.1081746 [Accessed January 7, 2015].

Chepken, C. et al., 2012. ICTD interventions. In *Proceedings of the Fifth International Conference on Information and Communication Technologies and Development - ICTD '12*. New York, New York, USA: ACM Press, p. 241. Available at: http://dl.acm.org/citation.cfm?id=2160673.2160704 [Accessed March 10, 2014].

Churchill, D. & Hedberg, J., 2008. Learning object design considerations for small-screen handheld devices. *Computers & Education*, 50(3), pp.881–893. Available at: http://www.sciencedirect.com/science/article/pii/S0360131506001412 [Accessed May 13, 2014].

Cohen, L., Manion, L. & Morrison, K., 2007. *Research Methods in Education* 6th ed., Routledge.

Cooper, S., Dann, W. & Pausch, R., 2000. Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges*, 15(5), pp.107–107–116–116. Available at: http://dl.acm.org/citation.cfm?id=364133.364161 [Accessed January 15, 2015].

Creswell, J. & Clark, V.P., 2007. Choosing a mixed methods design. In *Designing and conducting mixed methods research*. pp. 53–106. Available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Choosing+a+mixed+methods+design#0.

Cunningham, D. & Duffy, T., 1996. Constructivism: Implications for the design and delivery of instruction. *Handbook of research for educational communications and technology*, pp.170–198. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.2455&rep=rep1&type=pdf [Accessed March 24, 2014].

D'Souza, D. et al., 2008. Transforming learning of programming: a mentoring project. , pp.75–84. Available at: http://dl.acm.org/citation.cfm?id=1379249.1379256 [Accessed January 23, 2015].

Dann, W., Cooper, S. & Pausch, R., 2001. Using visualization to teach novices recursion. In *Proceedings of the 6th annual conference on Innovation and technology in computer science education - ITiCSE '01*. New York, New York, USA: ACM Press, pp. 109–112. Available at: http://dl.acm.org/citation.cfm?id=377435.377507 [Accessed January 15, 2015].

Dann, W.P., Cooper, S. & Pausch, R., 2011. Learning to Program with Alice. Available at: http://dl.acm.org/citation.cfm?id=2011893 [Accessed January 6, 2015].

DeClue, T., Kimball, J. & Cain, J., 2012. Learning theory in Computer Science 1: an experiment supporting the use of multiple languages. *Journal of Computing Sciences in Colleges*, 27(5), pp.198–204. Available at: http://dl.acm.org/citation.cfm?id=2168874.2168919 [Accessed January 14, 2015].

Dhar, D. & Yammiyavar, P., 2012. Design Approach for E-learning Systems: Should it be User Centered or Learner Centered. In *2012 IEEE Fourth International Conference on Technology for Education*. IEEE, pp. 239–240. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6305982 [Accessed January 19, 2015].

Donner, J. & Toyama, K., 2009. Persistent themes in ICT4D Research : priorities for inter-methodological exchange. *Proceedings of the 57th session of the International Statistics Institute*, (June), pp.1–10.

Duncombe, R., 2010. Mobiles for development research: Quality and Impact. In *M4D 2010*. Available at: http://kau.diva-portal.org/smash/get/diva2:357565/FULLTEXT01#page=102.

Edwards, S.H., Tilden, D.S. & Allevato, A., 2014. Pythy. In *Proceedings of the 45th ACM technical symposium on Computer science education - SIGCSE '14*. New York, New York, USA: ACM Press, pp. 641–646. Available at: http://dl.acm.org/citation.cfm?id=2538862.2538977 [Accessed January 6, 2015].

Elias, T., 2011. Universal instructional design principles for mobile learning. *The International Review of Research in Open and Distributed Learning*, 12(2), pp.143–156. Available at: http://www.irrodl.org/index.php/irrodl/article/view/965/1751 [Accessed January 19, 2015].

Elliott, A.C. & Woodward, W.A., 2007. *Statistical Analysis Quick Reference Guidebook: With SPSS Examples*, SAGE Publications. Available at: http://books.google.com/books?hl=en&lr=&id=SOsX0IbNxeIC&pgis=1 [Accessed December 9, 2014].

Ertmer, P.A. & Newby, T.J., 2008. Behaviorism, Cognitivism, Constructivism: Comparing Critical Features from an Instructional Design Perspective. *Performance Improvement Quarterly*, 6(4), pp.50–72. Available at: http://doi.wiley.com/10.1111/j.1937-8327.1993.tb00605.x [Accessed March 24, 2014].

Esper, S., Simon, B. & Cutts, Q., 2012. Exploratory homeworks: An Active Learning Tool for Textbook Reading. In *Proceedings of the ninth annual international conference on International computing education research - ICER '12*. Auckland, New Zealand: ACM Press, p. 105. Available at: http://dl.acm.org/citation.cfm?id=2361276.2361297 [Accessed March 22, 2014].

Farooq, M.S. et al., 2014. An evaluation framework and comparative analysis of the widely used first programming languages. *PloS one*, 9(2), p.e88941. Available at: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0088941#pone.0088941-Stroustrup1 [Accessed January 14, 2015].

Fosnot, C.T., 2005. *Constructivism: Theory, Perspectives, and Practice, Second Edition*, Teachers College Press.

Fosnot, C.T. & Randall, S.P., 1996. Constructivism: A Psychological Theory of Learning. In *Constructivism: Theory, perspectives, and practice*. pp. 8–33. Available at: http://rsperry.com/fosnotandperry.pdf [Accessed March 24, 2014].

Garner, S., 2004. The CLOZE procedure and the learning of programming. In *8th WSEAS International Conference on COMPUTERS*. Available at: http://www.wseas.us/e-library/conferences/athens2004/papers/487-324.pdf [Accessed December 29, 2014].

Gaspar, A. & Langevin, S., 2007. Restoring "coding with intention" in introductory programming courses. In *Procedding of the 8th ACM SIG-information conference on Information technology education - SIGITE '07*. New York, New York, USA: ACM Press, p. 91. Available at: http://dl.acm.org/citation.cfm?id=1324302.1324323 [Accessed January 14, 2015].

Girault, I. & D'Ham, C., 2013. Scaffolding a Complex Task of Experimental Design in Chemistry with a Computer Environment. *Journal of Science Education and Technology*, 23(4), pp.514–526. Available at: http://link.springer.com/10.1007/s10956-013-9481-5 [Accessed August 13, 2014].

Grandell, L. et al., 2006. Why complicate things?: introducing programming in high school using Python. , pp.71–80. Available at: http://dl.acm.org/citation.cfm?id=1151869.1151880 [Accessed January 14, 2015].

Guo, P., 2014. Python is Now the Most Popular Introductory Teaching Language at Top U.S. Universities. Available at: http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-us-universities/fulltext [Accessed March 22, 2015].

Guzdial, M. et al., 1995. Learner-centered system design. In *Proceedings of the conference on Designing interactive systems processes, practices, methods, & techniques - DIS '95*. New York, New York, USA: ACM Press, pp. 143–147. Available at: http://dl.acm.org/citation.cfm?id=225434.225450 [Accessed January 27, 2015].

Guzdial, M., 2011. Predictions on Future CS1 Languages. *Computer Education Blog*. Available at: https://computinged.wordpress.com/2011/01/24/predictions-on-future-cs1-languages/ [Accessed March 22, 2015].

Guzdial, M., 1994. Software-Realized Scaffolding to Facilitate Programming for Science Learning. *Interactive Learning E*, 4(1), pp.001–044. Available at: http://www.tandfonline.com.ezproxy.uct.ac.za/doi/abs/10.1080/1049482940040101#.Uy5z16iSwdY [Accessed March 23, 2014].

Guzdial, M. et al., 1998. Supporting Programming and Learning-to-Program with an Integrated CAD and Scaffolding Workbench. *Interactive Learning Environments*, 6(1-2), pp.143–179. Available at: http://www.tandfonline.com/doi/abs/10.1076/ilee.6.1.143.3609 [Accessed February 19, 2014].

Guzdial, M., 2015. What's the best way to teach computer science to beginners? *Communications of the ACM*, 58(2), pp.12–13. Available at: http://dl.acm.org/ft_gateway.cfm?id=2714488&type=html [Accessed March 24, 2015].

Hannah, B., 2010. *Information at the Grassroots: Analyzing the media use and communication habits of Kenyans to support effective development*, Available at: http://www.audiencescapes.org/sites/default/files/AudienceScapes Kenya Survey Research Report 2010.pdf.

Harmon, M., 2011. *t-Tests in Excel - The Excel Statistical Master*, Mark Harmon. Available at: http://books.google.com/books?hl=en&lr=&id=C1OHSbQUvAsC&pgis=1 [Accessed November 25, 2014].

Hashim, A., 2007. *Mobile Technology for Learning Java Programming - Design and Implementation of a Programming Tool for VISCOS Mobile*. University of Joensuu. Available at: ftp://193.167.42.127/pub/Theses/2007_MSc_Hahsim_Ahmed.pdf [Accessed March 22, 2014].

Holt, R.C. & Cordy, J.R., 1988. The Turing programming language. *Communications of the ACM*, 31(12), pp.1410–1423. Available at: http://dl.acm.org/citation.cfm?id=53580.53581 [Accessed January 15, 2015].

Hornyak, T., 2014. Android grabs record 85 percent smartphone share. *PCWorld*. Available at: http://www.pcworld.com/article/2460020/android-grabs-record-85-percent-smartphone-share.html [Accessed January 28, 2015].

Hristova, M. et al., 2003. Identifying and correcting Java programming errors for introductory computer science students. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education - SIGCSE '03*. New York, New York, USA: ACM Press, p. 153. Available at: http://dl.acm.org/citation.cfm?id=611892.611956 [Accessed January 19, 2015].

Hürst, W., Lauer, T. & Nold, E., 2007. A study of algorithm animations on mobile devices. *ACM SIGCSE Bulletin*, 39(1), p.160. Available at: http://dl.acm.org/citation.cfm?id=1227504.1227368 [Accessed January 27, 2015].

Ibrahim, R. et al., 2010. Students Perceptions of Using Educational Games to Learn Introductory Programming. *Computer and Information Science*, 4(1), p.p205. Available at: http://www.ccsenet.org/journal/index.php/cis/article/view/8246 [Accessed January 14, 2015].

Ihantola, P., Helminen, J. & Karavirta, V., 2013. How to study programming on mobile touch devices. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research - Koli Calling '13*. New York, New York, USA: ACM Press, pp. 51–58. Available at: http://dl.acm.org/citation.cfm?id=2526968.2526974 [Accessed February 4, 2014].

Jackson, S.L., Krajcik, J. & Soloway, E., 1998. The design of guided learner-adaptable scaffolding in interactive learning environments. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '98*. Los Angeles CA USA: ACM Press, pp. 187–194. Available at: http://dl.acm.org/citation.cfm?id=274644.274672 [Accessed March 11, 2014].

Jenkins, J. et al., 2012. Perspectives on active learning and collaboration. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education - SIGCSE '12*. New York, New York, USA: ACM Press, p. 185. Available at: http://dl.acm.org/citation.cfm?id=2157136.2157194 [Accessed January 15, 2015].

Jenkins, J., Brannock, E. & Dekhane, S., 2010. JavaWIDE: innovation in an online IDE: tutorial presentation. *Journal of Computing Sciences in Colleges*, 25(5), pp.102–104. Available at: http://dl.acm.org/citation.cfm?id=1747137.1747155 [Accessed January 15, 2015].

Jenkins, T., 2002. On the difficulty of learning to program. In *Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences*. p. Vol 4. Available at: http://www.ics.heacademy.ac.uk/Events/conf2002/tjenkins.pdf [Accessed January 7, 2015].

Jobe, W., 2014. *Do-It-Yourself Learning in Kenya : Exploring mobile technologies for merging non-formal and informal learning*. Department of Computer and Systems Sciences, Stockholm University. Available at: http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A758087&dswid=8277 [Accessed April 6, 2015].

Jones et al., 1999. Contexts for evaluating educational software. *Interacting with Computers*, 11(5), pp.499–516. Available at:

http://iwc.oxfordjournals.org.ezproxy.uct.ac.za/content/11/5/499.short [Accessed March 31, 2014].
Jones, M. et al., 1999. Improving Web interaction on small displays. *Computer Networks*, 31(11-16), pp.1129–1137. Available at: http://www.sciencedirect.com/science/article/pii/S1389128699000134 [Accessed January 19, 2015].
Kafyulilo, A., 2012. Access, use and perceptions of teachers and students towards mobile phones as a tool for teaching and learning in Tanzania. *Education and Information Technologies*, 19(1), pp.115–127. Available at: http://link.springer.com/10.1007/s10639-012-9207-y [Accessed July 30, 2014].
Kirschner, P.A., Sweller, J. & Clark, R.E., 2006. Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching. *Educational Psychologist*, 41(2), pp.75–86. Available at: http://www.tandfonline.com/doi/abs/10.1207/s15326985ep4102_1 [Accessed December 3, 2014].
Kölling, M. et al., 2010. The BlueJ System and its Pedagogy. Available at: http://www.tandfonline.com/doi/abs/10.1076/csed.13.4.249.17496#.VKz77yuVLYg [Accessed January 7, 2015].
Kukulska-Hulme, A., 2005. *Mobile usability and user experience* A. Kukulska-Hulme & J. Traxler, eds., Routledge. Available at: http://books.google.com/books?hl=en&lr=&id=onctUPCDt3wC&pgis=1 [Accessed March 12, 2014].
Kukulska-Hulme, A., 2007. Mobile Usability in Educational Contexts: What have we learnt? *The International Review of Research in Open and Distributed Learning*, 8(2). Available at: http://www.irrodl.org/index.php/irrodl/article/view/356/901 [Accessed January 19, 2015].
Lahtinen, E., Ala-Mutka, K. & Järvinen, H.-M., 2005. A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin*, 37(3), p.14. Available at: http://dl.acm.org/citation.cfm?id=1151954.1067453 [Accessed February 7, 2014].
Lalji, Z. & Good, J., 2008. Designing new technologies for illiterate populations: A study in mobile phone interface design. *Interacting with Computers*, 20(6), pp.574–586. Available at: http://iwc.oxfordjournals.org/content/20/6/574.full [Accessed March 23, 2015].
Lazar, J. & Preece, J., 1999. Designing and implementing Web-based surveys. *Journal of Computer Information Systems*, 39(4), pp.63–68.
Leech, N.L. & Onwuegbuzie, A.J., 2007. A typology of mixed methods research designs. *Quality & Quantity*, 43(2), pp.265–275. Available at: http://link.springer.com/10.1007/s11135-007-9105-3 [Accessed July 11, 2014].
Luchini et al., 2002. Supporting learning in context: extending learner-centered design to the development of handheld educational software. In *Proceedings. IEEE International Workshop on Wireless and Mobile Technologies in Education*. IEEE Comput. Soc, pp. 107–111. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1039230 [Accessed January 27, 2015].
Luchini, K. et al., 2002. Scaffolding in the small. In *CHI '02 extended abstracts on Human factors in computing systems - CHI '02*. New York, New York, USA: ACM Press, p. 792. Available at: http://dl.acm.org/citation.cfm?id=506443.506600 [Accessed February 1, 2015].
Luchini, K. et al., 2002. Supporting learning in context: extending learner-centered design to the development of handheld educational software. In *Proceedings. IEEE International Workshop on Wireless and Mobile Technologies in Education*. IEEE Comput. Soc, pp. 107–111. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1039230 [Accessed December 16, 2014].

Luchini, K., Quintana, C. & Soloway, E., 2004. Design guidelines for learner-centered handheld tools. In *Proceedings of the 2004 conference on Human factors in computing systems - CHI '04*. New York, New York, USA: ACM Press, pp. 135–142. Available at: http://dl.acm.org/citation.cfm?id=985692.985710 [Accessed February 19, 2014].

Luchini, K., Quintana, C. & Soloway, E., 2003. Pocket PiCoMap. In *Proceedings of the conference on Human factors in computing systems - CHI '03*. New York, New York, USA: ACM Press, p. 321. Available at: http://dl.acm.org/citation.cfm?id=642611.642668 [Accessed February 19, 2014].

Maleko, M., 2014. *The mobile social learning environment for novice programmers*. RMIT. Available at: http://researchbank.rmit.edu.au/view/rmit:161169 [Accessed February 1, 2015].

Maleko, M., Hamilton, M. & D'Souza, D., 2012. Novices' perceptions and experiences of a mobile social learning environment for learning of programming. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education - ITiCSE '12*. New York, New York, USA: ACM Press, p. 285. Available at: http://dl.acm.org/citation.cfm?id=2325296.2325364 [Accessed May 31, 2014].

Mbogo, C., Blake, E. & Suleman, H., 2013. A mobile scaffolding application to support novice learners of computer programming. In *Proceedings of the Sixth International Conference on Information and Communications Technologies and Development Notes - ICTD '13 - volume 2*. Cape Town: ACM Press, pp. 84–87. Available at: http://dl.acm.org/citation.cfm?id=2517899.2517941 [Accessed February 19, 2014].

Mbogo, C., Blake, E. & Suleman, H., 2014. Supporting the Construction of Programs on a Mobile Device: A Scaffolding Framework. In *Proceedings of 4th International Conference on M4D Mobile Communication for Development*. Dakar, Senegal, p. 155. Available at: http://people.cs.uct.ac.za/~edwin/MyBib/2014-m4d.pdf [Accessed March 11, 2014].

Miller, B.N. & Ranum, D.L., 2012. Beyond PDF and ePub. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education - ITiCSE '12*. New York, New York, USA: ACM Press, p. 150. Available at: http://dl.acm.org/citation.cfm?id=2325296.2325335 [Accessed January 27, 2015].

Mohamed, S., Hamilton, M. & D'Souza, D., 2011. Understanding novice programmer difficulties via guided learning. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education - ITiCSE '11*. New York, New York, USA: ACM Press, p. 213. Available at: http://dl.acm.org/citation.cfm?id=1999747.1999808 [Accessed March 11, 2014].

Moreno, R., 2004. Decreasing Cognitive Load for Novice Students: Effects of Explanatory versus Corrective Feedback in Discovery-Based Multimedia. *Instructional Science*, 32(1/2), pp.99–113. Available at: http://link.springer.com/10.1023/B:TRUC.0000021811.66966.1d [Accessed March 25, 2015].

Mtebe, J. & Raisamo, R., 2014. Investigating students ' behavioural intention to adopt and use mobile learning in higher education in East Africa Joel S . Mtebe University of Dar es Salaam , Tanzania Roope Raisamo University of Tampere , Finland. *International Journal of Education and Development using Information and Communication Technology*, 10(3), pp.4–20.

Mueller, F. & Hosking, A.L., 2003. Penumbra. In *Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange - eclipse '03*. New York, New York, USA: ACM Press, pp. 65–68. Available at: http://dl.acm.org/citation.cfm?id=965660.965674 [Accessed January 15, 2015].

Ng'ambi, D., 2005. Mobile Dynamic Frequently Asked Questions (DFAQ) for student and learning support. *Mobile Technology: The future of learning in your hands*, pp.116–119.

NIST, 2001. *Common Industry Format for Usability Test Reports*, Available at: http://www.idemployee.id.tue.nl/g.w.m.rauterberg/lecturenotes/common-industry-format.pdf [Accessed April 26, 2014].

Pears, A. & Rogalli, M., 2011. mJeliot. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research - Koli Calling '11*. New York, New York, USA: ACM Press, p. 16. Available at: http://dl.acm.org/citation.cfm?id=2094131.2094135 [Accessed February 13, 2014].

Piaget, J., 1964. Part I: Cognitive development in children: Piaget development and learning. *Journal of Research in Science Teaching*, 2(3), pp.176–186. Available at: http://doi.wiley.com/10.1002/tea.3660020306 [Accessed June 8, 2014].

Pillay, N. & Jugoo, V.R., 2005. An investigation into student characteristics affecting novice programming performance. *ACM SIGCSE Bulletin*, 37(4), p.107. Available at: http://dl.acm.org/citation.cfm?id=1113847.1113888 [Accessed March 24, 2014].

Piteira, M. & Costa, C., 2012. Computer programming and novice programmers. In *Proceedings of the Workshop on Information Systems and Design of Communication - ISDOC '12*. New York, New York, USA: ACM Press, pp. 51–53. Available at: http://dl.acm.org/citation.cfm?id=2311917.2311927 [Accessed May 31, 2014].

Powers, K., Ecott, S. & Hirshfield, L.M., 2007. Through the looking glass. *ACM SIGCSE Bulletin*, 39(1), p.213. Available at: http://dl.acm.org/citation.cfm?id=1227504.1227386 [Accessed January 15, 2015].

Pullan, W., Drew, S. & Tucker, S., 2013. An integrated approach to teaching introductory programming. In *2013 Second International Conference on E-Learning and E-Technologies in Education (ICEEE)*. IEEE, pp. 81–86. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6644352 [Accessed May 11, 2014].

Queirós, R.A.P. & Leal, J.P., 2012. PETCHA. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education - ITiCSE '12*. Haifa, Israel: ACM Press, p. 192. Available at: http://dl.acm.org/citation.cfm?id=2325296.2325344 [Accessed February 19, 2014].

Quintana, C. et al., 2004. A Scaffolding Design Framework for Software to Support Science Inquiry. *Journal of the Learning Sciences*, 13(3), pp.337–386. Available at: http://dx.doi.org/10.1207/s15327809jls1303_4 [Accessed February 4, 2014].

Quintana, C., Fretz, E., et al., 2000. Evaluation criteria for scaffolding in learner-centered tools. In *CHI '00 extended abstracts on Human factors in computing systems - CHI '00*. New York, New York, USA: ACM Press, p. 189. Available at: http://dl.acm.org/citation.cfm?id=633292.633396 [Accessed May 11, 2014].

Quintana, C., Krajcik, J. & Soloway, E., 2002a. A Case Study to Distill Structural Scaffolding Guidelines for Scaffolded Software Environments. In *Proceedings of the SIGCHI conference on Human factors in computing systems Changing our world, changing ourselves - CHI '02*. New York, New York, USA: ACM Press, p. 81. Available at: http://dl.acm.org/citation.cfm?id=503376.503392 [Accessed February 19, 2014].

Quintana, C., Krajcik, J. & Soloway, E., 2000. Exploring a Structure Definition for Learner-Centered Design. In *Fourth international conference of the learning sciences*. Psychology Press, pp. 256–263. Available at: https://books.google.com/books?hl=en&lr=&id=0JM5N9PUZM8C&pgis=1 [Accessed March 23, 2015].

Quintana, C., Krajcik, J. & Soloway, E., 2002b. Scaffolding Design Guidelines for Learner-Centered Software Environments. Available at: http://eric.ed.gov/?id=ED467503 [Accessed January 28, 2015].

Reis, C. & Cartwright, R., 2004. Taming a professional IDE for the classroom. *ACM SIGCSE Bulletin*, 36(1), p.156. Available at: http://dl.acm.org/citation.cfm?id=1028174.971357 [Accessed January 15, 2015].

Rogerson, C. & Scott, E., 2010. The Fear Factor: How It Affects Students Learning to Program in a Tertiary Environment. *Journal of Information Technology Education: Research*, 9(1), pp.147–171. Available at: http://www.editlib.org/p/111361/ [Accessed January 14, 2015].

Roy, K., 2012. App inventor for android. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education - SIGCSE '12*. New York, New York, USA: ACM Press, p. 283. Available at: http://dl.acm.org/citation.cfm?id=2157136.2157222 [Accessed January 16, 2015].

Sauro, J. & Lewis, J.R., 2012. *Quantifying the User Experience*, Elsevier. Available at: http://www.sciencedirect.com/science/article/pii/B9780123849687000023 [Accessed October 26, 2014].

Saye, J. & Brush, T., 2001. The Use of Embedded Scaffolds with Hypermedia-Supported Student-Centered Learning. *Journal of Educational Multimedia and Hypermedia*, 10(4), pp.333–356. Available at: http://www.editlib.org/p/8439/ [Accessed March 22, 2015].

Schunk, Dale, H., 2011. *Learning Theories: An Educational Perspective*, Pearson; 6 edition.

Seifert, J. et al., 2011. Mobidev. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services - MobileHCI '11*. New York, New York, USA: ACM Press, p. 109. Available at: http://dl.acm.org/citation.cfm?id=2037373.2037392 [Accessed January 24, 2015].

Senga, E., 2010. *A Service-Oriented Approach to Implementing an Adaptive User Interface*. Nelson Mandela Metropolitan University.

Shein, E., 2015. Python for beginners. *Communications of the ACM*, 58(3), pp.19–21. Available at: http://dl.acm.org/ft_gateway.cfm?id=2716560&type=html [Accessed March 22, 2015].

Siek, K.A., Rogers, Y. & Connelly, K.H., 2005. Fat finger worries: how older and younger users physically interact with PDAs. In M. F. Costabile & F. Paternò, eds. *Human-Computer Interaction-INTERACT 2005*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 267–280. Available at: http://www.springerlink.com/index/10.1007/11555261 [Accessed January 19, 2015].

Soloway, E. et al., 1996. Learning theory in practice: case studies of learner-centered design. In *Proceedings of the SIGCHI conference on Human factors in computing systems common ground - CHI '96*. New York, New York, USA: ACM Press, pp. 189–196. Available at: http://dl.acm.org/ft_gateway.cfm?id=238476&type=html [Accessed May 31, 2014].

Soloway, E., Guzdial, M. & Hay, K.E., 1994. Learner-centered design: the challenge for HCI in the 21st century. *interactions*, 1(2), pp.36–48. Available at: http://dl.acm.org/citation.cfm?id=174809.174813 [Accessed May 31, 2014].

Sphere Research Labs, 2010. Ideone ™ API. , pp.1–11.

Storey, M.-A. et al., 2003. Improving the usability of Eclipse for novice programmers. In *Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange - eclipse '03*. New York, New York, USA: ACM Press, pp. 35–39. Available at: http://dl.acm.org/citation.cfm?id=965660.965668 [Accessed January 4, 2015].

Sun-Microsystems, 1997. Java Code Conventions. Available at: http://www.oracle.com/technetwork/java/codeconventions-150003.pdf [Accessed January 19, 2015].

Svensson, J. & Wamala, C., 2012. M4D : Mobile Communication for Development. , p.21. Available at: http://www.diva-portal.org/smash/record.jsf?pid=diva2:549742 [Accessed March 11, 2014].

Taylor, J., 2006. Evaluating mobile learning: What are appropriate methods for evaluating learning in mobile enviroments. In *Big issues in mobile learning*. pp. 24–26.

Tillmann, N. et al., 2011. TouchDevelop. In *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software - ONWARD '11*. New York, New York, USA: ACM Press, p. 49. Available at: http://dl.acm.org/citation.cfm?id=2048237.2048245 [Accessed February 8, 2014].

Traxler, J., 2011. Learning with Mobile Devices Somewhere Near the Bottom of the Pyramid « Educational Technology Debate. Available at: http://edutechdebate.org/affordable-technology/learning-with-mobile-devices-somewhere-near-the-bottom-of-the-pyramid/ [Accessed March 19, 2015].

Traxler, J. & Kukulska-Hulme, A., 2005. Evaluating Mobile Learning: Reflections on Current Practice. In *mLearn 2005*. Available at: http://oro.open.ac.uk/12819/1/mlearn05_Traxler&Kukulska-Hulme.pdf [Accessed January 26, 2015].

Traxler, J. & Leach, J., 2006. Innovative and Sustainable Mobile Learning in Africa. In *2006 Fourth IEEE International Workshop on Wireless, Mobile and Ubiquitous Technology in Education (WMTE'06)*. IEEE, pp. 98–102. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4032531 [Accessed January 15, 2015].

Traxler, J. & Vosloo, S., 2014. Introduction: The prospects for mobile learning. *PROSPECTS*, 44(1), pp.13–28. Available at: http://link.springer.com/10.1007/s11125-014-9296-z [Accessed March 24, 2015].

Vavoula, G. & Sharples, M., 2009. Meeting the Challenges in Evaluating Mobile Learning: A 3-level Evaluation Framework. *International Journal of Mobile and Blended Learning*, 1(2), pp.54–75. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.357.2367&rep=rep1&type=pdf [Accessed April 17, 2014].

Vihavainen, A. et al., 2013. Scaffolding students' learning using test my code. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education - ITiCSE '13*. Canterbury, England: ACM Press, p. 117. Available at: http://dl.acm.org/citation.cfm?id=2462476.2462501 [Accessed February 19, 2014].

Vihavainen, A., Paksula, M. & Luukkainen, M., 2011. Extreme apprenticeship method in teaching programming for beginners. In *Proceedings of the 42nd ACM technical symposium on Computer science education - SIGCSE '11*. New York, New York, USA: ACM Press, p. 93. Available at: http://dl.acm.org/citation.cfm?id=1953163.1953196 [Accessed February 19, 2014].

Vygotsky, L.S., 1978. *Mind in Society: The Development of Higher Psychological Processes*, Harvard University Press.

Wagner, A. et al., 2013. Using app inventor in a K-12 summer camp. In *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*. New York, New York, USA: ACM Press, p. 621. Available at: http://dl.acm.org/citation.cfm?id=2445196.2445377 [Accessed January 16, 2015].

Watson, C. & Li, F.W.B., 2014. Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation & technology in computer science education - ITiCSE '14*. New York, New York, USA: ACM Press, pp. 39–44. Available at: http://dl.acm.org/citation.cfm?id=2591708.2591749 [Accessed January 14, 2015].

Winslow, L.E., 1996. Programming pedagogy---a psychological overview. *ACM SIGCSE Bulletin*, 28(3), pp.17–22. Available at: http://dl.acm.org/citation.cfm?id=234867.234872 [Accessed January 14, 2015].

Winterbottom, C., 2010. *VRBridge: A Constructivist Approach to Supporting Interaction Design and End-User Authoring in Virtual Reality*. University of Cape Town. Available at: http://pubs.cs.uct.ac.za/archive/00000607/01/CaraWinterbottomPhDThesisFinal.pdf [Accessed March 24, 2014].

Winterbottom, C. & Blake, E., 2004. Designing a VR interaction authoring tool using constructivist practices. In *Proceedings of the 3rd international conference on Computer graphics, virtual reality, visualisation and interaction in Africa - AFRIGRAPH '04*. New York, New York, USA:

ACM Press, p. 67. Available at: http://dl.acm.org/citation.cfm?id=1029949.1029961 [Accessed June 10, 2014].
Wolber, D., 2011. App inventor and real-world motivation. In *Proceedings of the 42nd ACM technical symposium on Computer science education - SIGCSE '11*. New York, New York, USA: ACM Press, p. 601. Available at: http://dl.acm.org/citation.cfm?id=1953163.1953329 [Accessed January 16, 2015].
Wolz, U. et al., 2009. Starting with scratch in CS 1. *ACM SIGCSE Bulletin*, 41(1), p.2. Available at: http://dl.acm.org/citation.cfm?id=1539024.1508869 [Accessed January 14, 2015].
Wood, D., Bruner, J.S. & Ross, G., 1976. The Role of Tutoring in Problem Solving. *Journal of Child Psychology and Psychiatry*, 17(2), pp.89–100. Available at: http://doi.wiley.com/10.1111/j.1469-7610.1976.tb00381.x [Accessed March 11, 2014].
Yizengaw, T., 2008. *Challenges of Higher Education in Africa and Lessons of Experience for the Africa-US Higher Education Collaboration Initiative*, Available at: http://www.uhasselt.be/Documents/UHasselt_EN/International/Lezing N-Z 2013/challegnes_in_africa.pdf [Accessed January 14, 2015].
Yuh-Shyan Chen et al., 2002. A mobile scaffolding-aid-based bird-watching learning system. In *Proceedings. IEEE International Workshop on Wireless and Mobile Technologies in Education*. IEEE Comput. Soc, pp. 15–22. Available at: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1039216 [Accessed April 26, 2014].
Zimmerman, D. & Yohon, T., 2009. Small-screen interface design: Where are we? Where do we go? In *2009 IEEE International Professional Communication Conference*. IEEE, pp. 1–5. Available at: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=5208667 [Accessed December 16, 2014].

APPENDICES

Appendix A: Table of the Scaffolding Framework

This table describes in detail the types of cognitive challenges that that face learners, specifying scaffolding type and guideline and scaffolding technique that can be implemented on a mobile device to address learner challenges in programming.

Type of cognitive challenge	*Specific learning challenge*	*Scaffolding type*	*Scaffolding guideline*	*Scaffolding strategy that can be implemented on a Mobile Device*
Sense Making	• Unclear error messages when debugging • Debugging is sometimes frustrating • Sometimes it's hard to figure out what the error message is trying to tell you when you try to run the program • It is sometimes difficult to understand exactly what is being asked or how to correct your program when it does not run as intended	Supportive	Use representation and language that bridge learners' understanding of programming.	Prompt the learner as soon as they make a mistake in a piece of code instead of having to wait till they compile the program. Use clear and easy to understand language in the prompt.
	• Constructing logic from programs is difficult • I struggle in thinking logically	Reflective/ Intrinsic	Structure task and functionality by restricting a complex task by setting proper boundaries for learners. Use representations that learners can inspect in different ways to reveal important	Force the learner to complete 'first level' tasks before 'unlocking' 'second level tasks' and so on. An example of a first level task would be in declaring a class. A second level task would be to complete the 'main method'. A third level task could be to complete a method that will be called from the main method.

Type of cognitive challenge	*Specific learning challenge*	*Scaffolding type*	*Scaffolding guideline*	*Scaffolding strategy that can be implemented on a Mobile Device*
			properties about underlying data.	Enable the learner to 'dive in' to specific program parts while they can also 'step out' and view the full program. Provide examples that are relevant to the part of the program being worked on
	• The ability to join codes or to build objects alone combining to form classes and finally a system to do a certain task	Supportive/Intrinsic	Organize the mobile strategy around the semantics of the programming language.	Provide general program structures such as keywords and opening and closing braces that a learner can edit. Editing should be restricted to be within the allowed syntax of that program part. For example, in a class declaration, **public class sum** , the learner can edit sum to their desired name but not be able to add any other code after sum. Decompose the program into parts and present the program structure as it appears on a PC IDE. Once decomposed, provide a visual repre-sentation of accessible program parts to enable a learner to have an overview of a program. For example, in Java, program parts are 'the header comments', 'import statements', 'class declaration', 'methods', 'main

Type of cognitive challenge	*Specific learning challenge*	*Scaffolding type*	*Scaffolding guideline*	*Scaffolding strategy that can be implemented on a Mobile Device*
				class'. Presenting the program on the mobile device in this 'chunked' format could assist a learner in logical thinking
	• The simple yet confusing rules of programming i.e, initialization, breaks ups with semicolons on parameters, constructors and how they are used etc.	Supportive	Use representation and language that bridge learners' understanding of programming.	Provide steps to complete program parts. Implement determination of the program part that the learner is attempting to complete. For example, if writing code for a method within a constructor like '**return num'**, the learner can be prompted that the piece of code does not belong in a constructor but in a method.
	• Sometimes I am not sure how the syntax should be done and there are no internet resources to help me • When I was learning Java I was struggling with the syntax which made it hard for me to work the logic part because I wasn't sure about other functions usage	Supportive	Use representation and language that bridge learners' understanding of programming.	Provide steps to complete program parts. Guiding the learner through subtasks by providing messages that appear when appropriate.
Process Management	• I prefer learning through video tutorials but the internet is either too slow or expensive	Supportive	Embed expert guidance about programming practices.	Embed information that a learner can use in the absence of other resources. These include the steps for completing a task and

Type of cognitive challenge	*Specific learning challenge*	*Scaffolding type*	*Scaffolding guideline*	*Scaffolding strategy that can be implemented on a Mobile Device*
				related examples that they can access with minimal cost. Allow learner to save the examples that they wish to view later. Enable storage of programs on the mobile device that they can reuse on a PC in the classroom or at home.
	• Poor presentation of programs by lecturer without sufficient time to practice the code	Supportive	Embed expert guidance about programming practices.	Complement classroom learning by providing assistance in completing the program task.
			Embed expert guidance about programming practices.	Complement classroom learning by providing assistance in completing the program task.
	• It takes too much time to code programs • Finding ways to accomplish a task in the shortest way possible	Supportive/Intrinsic	Organize the mobile strategy around the semantics of the programming language. Automatically handle routine tasks.	Decomposing the program into parts gives quick access to the parts the learner needs to work on. Automatically complete program parts such as keywords and opening and closing braces.
			Automatically handle routine tasks.	Automatically complete program parts such as keywords and opening and closing braces.
	• Learning programming for the first time at university level has been a big challenge espe-cially for me	Supportive/Intrinsic	Provide structure for complex tasks and functionality.	Decomposing the program into parts would assist a novice learner to logically follow the flow of a program

Type of cognitive challenge	*Specific learning challenge*	*Scaffolding type*	*Scaffolding guideline*	*Scaffolding strategy that can be implemented on a Mobile Device*
	because I am from a rural area with no computer back-ground			Offer context specific help. For example if a learner is working on recursion, they are scaffolded using expert knowledge on recursion. Enable the scaffolding to fade away as the learner progresses and offer more 'advanced features'. Provide periodic 'self-assessment' so that the learner can test themselves.
	• Programming is fun but most students like me loose interest quickly.	Reflection	Facilitate ongoing articulation and reflection during program construction.	Enable the scaffolding to fade away as the learner progresses and offer more 'advanced features'. Provide periodic 'self-assessment' so that the learner can test themselves.
	• Coding is very challenging especially when using java programming language • The language is very strict and you really have to know specific instructions to accomplish a task	Supportive	Provide structure for complex tasks and functionality.	Decomposing the program into parts would assist a novice learner to logically follow the flow of a program
Articulation and Reflec-tion	• Lack of documentation and practical examples	Reflective	Embed expert guidance about programming practices.	Provide examples that are relevant to the program part being completed.
	• Translating an algorithm into code, sometimes I	Supportive/Intrinsic	Use representation and language that	Provide steps on the correct syntax to

Type of cognitive challenge	*Specific learning challenge*	*Scaffolding type*	*Scaffolding guideline*	*Scaffolding strategy that can be implemented on a Mobile Device*
	manage to solve the problem in my head but then communicating it to Python can be a bit of a challenge		bridge learners' understanding of programming.	complete a program part
	• Moving from Python to Java was a challenge to me because when you are programming in Java, you have to be more specific in terms of variable types and return value types	Supportive	Embed expert guidance about programming practices.	Provide steps on how to complete subtasks

Appendix B: Summary of Scaffolding Design Framework

Table source (Quintana et al. 2004)

TABLE 1
Summary of the Scaffolding Design Framework

Scaffolding Guidelines	*Scaffolding Strategies*
Science inquiry component: Sense making	
Guideline 1: Use representations and language that bridge learners' understanding	1a: Provide visual conceptual organizers to give access to functionality
	1b: Use descriptions of complex concepts that build on learners' intuitive ideas
	1c: Embed expert guidance to help learners use and apply science content
Guideline 2: Organize tools and artifacts around the semantics of the discipline	2a: Make disciplinary strategies explicit in learners' interactions with the tool
	2b: Make disciplinary strategies explicit in the artifacts learners create
Guideline 3: Use representations that learners can inspect in different ways to reveal important properties of underlying data	3a: Provide representations that can be inspected to reveal underlying properties of data
	3b: Enable learners to inspect multiple views of the same object or data
	3c: Give learners "malleable representations" that allow them to directly manipulate representations
Science inquiry component: Process management	
Guideline 4: Provide structure for complex tasks and functionality	4a: Restrict a complex task by setting useful boundaries for learners
	4b: Describe complex tasks by using ordered and unordered task decompositions
	4c: Constrain the space of activities by using functional modes
Guideline 5: Embed expert guidance about scientific practices	5a: Embed expert guidance to clarify characteristics of scientific practices
	5b: Embed expert guidance to indicate the rationales for scientific practices
Guideline 6: Automatically handle nonsalient, routine tasks	6a: Automate nonsalient portions of tasks to reduce cognitive demands
	6b: Facilitate the organization of work products
	6c: Facilitate navigation among tools and activities
Science inquiry component: Articulation and reflection	
Guideline 7: Facilitate ongoing articulation and reflection during the investigation	7a: Provide reminders and guidance to facilitate productive planning
	7b: Provide reminders and guidance to facilitate productive monitoring
	7c: Provide reminders and guidance to facilitate articulation during sense-making
	7d: Highlight epistemic features of scientific practices and products

Appendix C: Ethical Clearances

Appendix C1: Ethical clearance from University of Cape Town

Department of Environmental and Geographical Science
University of Cape Town
RONDEBOSCH 7701
South Africa

e-mail: Michael.meadows@uct.ac.za
phone : + 27 21 650 2873
fax : +27 21 650 3791

14th April 2013

Ms Charity Mbogo
Department of Computer Science
University of Cape Town
chao.mbogo@uct.ac.za

Dear Ms Mbogo

Investigating Mobile- Based Strategies to Support Learning of Computer Programming beyond the classroom in higher education: Case of Kenya and South Africa

I am pleased to inform you that, having scrutinized the details of your above-named application for research ethics clearance, the Faculty of Science Research Ethics Committee has approved your proposal in terms of its attention to ethical principles.

Your approval code for the project is: SFREC 012_2013

I wish you success in the work involved.

Yours sincerely

M Meadows

Michael E Meadows
Professor and Head of Department
Chair: Science Faculty Research Ethics Committee

Appendix C2: Permission to access learners at University of Cape Town

	RESEARCH ACCESS TO STUDENTS	DSA 100

NOTES

1. This form must be FULLY completed by applicants that want to access UCT students for the purpose of research.
2. Return the completed application form together with your research proposal to: Moonira.Khan@uct.ac.za; or deliver to: Attention: Executive Director, Department of Student Affairs, North Lane, Steve Biko Students' Union, Room 7.22, Upper Campus, UCT.
3. The turnaround time for a reply is approximately 10 working days.
4. NB: It is the responsibility of the researcher/s to apply for and to obtain ethical clearance and access to staff and/or students, respectively to the (a) Faculty's 'Ethics in Research Committee' (EiRC) for ethics approval, and (b) Executive Director, HR for approval to access staff for research purposes and the (c) Executive Director, Student Affairs for approval to access students for research purposes.
5. For noting, a requirement of UCT (according to Senate policy) is that items (1) and (4) apply even if prior clearance has been obtained by the researcher/s from any other institution.

SECTION A: RESEARCH APPLICANT/S DETAILS

Position	Staff / Student No	Title and Name	Contact Details (Email / Cell / land line)
A.1 Student Number	MBGCHA002	MS CHARITY MBOGO	chao.mbogo@uct.ac.za
A.2 Academic / PASS Staff No.			
A.3 Visiting Researcher ID No.			
A.4 University at which a student or employee	UCT	Address if *not* UCT:	
A.5 Faculty/ Department/School	Computer Science		
A.6 APPLICANTS DETAILS If different from above	Title and Name	Tel.	Email

SECTION B: RESEARCHER/S SUPERVISOR/S DETAILS

Position	Title and Name	Tel.	Email
B.1 Supervisor	Prof Edwin Blake		edwin@cs.uct.ac.za
B.2 Co-Supervisor/s (a)	Prof Hussein Suleman		hussein@cs.uct.ac.za

SECTION C: APPLICANT'S RESEARCH STUDY FIELD AND APPROVAL STATUS

C.1 Degree (if a student)	PhD Computer Science
C.2 Research Project Title	Investigating mobile-based strategies to support learning of computer programming beyond the classroom in higher education
C.3 Research Proposal	Attached: Yes ☒ No ☐
C.4 Target population	Undergraduate computer programming students in higher education
C.5 Lead Researcher details	If different from applicant:
C6. Will use research assistant/s	Yes ☐ No ☒
C.7 Research Methodology and Informed consent:	Research methodology: Focus groups, Questionnaires, Mobile Prototype Probe and Interviews. Informed consent: YES. The students will be advised to participate willingly in the study.
C.8 Ethics clearance status from UCT's Ethics in Research Committee (EiRC)	Approved by the EiRC: Yes ☒ No ☐ Awaiting response : ☐ If yes, attach copy and state the date and ref. no of EiRC approval: 14/04/2013 SFREC 012_2013 Date of application if awaiting response:

SECTION D: APPLICANT/S APPROVAL STATUS FOR ACCESS TO STUDENTS FOR RESEARCH PURPOSE

(To be completed by the ED, DSA or Nominee)

	Approved / With Terms / Not	* Conditional approval with terms	Applicant/s Ref. No.:	
D.1 APPROVAL STATUS	Yes ■	(a) Access to students for this research study must only be undertaken after written ethics approval has been obtained. (b) In event any ethics conditions are attached, these must be complied with before access to students.	MBGCHA002 / Charity Mbogo	
D.2 APPROVED BY:	Designation	Name	Signature	Date
	Executive Director Department of Student Affairs	*Ms MBM Khan*		*22 April 2013*

Appendix C3: Ethical clearance from Kenya Methodist University

KENYA METHODIST UNIVERSITY

P. O. Box 267 Meru - 60200, Kenya
Tel: 254-064-30301/31229/30367/31171
Fax: 254-64-30162
Email: info@kemu.ac.ke

Dear Ms. Chao Mbogo,

SUB: REQUEST TO COLLECT DATA

I am in receipt of your letter dated April 22nd, 2013 requesting for permission to administer questionnaires to KeMU staff and students under your research project titled **"Investigating mobile-based strategies to support learning of computer programming beyond the classroom in higher education. Case of Kenya and South Africa."**

Provisional permission is hereby granted to you to collect data in accordance with the content of your project proposal. This is subject to tabling your document in the full board of Scientific and Ethics Review Committee (SERC)

Thank You,

Prof. G. W. Odhiambo-Otieno, PhD
Secretary – Scientific and Ethics Review Committee and Dean, Research and Development

Cc: Prof. Alice Mutungi – Chair of SERC

/AM

Appendix D: Consent form signed by learners before participating in study

CONSENT FORM **Form 1**

Full title of Project: Scaffolding the construction of programs on a mobile device

Name, position and contact address of Researcher:
Chao Mbogo
Computer Science Department
University of Cape Town
chao.mbogo@uct.az.za

Purpose of the study: Scaffolding refers to support provided to a learner in order for them to complete a given task. This study seeks to investigate the effectiveness of scaffolding strategies while constructing a program on a mobile device, and to investigate how learners of programming use the scaffolds to construct programs on the mobile device.

		Please Tick Box
1.	I confirm that I have understood the purpose of the above study as explained by the researcher.	☐
2.	I understand that my participation is **voluntary** and that I am free to **withdraw** at any time, without giving reason.	☐
3.	I agree to take part in the above study.	☐
4.	I agree to the interview / focus group / consultation being **audio** recorded	☐
5.	I agree to the interview / focus group / consultation being **image/video** recorded	☐
6.	I agree to the use of anonymised quotes in publications	☐
7.	I agree to the use of my anonymised feedback as part of the results of the study	☐

Name of Participant Date Signature

Appendix E: Questionnaires

Appendix E1: Experiment 1 questionnaire

Evaluation Questionnaire I - Usability and User Experience Testing of ScafFold

This questionnaire measures the interface usability.

Thank you for taking the time to complete this evaluation questionnaire for the mobile learning intervention, ScafFold.

This questionnaire takes at most 10 minutes to complete, when completed at a go.

Any questions regarding completion of the questionnaire can be directed to chao.mbogo@uct.ac.za

There are 19 questions in this survey

Demographic

1 [Course]What course(s) are you studying? *

Please choose **all** that apply:

- ☐ Computer Science
- ☐ Computer Information Systems
- ☐ Information Systems
- ☐ Business Information Technology
- ☐ Other: [ ]

2 [Qualification]What degree are you studying towards? *

Please choose **only one** of the following:

- ○ Masters
- ○ Bachelors
- ○ Honors
- ○ Diploma
- ○ Other [ ]

Evaluation - Interface Usability

3 [Attractive]ScafFold is attractive. *

Please choose **only one** of the following:

- ○ Strongly disagree
- ○ Disagree
- ○ Neither agree nor disagree
- ○ Agree
- ○ Strongly agree

Make a comment on your choice here:

4 [Ease of Use]The interface is easy to follow and use. *

Please choose **only one** of the following:

- ○ Strongly disagree
- ○ Disagree
- ○ Neither agree nor disagree
- ○ Agree
- ○ Strongly agree

Make a comment on your choice here:

5 [Feedback]ScafFold has feedback that give sufficient information to the user on all screens. *

Please choose **only one** of the following:

- ○ Strongly disagree
- ○ Disagree
- ○ Neither agree nor disagree
- ○ Agree
- ○ Strongly agree

Make a comment on your choice here:

6 [language]Labels and textual information are presented in farmiliar language. *

Please choose **only one** of the following:

- ○ Strongly disagree
- ○ Disagree
- ○ Neither agree nor disagree
- ○ Agree
- ○ Strongly agree

Make a comment on your choice here:

7 [User Control]ScafFold has clear ways of undoing/redoing an action in case the user makes a mistake *

Please choose **only one** of the following:

- ○ Strongly disagree
- ○ Disagree
- ○ Neither agree nor disagree
- ○ Agree
- ○ Strongly agree

Make a comment on your choice here:

8 [Consistency]There is consistency in use of words and actions between the windows. *

Please choose **only one** of the following:

- ○ Strongly disagree
- ○ Disagree
- ○ Neither agree nor disagree
- ○ Agree
- ○ Strongly agree

Make a comment on your choice here:

9 [Help]There is enough guidance/assistance to inform users of what to do. *

Please choose **only one** of the following:

- ○ Strongly disagree
- ○ Disagree
- ○ Neither agree nor disagree
- ○ Agree
- ○ Strongly agree

Make a comment on your choice here:

10 [irrelevant]There is no irrelevant information displayed on the screens. *

Please choose **only one** of the following:

- ○ Strongly disagree
- ○ Disagree
- ○ Neither agree nor disagree
- ○ Agree
- ○ Strongly agree

Make a comment on your choice here:

11 [Textual]All textual aspects are legible and recognizable. *

Please choose **only one** of the following:

- ○ Strongly disagree
- ○ Disagree
- ○ Neither agree nor disagree
- ○ Agree
- ○ Strongly agree

Make a comment on your choice here:

User Experience

12 [Negative]List the most negative interface usability features of ScaffoldX. *

Please write your answer here:

13 [Positive]List the most positive interface usability features of ScaffoldX. *

Please write your answer here:

14 [Fitting]Do you think ScafFold would fit in the learning environment to effectively support novice programming learners outside the classroom? *

Please choose **only one** of the following:

- ○ Yes
- ○ No

15 [FittingN]Why do you think ScafFold would not fit in the learning environment? *

Only answer this question if the following conditions are met:
° ((Fitting.NAOK == "N"))

Please write your answer here:

16 [Errors]Did any errors occur during your use of ScafFold that stopped it from working? *

Please choose **only one** of the following:

- ○ Yes
- ○ No

17 [ErrorsY]Briefly describe the error(s) *

Only answer this question if the following conditions are met:
° ((Errors.NAOK == "Y"))

Please write your answer here:

18 [Features]To what extent do you agree that the following features of ScafFold support the construction of programs on a mobile device? *

Please choose the appropriate response for each item:

	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
Presentation of a program in chunks on Main Interface	○	○	○	○	○
Use of Tabs at the top of the screen	○	○	○	○	○
Completion of the program one part at a time in Code Editor	○	○	○	○	○
Steps on how to interact with the application to complete a program	○	○	○	○	○
Hints	○	○	○	○	○
Error prompts	○	○	○	○	○
Dialog Selection of System.out, For-loop and Buffered Reader	○	○	○	○	○
Provision of some default code to edit	○	○	○	○	○
Examples	○	○	○	○	○
Ability to View Full Program any time	○	○	○	○	○
Compiling and running the program	○	○	○	○	○
Ability to store programs on mobile device	○	○	○	○	○
Availability of a tutorial	○	○	○	○	○

19 [Comments]Add any other comments you may have on ScafFold.

Please write your answer here:

Appendix E2: Experiment 2 and 3 questionnaire

Post Experiment Session Questionnaire - Experiments 3 and 4

This questionnaires gathers data on user's opinion on use of the application.

Thank you for taking the time to complete this evaluation questionnaire for the mobile learning intervention, ScafFold.

Please give as much feedback as possible in the available textboxes for each question. Kindly complete all questions truthfully in order to preserve the integrity of the results.

This questionnaire takes at most 10 minutes to complete, when completed at a go.

Any questions regarding completion of the questionnaire can be directed to chao.mbogo@uct.ac.za

There are 6 questions in this survey

Demographic

This section collects data about your background

1 [Institution]At which institution do you study *

Please select one answer

Please choose **all** that apply:

- ☐ Nairobi Dev School
- ☐ UWC
- ☐ KeMU
- ☐ UoN
- ☐ Other: _____

2 [Degree]What degree are you studying towards? *

Please select one answer

Please choose **all** that apply:

- ☐ Certificate
- ☐ Diploma
- ☐ Bachelors
- ☐ Other: _____

3 [Course]What course are you studying? *

Please select one answer

Please choose **all** that apply:

- ☐ Computer Science
- ☐ Computer Information Systems
- ☐ Information Systems
- ☐ Business Information Technology
- ☐ Other:

Opinion

This section explores your opinion on use of the application

4 [Support]Indicate the features of the application that most supported your construction of programs on the mobile device. Give as much detail as you can. *

Please write your answer here:

5 [Additions]In your opinion, is there anything missing from the application that would support construction of programs on a mobile device?

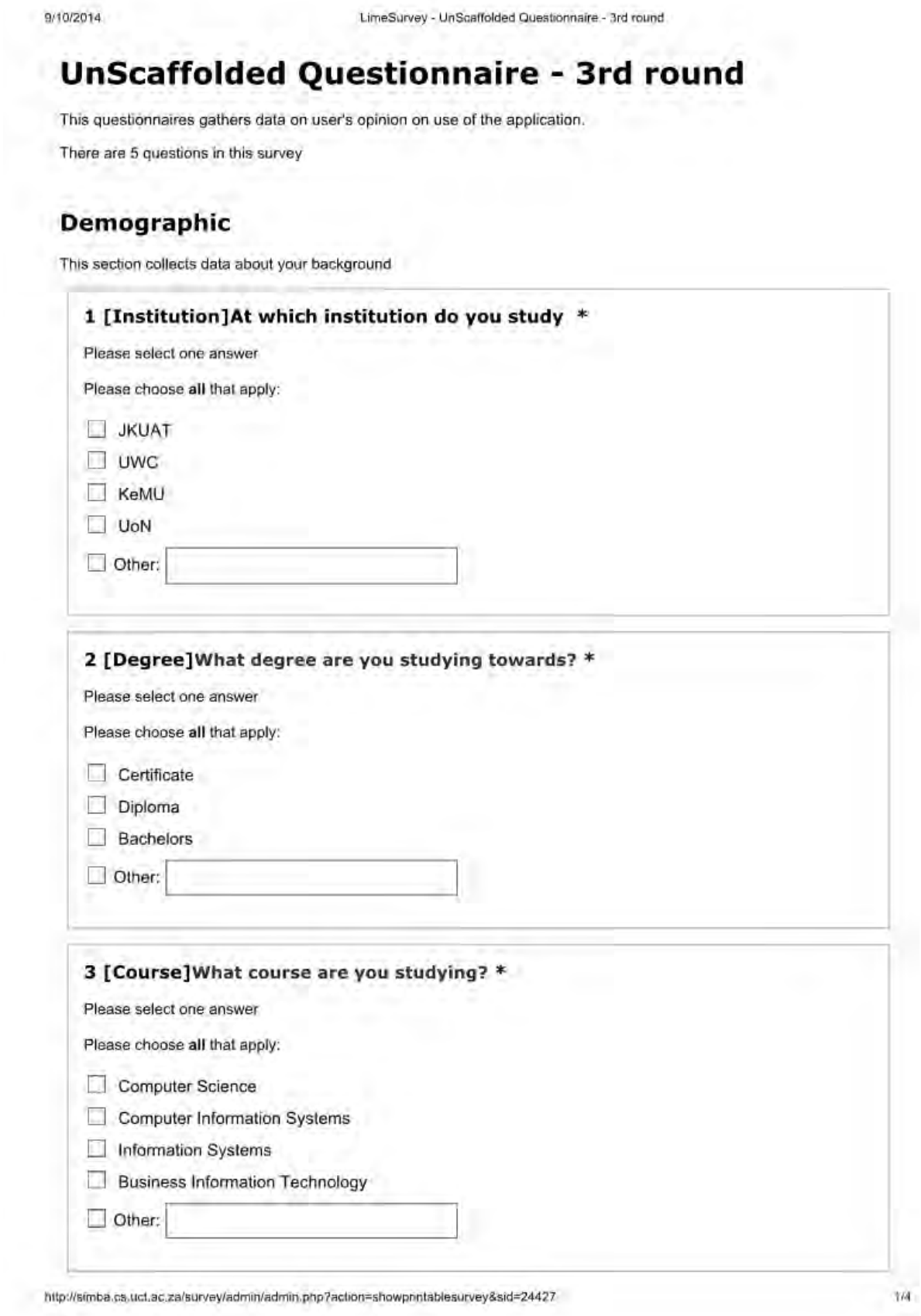
Please write your answer here:

6 [Recommend]Would you recommend the use of the application to a friend? *

Please choose **only one** of the following:

- ○ Yes
- ○ No

Appendix E3: Experiment 3 questionnaire for control group



9/10/2014 LimeSurvey - UnScaffolded Questionnaire - 3rd round

UnScaffolded Questionnaire - 3rd round

This questionnaires gathers data on user's opinion on use of the application.

There are 5 questions in this survey

Demographic

This section collects data about your background

1 [Institution]At which institution do you study *

Please select one answer

Please choose **all** that apply:

- ☐ JKUAT
- ☐ UWC
- ☐ KeMU
- ☐ UoN
- ☐ Other:

2 [Degree]What degree are you studying towards? *

Please select one answer

Please choose **all** that apply:

- ☐ Certificate
- ☐ Diploma
- ☐ Bachelors
- ☐ Other:

3 [Course]What course are you studying? *

Please select one answer

Please choose **all** that apply:

- ☐ Computer Science
- ☐ Computer Information Systems
- ☐ Information Systems
- ☐ Business Information Technology
- ☐ Other:

http://simba.cs.uct.ac.za/survey/admin/admin.php?action=showprintablesurvey&sid=24427 1/4

Use of Mobile Phones to Construct Programs

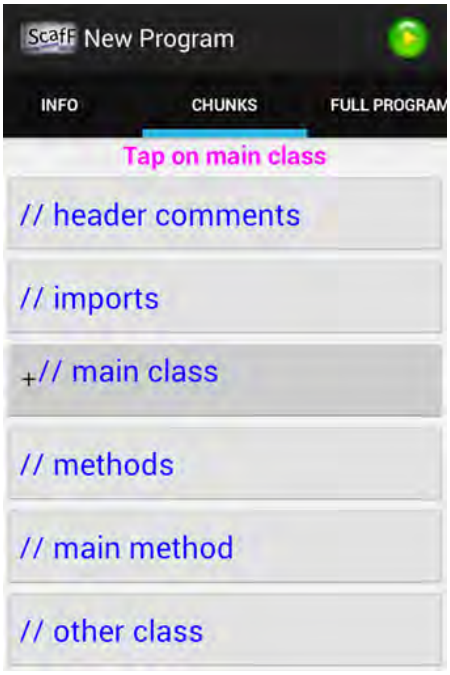
Experience with constructing programs on a mobile programming environment.

5 [Experience]What is your perception and experience on constructing programs on a mobile phone? *

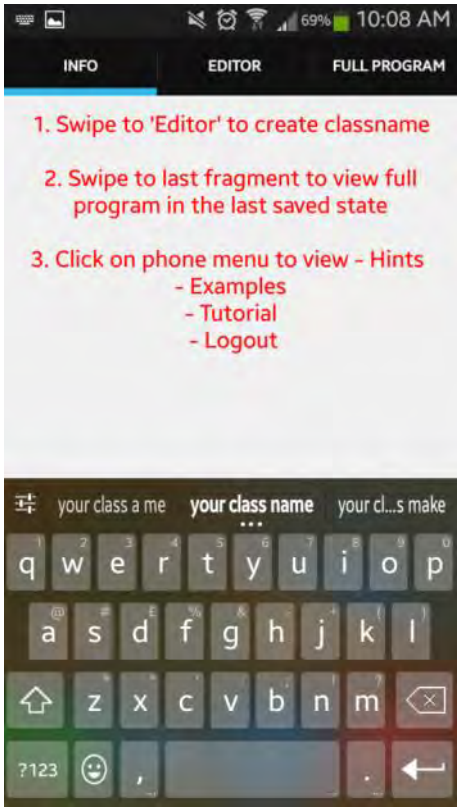
Please write your answer here:

Appendix F: Screenshots of the second prototype with modifications

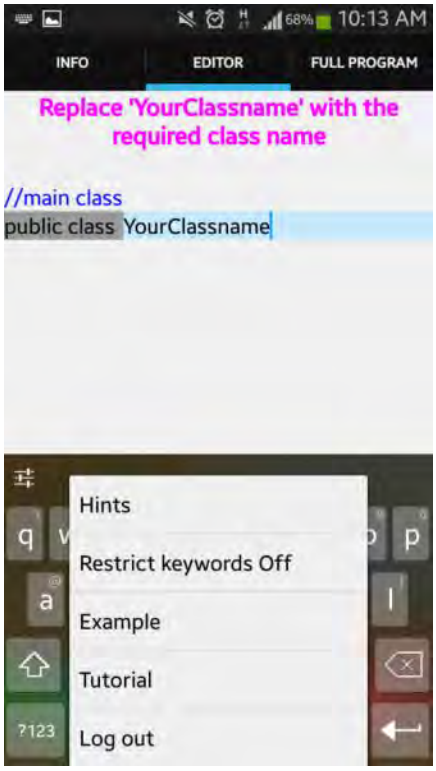
Appendix F1: Screenshot showing use of tabs in the main interface, a green run button at the top of the screen, and addition of 'other class' chunk

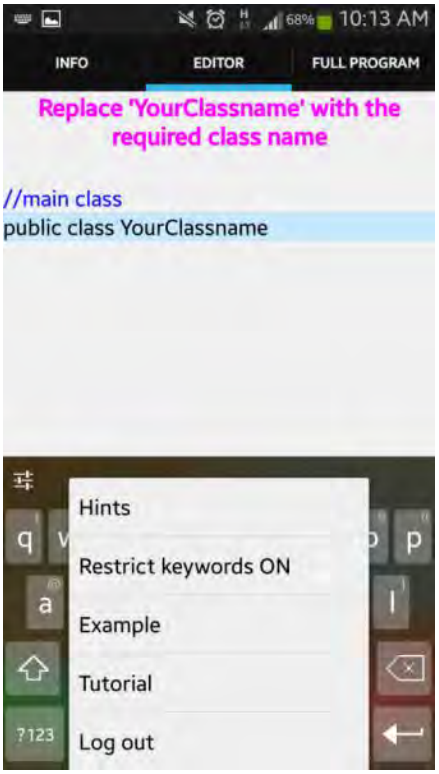


Appendix F2: Screenshot showing use of tabs in the editor

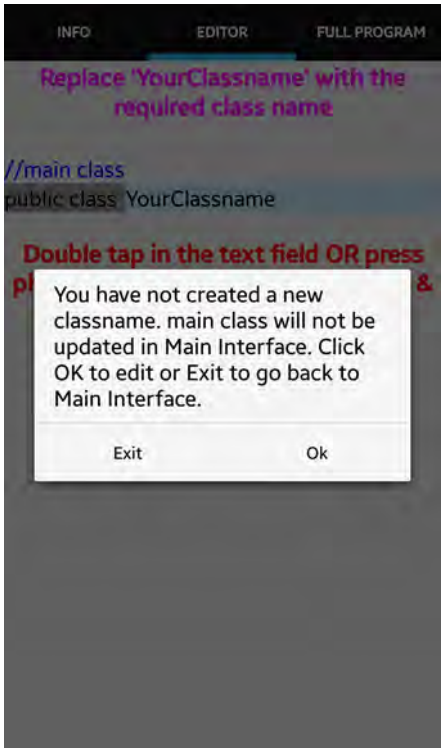


Appendix F3: Screenshot showing 'public class' keyword in main class disabled, showing menu options that can be selected to enable (left figure) or disable it (right figure)

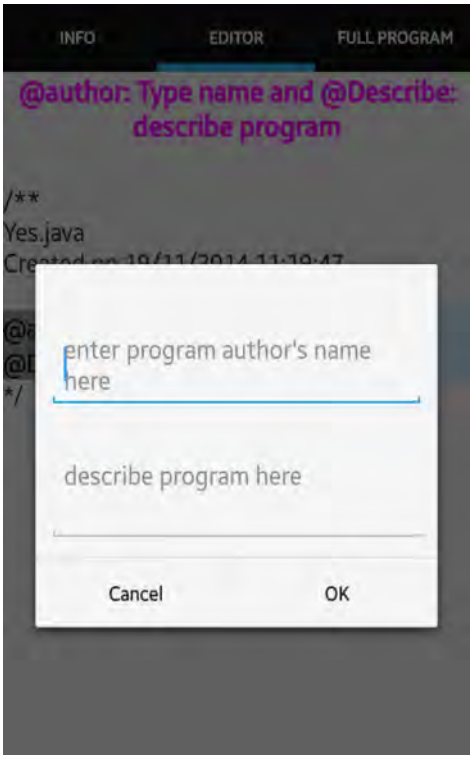


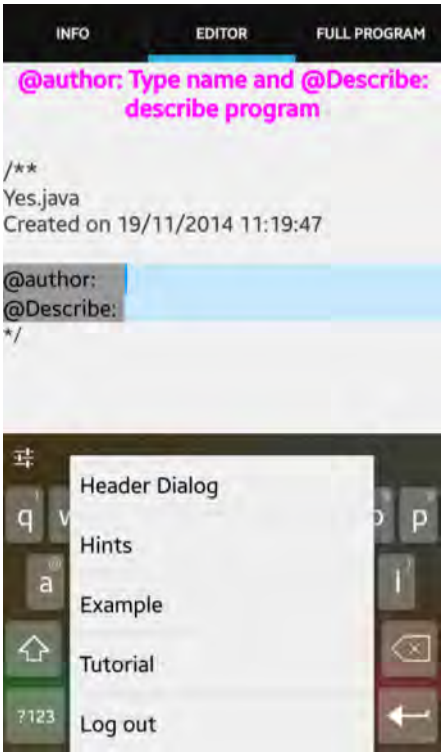


Appendix F4: Screenshot showing instructions in the main class indicating that a user can proceed without creating the main class

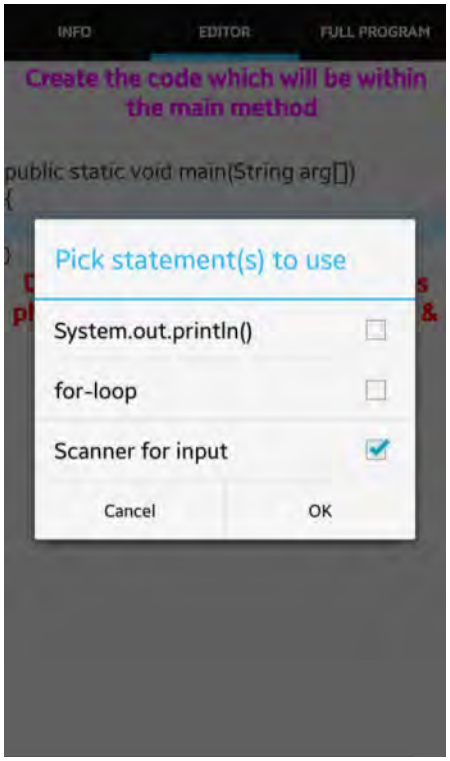


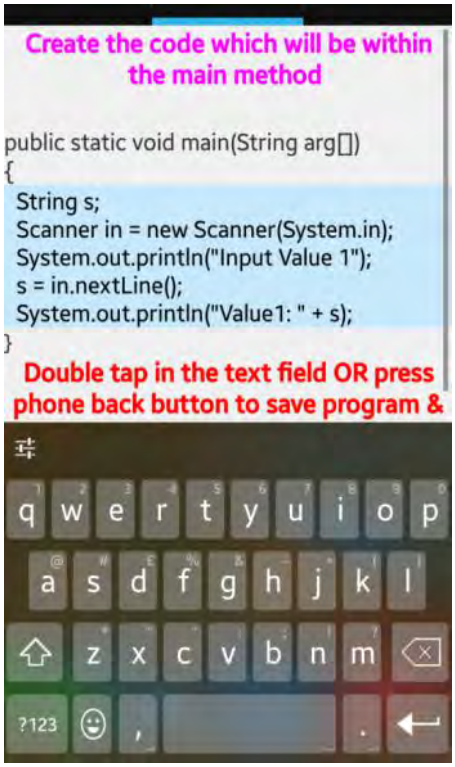
Appendix F4: Screenshot showing a header dialog (left figure) that can be enabled using a menu option (right figure)



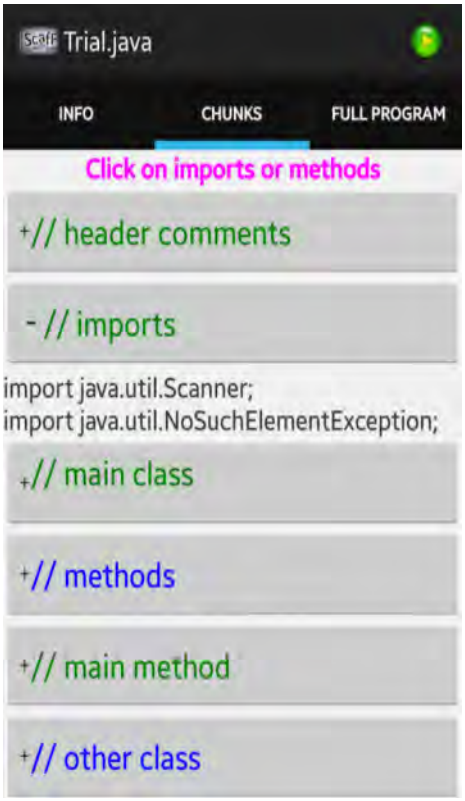


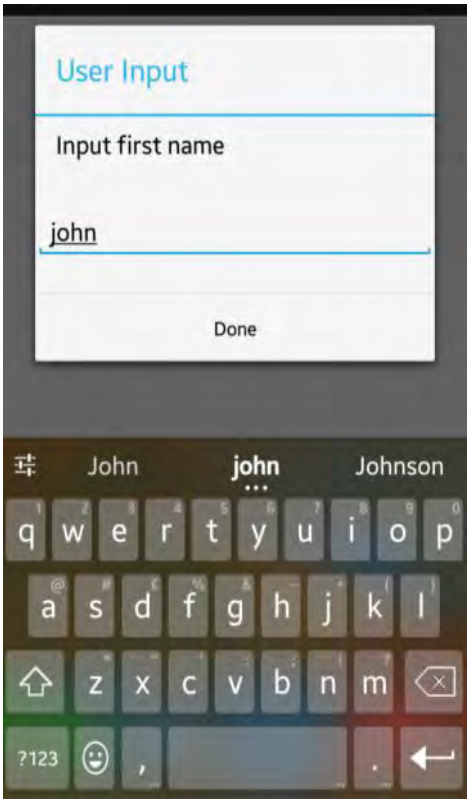
Appendix F5: Screenshot showing the Scanner class option (left figure) and the corresponding default text (right figure) that is to be edited and reused





Appendix F6: Screenshot showing the import statements that are automatically inserted in the imports chunk (left figure) and the resulting dialog box for user input when the program is compiled (right figure)





Appendix G: Raw Data for Number of Tasks

Appendix G1: Number of tasks attempted and completed per user for KeMU, Experiment 2

KEMU - Experiment 2						
experimental (number of tasks)				control (number of tasks)		
user	attempted (experimental)	completed (experimental)		user	attempted (control)	completed (control)
user1	2	1		user1	2	1
user2	2	1		user2	2	0
user3	3	1		user3	2	0
user4	2	2		user4	3	1
user5	3	2		user5	2	0
user6	3	2		user6	2	1
user7	3	2		user7	3	2
TOTAL	18	11		**TOTAL**	16	5

Appendix G2: Number of tasks attempted and completed per user for UWC, Experiment 2

UWC - Experiment 2						
	experimental				CONTROL	
	attempted tasks (experimental)	completed tasks (experimental)			attempted tasks (control)	completed tasks (control)
user1	2	0		user1	1	1
user2	2	1		user2	4	0
user3	4	3		user3	2	1
user4	4	3		user4	1	1
user5	2	2		user5	1	1
user6	1	1		user6	1	0
user7	1	1		user7	1	0
user8	1	1		user8	1	1
user9	3	2		user9	1	0
user10	2	1		user10	1	0
user11	3	3		user11	2	1
user12	3	2		user12	2	1
user13	3	2		user13	2	2
user14	1	0				
TOTAL	**32**	**22**		**TOTAL**	**20**	**9**

Appendix G3: Number of tasks attempted and completed per user for JKUAT, Experiment 2

	attempted tasks (experimental)	completed tasks (experimental)			attempted tasks (control)	completed tasks (control)
user1	3	2		user1	2	1
user2	3	1		user2	2	0
user3	1	0		user3	3	2
user4	2	1		user4	2	0
user5	3	2		user5	2	2
user6	2	1		user6	3	2
user7	2	2		user7	3	2
user8	2	1		user8	3	3
user9	2	0		user9	2	1
user10	2	0		user10	2	1
user11	5	4		user11	2	2
user12	3	2		user12	3	2
user13	1	0		user13	2	2
				user14	2	1
				user15	5	2
				user16	2	0
TOTAL	**31**	**16**		**TOTAL**	**40**	**23**

Appendix G4: Number of tasks attempted and completed per user for KeMU, Experiment 3

	KeMU - Experiment 3					
	Experimental				CONTROL	
	attempted tasks (experimental)	completed tasks (exp)			attempted tasks (control)	completed tasks (control)
user1	1	1		user1	2	0
user2	3	2		user2	2	1
user3	2	1		user3	2	0
user4	3	2		user4	1	0
user5	3	1		user5	1	0
user6	1	1		user6	2	1
user7	4	3		user7	1	0
user8	1	0		user8	2	1
user9	2	1		user9	3	0
user10	3	3		user10	2	0
user11	3	1		user11	2	1
user12	3	3				
user13	3	3				
TOTAL	**32**	**22**		**TOTAL**	**20**	**4**

Appendix G5: Number of tasks attempted and completed per user for JKUAT, Experiment 3

	jKUAT- Experiment 3					
	Experimental				CONTROL	
	attempted tasks (experimental)	completed tasks(Experimental)			attempted tasks (control)	completed tasks (control)
user1	3	2		user1	2	0
user2	2	1		user2	1	0
user3	6	5		user3	2	0
user4	1	0		user4	1	0
user5	2	0		user5	2	0
user6	3	2		user6	6	2
user7	3	2		user7	1	0
user8	3	2		user8	3	1
user9	1	0		user9	1	0
user10	3	2		user10	3	2
user11	6	6		user11	2	0
user12	6	6		user12	4	3
user13	4	3		user13	3	3
user14	4	3		user14	1	0
user15	4	3		user15	1	0
user16	3	1		user16	2	0
user17	4	0		user17	2	0
user18	3	3		user18	3	3
user19	6	5		user19	3	2
user20	4	4		user20	1	0
user21	5	3		user21	3	1
user22	4	3		user22	2	0
user23	5	4		user23	2	1
user24	1	0		user24	6	3
TOTAL	**86**	**60**		**TOTAL**	**57**	**21**

Appendix H: Raw Data for Time-on-Task

Appendix H1: Time-on-task data for learners in Control and Experimental groups at UWC Experiment 2

	time-on-task per user (all tasks) - UNSORTED								
	experimental				CONTROL				
User	Task	Time on incomplete task	Time on complete task	Total time per user	Time on incomplete task	Time on complete task	Total time per user	User	task
user1	1	6.14				37.45	37.45	user1	1
user1	2	20.37		26.51	19.04			user2	1
user2	1		36.34		15.51			user2	2
user2	2	2.1		38.44	5.3			user2	3
user3	1		35.3		1.18		41.03	user2	4
user3	2		13.31			28.15		user3	1
user3	3		7.17		13.13		41.28	user3	2
user3	4	0.55		56.33		23.25	23.25	user4	2
user4	1		8.27			25.14		user5	1
user4	2		13.11		37.07		37.07	user6	1
user4	3		22.16		22.41		22.41	user7	1
user4	4	7.07		50.61		26.19	26.19	user8	1
user5	1		30.27		37.39		37.39	user9	1
user5	2		14.34	44.61	39.58		39.58	user10	1
user6	1		26.06	26.06		15.23		user11	1
user7	1		22.1	22.1	24.06		39.29	user11	2
user8	1		38.22	38.22		13.34		user12	1
user9	1		11.59		24.06		37.4	user12	2
user9	2		12			13.5		user13	1
user9	3	11.34		34.93		17.41	30.91	user13	2
user10	1		36.17						
user10	2	5.48		41.65					
user11	1		26.46						
user11	2		15.25						
user11	3		6.51	48.22					
user12	1		26.49						
user12	2		20.21						
user12	3	0.48		47.18					
user13	1		17.13						
user13	2		18.31						
user13	3	7.07		42.51					
user14	1	14.47		14.47					

time-on-task per user (completed tasks)

experimental			CONTROL		
User	**Task**	**Time on complete task (exp)**	**Time on complete task (control)**	**User**	**task**
user2	1	36.34	37.45	**user1**	1
user3	1	35.3	28.15	**user3**	1
user4	1	8.27	25.14	**user5**	1
user5	1	30.27	26.19	**user8**	1
user6	1	26.06	15.23	**user11**	1
user7	1	22.1	13.34	**user12**	1
user8	1	38.22	13.5	**user13**	1
user9	1	11.59			
user10	1	36.17			
user11	1	26.46			
user12	1	26.49			
user13	1	17.13			
user3	2	13.31	23.25	**user4**	2
user4	2	13.11	17.41	**user13**	2
user5	2	14.34			
user9	2	12			
user11	2	15.25			
user12	2	20.21			
user13	2	18.31			
user3	3	7.17			
user4	3	22.16			
user11	3	6.51			

Time on complete task (exp)		*Time on complete task (control)*	
Mean	20.76227	Mean	22.18444444
Standard Err	2.128825	Standard Er	2.685722596
Median	19.26	Median	23.25
Mode	#N/A	Mode	#N/A
Standard De	9.985076	Standard D	8.057167789
Sample Varia	99.70175	Sample Var	64.91795278
Kurtosis	-1.01959	Kurtosis	-0.08586952
Skewness	0.371466	Skewness	0.635731812
Range	31.71	Range	24.11
Minimum	6.51	Minimum	13.34
Maximum	38.22	Maximum	37.45
Sum	456.77	Sum	199.66
Count	22	Count	9
Confidence L	4.427135	Confidence	6.193287413

t-Test: Two-Sample Assuming Unequal Varianc

	Time on complete task (exp)	*Time on complete task (control)*
Mean	20.76	22.18
Variance	99.7	64.92
Observations	22	9
Hypothesized Mean Difference	0	
df	18	
t Stat	-0.41	
P(T<=t) one-tail	0.342	
t Critical one-tail	1.734	
P(T<=t) two-tail	0.683	
t Critical two-tail	2.101	

Appendix H2: Time-on-task data for learners in Control and Experimental groups at JKUAT Experiment 2

	time-on-task per user (all tasks) - UNSORTED								
	experimental				CONTROL				
User	Task	Time on incomplete task	Time on complete task	Total time per user	Time on incomplete task	Time on complete task	Total time per user	User	task
user1	1		11.54			42.55		user1	2
user1	2		18.13		12.05		54.6	user1	3
user1	3	4.13		33.8	58.34			user2	2
user2	1		21.52		22.17		80.51	user2	3
user2	2	35.26				24.4		user3	1
user2	3	2.06		58.84		10.11		user3	2
user3	1	22.36		22.36	30.43		64.94	user3	3
user4	1		5.11		34.13			user4	1
user4	2	71.08		76.19	19.47		53.6	user4	3
user5	1		7.51			18		user5	2
user5	2		45.02			15.07	33.07	user5	3
user5	3	1.4		53.93		4.2		user6	1
User6	1		20.3			32.36		user6	2
User6	4	43.27		63.57	49.49		86.05	user6	3
User7	1		6.46			15.14		user7	1
User7	2		73.27	79.73		36.39		user7	2
User8	1		10.27		43.52		95.05	user7	3
User8	2	108.11		118.38		56.5		user8	2
User9	1	24.58				23.53		user8	3
User9	3	43.03		67.61		16	96.03	user8	4
User10	1	58.53				36.13		user9	2
User10	2	20.51		79.04	31.23		67.36	user9	3
User11	1		10.53			17.08		user10	1
User11	2		15.14		20.28		37.36	user10	2
User11	3		39.05			17.25		user11	1
User11	4		19.42			36.36	53.61	user11	2
User11	5	47.38		131.52		9.45		user12	1
User12	1		32.08			13.34		user12	2
User12	2		24.03		14.53		37.32	user12	3
User12	3	15.28		71.39		22.45		user13	1
User13	1	13.02		13.02		27.5	49.95	user13	2
						27.53		user14	1
					50.31		77.84	user14	2
						5.27		user15	1
						9.41		user15	2
					8.08			user15	3
					28.01			user15	4
					6.39		57.16	user15	5
					5.11			user16	1
					91.34		96.45	user16	3

time-on-task per user (completed tasks)						
experimental			CONTROL			
User	Task	Time on complete task (exp)	Time on complete task (control)	User	task	
user4	1	5.11	17.08	user10	1	
User7	1	6.46	17.25	user11	1	
user5	1	7.51	9.45	user12	1	
User8	1	10.27	22.45	user13	1	
User11	1	10.53	27.53	user14	1	
user1	1	11.54	5.27	user15	1	
User6	1	20.3	24.4	user3	1	
user2	1	21.52	4.2	user6	1	
User12	1	32.08	15.14	user7	1	
User11	2	15.14	42.55	user1	2	
user1	2	18.13	36.36	user11	2	
User12	2	24.03	13.34	user12	2	
user5	2	45.02	27.5	user13	2	
User7	2	73.27	9.41	user15	2	
	2		10.11	user3	2	
	2		18	user5	2	
	2		32.36	user6	2	
	2		36.39	user7	2	
	2		56.5	user8	2	
	2		36.13	user9	2	
User11	3	39.05	15.07	user5	3	
	3		23.53	user8	3	
User11	4	19.42	16	user8	4	

Time on complete task (experimental)		*Time on complete task (control)*		t-Test: Two-Sample Assuming Unequal Variances		
					Time on complete task (exp)	*Time on complete task (control)*
Mean	22.461	Mean	22.4357	Mean	22.46125	22.43565
Standard Err	4.4435	Standard Er	2.71162	Variance	315.9108	169.1161
Median	18.775	Median	18	Observations	16	23
Mode	#N/A	Mode	#N/A	Hypothesized	0	
Standard De	17.774	Standard De	13.0045	df	26	
Sample Vari	315.91	Sample Vari	169.116	t Stat	0.004917	
Kurtosis	3.5528	Kurtosis	0.56563	P(T<=t) one-t	0.498057	
Skewness	1.7655	Skewness	0.84474	t Critical one	1.705618	
Range	68.16	Range	52.3	P(T<=t) two-t	0.996114	
Minimum	5.11	Minimum	4.2	t Critical two	2.055529	
Maximum	73.27	Maximum	56.5			
Sum	359.38	Sum	516.02			
Count	16	Count	23			
Confidence l	9.471	Confidence	5.62355			

Appendix H3: Time-on-task data for learners in Control and Experimental groups at KeMU Experiment 3

	KeMU, Experiment 3								
	time-on-task per user (all tasks) - With all incomplete tasks								
	experimental				CONTROL				
User	Task	Time on incomplete task (exp)	Time on complete task (exp)	Total time per user	Time on incomplete task (ctrl)	Time on complete task (exp)	Total time per user	User	task
user1	1		48.47	48.47	31.09			user1	1
user2	1	27.29			29.05		60.14	user1	2
user2	2		14.09			42.26		user2	1
user2	3		15.25	56.63	25.39		67.65	user2	2
user3	1		60.54		5.01			user3	1
user3	2	25.09		85.63	34.3		39.31	user3	2
user4	1		31.14		23.39		23.39	user4	1
user4	2		14.54		70.37		70.37	user5	1
user4	3	28.23		73.91	44.41			user6	1
user5	1	57.13				9.39	53.8	user6	2
user5	2		12.52		58.3		58.3	user7	1
user5	3	3.36		73.01		30.42		user8	1
user6	1		30.06	30.06	34.36		64.78	user8	2
user7	1		12.38		38.47			user9	1
user7	2		6.42		36.49			user9	2
user7	3		5.42		7.46		82.42	user9	3
user7	4	27.46		51.68	46.55			user10	1
user8	8	75.19		75.19	18.18		64.73	user10	2
user9	1		34.52			27.36		user11	1
user9	2	14.49		49.01	31.42		58.78	user11	2
user10	1		34.15						
user10	2		8.08						
user10	3		17.23	59.46					
user11	1	36.22							
user11	2		15.34						
user11	3	12.04		63.6					
user12	1		27.09						
user12	2		8.19						
user12	3		8.56	43.84					
user13	1		36.56						
user13	2		15.49						
user13	3		3.42	55.47					

Appendix H4: Time-on-task data for learners in Control and Experimental groups at JKUAT Experiment 3

		JKUAT Experiment 3							
	time-on-task all tasks - With all incomplete tasks								
	experimental				CONTROL				
User	Task	Time on incomplete task (exp)	Time on complete task (exp)	Total time per user	Time on incomplete task (ctrl)	Time on complete task (exp)	Total time per user	User	task
user1	1		22.38		34.3			user1	1
user1	3		28.2		24.49		58.79	user1	2
user1	4	43.4		93.98	67.18		67.18	user2	1
user2	1		34.41		93.45			user3	1
user2	2	3.24		37.65	1.42		94.87	user3	2
user3	1		34.21		30		30	user4	1
user3	2		9.04		34.38			user5	1
user3	3		9.25		30.5		64.88	user5	2
user3	4		18.36			11.57		user6	1
user3	5		4.08			8.59		user6	2
user3	6	16.28		91.22	11.32			user6	3
user4	1	63.37		63.37	8.25			user6	4
user5	1	69.55			10.23			user6	5
user5	3	8.11		77.66	29.13		79.09	user6	6
user6	1		13.29		44.29		44.29	user7	1
user6	2		5.25		51			user8	1
user6	3	24.19		42.73		22.52		user8	2
user7	1		41.25		23.18		96.7	user8	3
user7	2		11.26		59.04		59.04	user9	1
user7	3	12.02		64.53		31.43		user10	1
user8	1		13.4			25.01		user10	2
user8	2		6.32		17.52		73.96	user10	3
user8	3	14.09		33.81	42.34			user11	1
user9	1	45.4		45.4	34.13		76.47	user11	2
user10	1		36.12			19.51		user12	1
user10	2		1.42			23.1		user12	2
user10	3	20.54		58.08		20.12		user12	3
user11	1		16.37		37.23		99.96	user12	4
user11	2		11.05			25.41		user13	1
user11	3		2.03			11.37		user13	2
user11	4		17			23.15	59.93	user13	3
user11	5		3.58		39.37		39.37	user14	1
user11	6		28.52	78.55	62.44		62.44	user15	1
user12	1		5.45		4.49			user16	1
user12	2		2.23		59		63.49	user16	4
user12	3		6.16		26.25			user17	1
user12	4		9.41		36.47		62.72	user17	2
user12	5		2.56			30		user18	1
user12	6		7.02	32.83		14.37		user18	2

user13	1		29.35			32.52	76.89	**user18**	3
user13	2		13.17			18.8		**user19**	1
user13	3		4.39			12.7		**user19**	2
user13	4	11.01		57.92	31.2		62.7	**user19**	3
user14	1		29.16		45.3		45.3	**user20**	1
user14	2		5.05			10.62		**user21**	1
user14	3	17.04			20.11			**user21**	2
user14	4		13.04	64.29	6.23		36.96	**user21**	3
user15	1		22.15		28.4			**user22**	1
user15	2		14.09		15.7		44.1	**user22**	4
user15	3		30.15			15.6		**user23**	1
user15	4	15.46		81.85	29.3			**user23**	2
user16	1	32.49				13.1		**user24**	1
user16	3	46				7.1		**user24**	2
user16	2		22.5	100.99		17.2		**user24**	3
user17	1	36.55			14			**user24**	4
user17	2	15.22			12.9			**user24**	5
user17	3	16.19			15.8		80.1	**user24**	6
user17	4	23.42		91.38					
user18	1		11.13						
user18	2		3.12						
user18	3		27.42	41.67					
user19	1		40.38						
user19	2		5.32						
user19	3		3.22						
user19	4		10.58						
user19	5	11.21							
user19	6		10.08	80.79					
user20	1		17.13						
user20	2		13.3						
user20	3		7.16						
user20	4		35.16	72.75					
user21	1		22.4						
user21	2		8.35						
user21	3		38.38						
user21	4	16.42							
user21	5	8		93.55					
user22	1		13.52						
user22	2		15.49						
user22	3		4.03						
user22	4	7.44		40.48					
user23	1		21.49						
user23	2		13.34						
user23	3		21.49						
user23	4		24.5						
user23	5	6.05		86.87					
user24	3	11.32		11.32					

Appendix I: Raw Data for Verbatim User Feedback

Appendix I1: Survey responses at UWC, Experiment phase 2

id	Completed	Indicate the features of the application that most supported your construction of programs on the mobile device. Give as much detail as you can. Give as much detail as you can.	In your opinion, is there anything missing from the application that would support construction of programs on a mobile device?	Would you recommend the use of the application to a friend?
2	2014-06-20 15:38:16	I really enjoyed the program,because it has made my life easy.It is structrure,there's a tab for methods,a tab for main, a tab for classes,a tab for documentation.And it allows you to go through the m by order,thus making me realise that documenting your code is very important.An when it compiles it is more like a reall "computer desk top",it highlights where you made a mistake and allows you to go back and fix errors.Without any lies,I love it.	Nothing missing other than the fact that it is a reall computer but mobile,and yehhhhhhhhh!!!!!!I can code whereever I am,in the bus,taxi,home,etc.....	Yes
3	2014-06-20 15:39:18	It helped in that most system(e.g. for loop, sout) were already created. It is well constructed in that, it clearly states on where to start fist.	Well i Think not	Yes
4	2014-06-20 15:34:43	none	IDE	No
5	2014-06-20 15:37:38	The instructions on which parts of the interface to begin with.	Yes, the fact that it was a touch screen phone was a disadvantage. I think that familiarity/preference for touch screens may be a confounding variable.	No
6	2014-06-20 15:37:56	its main class is well designed	jep when using other smart phone it will be difficult to get some icons for example other Nokia	Yes
7	2014-06-20 15:38:31	The application divides the program or code into sections then one can the track and write the code properly by following the sections	No	Yes
8	2014-06-20 15:39:05	The features of this application which were helpful was the fact that the statements were there already..	Yes,It is very difficult to navigate..It must be made easy so that people can enjoy it	Yes
9	2014-06-20 15:51:20	java netbeans application	Its interface must be improved so that it could be easy to access it. the application will be better if run on a button touched phones	Yes
10	2014-06-20 15:48:57	the menu that makes you write imports,class name main method etc.	it would be nice if it could save automatically	Yes
11	2014-06-20 15:46:21	creating the main method automatically, and assisting with the writing of the comments, and filling in the opening and closing,it also is simple to save the document, since it seems like it does it automatically	i didnt see the part that creates a "constructor as simple as creating the main method, and when you trying to edit your program it should be easier to browse and move around the document, the double clicks makes one loose patience....at this i can only recommend it to a friend if they are writing a very short program	Yes
12	2014-06-20 15:45:51	The separate segments of program	Spelling checks,different colors	Yes
13	2014-06-20 15:38:15	methods and import	java API docs	Yes
14	2014-06-20 15:46:08	Preset statement helped in typing. The sections are well laid out. The hints helped in where to type. The error handling is accurate in pinpointing errors. Very good program,would love to see it on a tablet.	Would be great if there were a few imports(packages) that are commonly used that are in the preset menu. I think there is a memory handling error on the device cause as I was coding the 4th program,all my code was erased after compiling.	Yes

Appendix I2: Survey responses at JKUAT, Experiment phase 2

USER ID	Completed	Indicate the features of the application that most supported your construction of programs on the mobile device. Give as much detail as you can. Give as much detail as you can.	In your opinion, is there anything missing from the application that would support construction of programs on a mobile device?	Would you recommend the use of the application to a friend?
1				
2	2014-07-23 16:22:56	how the codes are divide into chunks making the application easier to use	no	Yes
3	2014-07-23 16:29:41	application programming interface is excellent. its documentation is sufficient	it should also consider input stream reader and buffered reader for convinience	Yes
4				
5				
6	2014-07-23 16:26:18	It has the Application Programming Interface. It is platform independent It has the Android SDk manager	No:Everything is available in the application	Yes
7	2014-07-23 16:27:28	Auto complete where at some point t suggested words for easier typing.	No	Yes
8	2014-07-23 16:25:56	Very quick. Quick error detectioin	It is awesome	Yes
9	2014-07-23 16:28:40	There were available lists that made it easier to write code in the program.	from a personal point of view,the mobile platform is at its best.	Yes
10	2014-07-23 16:25:48	easy durable accurate	nothing	Yes
11	2014-07-23 16:31:21	Main method.	Nothing is missing.	Yes
12				
13	2014-07-23 16:30:55	statement dialog general organization i.e imports ,methods other classes etc	printf function	Yes
14	2014-07-23 16:41:54	The inbuilt java syntax really helped because for the beginner one doesnt have to cramp the syntax . The ability if the app to be compiled..	The lack of undo option.	Yes
15				
16	2014-07-23 16:34:25	scaffold: it is user friendly and has a very beautifull GUI interface. It is very easy to use and can be used anywhere since it is portable	to my opinion i do not think so	Yes
17	2014-07-23 16:34:51	scalffold application	No	Yes
18	2014-07-23 16:40:45	Well the organization is simple and easy to learn plus using of main methods easy and is already defined in the system.	Well the program would do well to provide easier ways to save changes. Plus the application needs to help in GUI programming also	Yes

Appendix I3: Survey responses at KeMU and JKUAT, Experiment phase 3

id	Completed	Indicate why you could not attempt all the questions?	Indicate the features of the application that most supported your construction of programs on the mobile phone. Give as much detail as you can.
1			
2	2014-10-21 09:52:48		
3	2014-10-21 09:52:07		
4			
5			
6	2014-10-21 10:07:18	very little java knowledge	the chunks made it easier to construct the program
7	2014-10-21 10:16:04		its well organised
8	2014-10-21 10:14:43		information icon on how to start the program. run icon to execute the program. full program view. tutorials on how to start a program
9	2014-10-21 19:45:15	time could not allow	Statements dialog Examples
10	2014-10-21 19:43:49	i did attempt a few	Its interface is understandable.
11			
12		they were challengeing	
13	2014-10-21 19:58:01		All the inbuilt features E.g Scanner facility,tutorials and system.out.print
14	2014-10-21 19:57:54		inbuilt features like-scanner,tutorial,systen.out.println,for-loops,easy to save program and retrieve it.direct where here is error after excution
15	2014-10-21 20:01:49		pre-defined methods, statements and functions
16			
17			
18			
19	2014-10-22 11:15:15	I came to class late so had limited time to attempt a few.	Availability of construction codes with instructions.
20			
21			
22			
23			
24	2014-10-29 17:03:38		The statements dialog really makes work easier removing the need to import some packages
25			
26	2014-10-29 17:13:51	The time was limited	the instructions were clear enough and the programs were easy to write
27			
28			
29			
30	2014-10-29 17:44:06		The ability to import packages and classes and run the programs. Its ability to detect and in most cases correct errors. Concise syntaxes and easily comprehendable instructions. Simple navigation.
31	2014-10-29 17:46:48		The application is ready to go
32	2014-10-29 18:05:44	I was not good in classes and methods so in such questions i had to leave blank spaces	The compilation and Execution of the program looked good and also the graphical user interface of the program is user friendly and well defined i.e sections of main class,imports, methods etc.....Atleast those who have little knowledge about programming can use this application.
33			
34	2014-10-29 18:43:19		The app is greate

Appendix J: ERROR ANALYSIS

Appendix J1: Raw data showing error analysis of UWC data from the experimental group in the second Experiment

User	SCAFFOLDED ERROR PROMPTS – Number of times	NUMBER OF RUN-TIME ERRORS	User enabled Scaffolding techniques used during run-time error correction
User1	Program 1 - none Program 2 - ***Main Class Error Line in wrong publi class classname format***	None none	none
User 2	Program 1 – none Program 2 - none	1 none	None
User 3	Program 1 - ***Main class Error classname does not begin with an upper case***	1	Full program main interface Instructions main interface
	Program 2 - none	6	Full program main interface
	Program 3- Main class error: line in wrong format Program 4 - none	3	none
User 4	Program 1 – none Program 2 – none Program 3 & 4 - none	None None Program 3 & 4 – 3 each	None None None
User 5	Program 1 – none Program 2 - none	Program 1 – none Program 2 - none	None none
User 6	Program 1 – ***Classname should not contain special character***	none	none
User 7	Program 1 – ***Main class Error classname does not begin with an upper case***	1	Instructions main interface
User 8	Program 1 – none	1	none

User	SCAFFOLDED ERROR PROMPTS – Number of times	NUMBER OF RUN-TIME ERRORS	User enabled Scaffolding techniques used during run-time error correction
User 9	Program 1 – none Program 2 - none	1 None	none
User 10	Program 1- none Program 2 - none	none	
User 11	Program 1- none	none	none
	Program 2 - none	2	none
	Program 3 - none	none	none
User 12	Program 1- none	none	none
	Program 2 - none	1	none
	Program 3 - none		
User 13	Program 1- none	none	none
	Program 2 - none	none	none
	Program 3 – public,void, return, static statements in main method (twice)		
User 14	Program 1- none	1	none

Appendix J2: Raw data showing error analysis of JKUAT data from the experimental group in the second Experiment

User	SCAFFOLDED ERROR PROMPTS – Number of times	NUMBER OF RUN-TIME ERRORS	User enabled Scaffolding techniques used during run-time error correction
User1	Program 1- Main Class Error: Classname does not begin with uppercase Program 2 - Main method : Main Method Error: A for loop syntax doesnt have two commas within the declaration Program 3 - none	None None 1 – Cannot find variable sum	none
User 2	Program 1 - Main class attempt to add extra line Program 2 – none	None 4 -	none

User	SCAFFOLDED ERROR PROMPTS – Number of times	NUMBER OF RUN-TIME ERRORS	User enabled Scaffolding techniques used during run-time error correction
	Program 3 – Classname does not contain public and class keywords	*RESULT: cannot find symbol average= sum/20;* package system does not exist system.out.println("Average" + average); (Three times)	
User 3	Program 1 - Classname does not contain public and class keywords	none	none
User 4	Program 1 - Main class attempt to add extra line	8- RESULT: Main.java:17: error: <identifier> expected {int sum, double average; ^ Main.java:17: error: not a statement {int sum, double average; ^ Main.java:19: error: not a statement for(I=1,I<= 20,I++); ^ Main.java:19: error: ';' expected for(I=1,I<= 20,I++); ^ Main.java:20: error: illegal start of expression {sum= sum+I ^ Main.java:20: error: ';' expected {sum= sum+I ^ Main.java:29: error: class, interface, or enum expected public static void main(String arg[]) ^ Main.java:38: error: class, interface, or enum expected } ^ 8 errors	none
User 5	Program 1- none Program 2 - Main class attempt to add extra line Program 3 - none	None 7 - ';' expected int sum=0 ^ 1	none
User 6	Program 1 - attempt to add extra line at main class (three times) Program 2 - Classname does not contain public and class keywords (twice) - attempt to add extra line at main class (once)	None none	

User	SCAFFOLDED ERROR PROMPTS – Number of times	NUMBER OF RUN-TIME ERRORS	User enabled Scaffolding techniques used during run-time error correction
User 7	Program 1 - Main Class Error: Classname does not begin with uppercase Program 2 - Classname does not contain public and class keywords - attempt to add extra line at main class - Classname does not begin with uppercase	None 18 run time errors	
User 8	Program 1 – none Program 2 - none	None 18 run time errors	
User 9	Program 1 - attempt to add extra line at main class Program 2 - none	None 2 - RESULT: illegal line end in character literal System.out.println("input age")' - cannot find symbol Scanner in=Scanner(System.in);	
User 10	Program 1 – (7) *Main Class Error: Classname does not begin with uppercase* *Main class attempt to add extra line* *Main class attempt to add extra line* *Main Class Line contains special character* Program 2 – 1 *Main Class Error: Classname does not begin with uppercase*	3 - *RESULT: Main.java:19: error: ')' expected system .out (scaffolding at jkuat) ^ Main.java:19: error: not a statement system .out (scaffolding at jkuat) ^ Main.java:19: error: ';' expected system .out (scaffolding at jkuat) ^ 3 errors* 3 – *RESULT: Main.java:20: error: not a statement sum+i; ^ Main.java:22: error: ';' expected average=sum/ 20 ^ Main.java:23: error: ';' expected system .out.println("sum is "+sum) ^ 3 errors :23/07/2014 19:09:00:690* 4	
User 11	Program 1 - Main class attempt to add extra line (twice) Program 2 – none Program 3 – none	None None 5 errors	none

User	SCAFFOLDED ERROR PROMPTS – Number of times	NUMBER OF RUN-TIME ERRORS	User enabled Scaffolding techniques used during run-time error correction
	Program 4 – none Program 5 - none	Program 4 – none Program 5 – 2 - RESULT: Main.java:15: error: '.class' expected Init(String, int); ^ 1 error :	
User 12	Program 1 - Main class attempt to add extra line (4) Program2 - none Program 3 – none Program 4 - none	None none 1 4 errors *RESULT: Main.java:19: error: ')' expected Scanner= Scanner(System in); ^ Main.java:19: error: illegal start of expression Scanner= Scanner(System in); ^ 2 errors*	
User 13	Program 1 - none	None	

Appendix J3: Raw data showing error analysis of JKUAT data from the experimental group in the third Experiment

User	SCAFFOLDED ERROR PROMPTS – Number of times	NUMBER OF RUN-TIME ERRORS	User enabled Scaffolding techniques used during run-time error correction
User1	PROGRAM 1 - COMPLETED Main Class Error: Classname contains .java - 2 Main Class: Error: Line contains special character - 2 Main class attempt to add extra line - 2	5 - '.class' expected System.Out.print n(double(x));	Full program
	PROGRAM 2 - COMPLETED Main Class Error: Classname does not contain public and class keywords: Main class attempt to add extra line:	3 - cannot find symbol Int a;	None

User	SCAFFOLDED ERROR PROMPTS – Number of times	NUMBER OF RUN-TIME ERRORS	User enabled Scaffolding techniques used during run-time error correction
	PROGRAM 3 – Incomplete Main Class Error: If there are only two words in the declaration:	2 - expected methStud()	None
User 2	PROGRAM 1 – COMPLETE Main class attempt to add extra line: - 2	1 - error: ')' expected	None
	PROGRAM 2 – Incomplete	None	None
User 3	PROGRAM 1 – COMPLETE *Main class attempt to add extra line: - 2*	2 - error: '.class' expected error: - variable x is already defined in method main(String[]) double x	Full program
	PROGRAM 2 – COMPLETE	1 error: reached end of file while parsing }	Full program
	PROGRAM 3 – COMPLETE	None	None
	PROGRAM 4 – COMPLETE	None	None
	PROGRAM 5 – COMPLETE	None	None
	PROGRAM 6 – Incomplete	1- cannot find symbol average()	None
User 4	PROGRAM 1 – InCOMPLETE Main Class Error: If there are only two words in the declaration:	None	None
User 5	PROGRAM 1 – InCOMPLETE Main Class Error: Classname does not begin with uppercase:	None	None
	PROGRAM 2 – InCOMPLETE	3 cannot find symbol System.Out.Println cannot find symbol double p=x*2;	Statement dialog Examples Full Program
User 6	PROGRAM 1 – COMPLETE	None	None
	PROGRAM 2 – COMPLETE	None	None
	PROGRAM 3 – INCOMPLETE	None	None
User 7	PROGRAM 1 – COMPLETE	None	None
	PROGRAM 2 – COMPLETE ***Main Class Error: Classname does not begin with uppercase:***	None	None

User	SCAFFOLDED ERROR PROMPTS – Number of times	NUMBER OF RUN-TIME ERRORS	User enabled Scaffolding techniques used during run-time error correction
	Main class attempt to add extra line:		
	PROGRAM 3 – INCOMPLETE	1 - package system does not exist system.out.println("enter data");	None
User 8	PROGRAM 1 – COMPLETE ***Main class attempt to add extra line: - 2***	1 - system does not exist system.out.println(x*2);	Full program
	PROGRAM 1 – COMPLETE	None	None
	PROGRAM 3 – INCOMPLETE	1- cannot find symbol Scaner input= new Scanner(System.in)	None
User 9	PROGRAM 1 – InCOMPLETE	3 - error: package system does not exist system.out.println(x);	Full program Statement dialog
User 10	PROGRAM 1 – COMPLETE *Main class attempt to add extra line:*	1- ';' expected System.out.println("a good program ") 2- - illegal character: \215 x=2?10;	Editor instructions
	PROGRAM 2 – COMPLETE	None	None
	PROGRAM 3 – INCOMPLETE *Main class attempt to add extra line:*	1 - cannot find symbol Scanner a=new Scanner(System.in);	Full program Main interface instructions
User 11	PROGRAM 1 – COMPLETE	1- ')' expected System.out.println(" " + 2x); 2- error: not a statement 2*x 3- illegal start of expression x=*2;	Full program – 3 times Main interface instructions
	PROGRAM 2 – COMPLETE ***Main Class Error: Classname does not contain public and class keywords:***	None - full program - 2, statement dialog, examples - twice	None
	PROGRAM 3 – COMPLETE	None - statement dialog, full program	None
	PROGRAM 4 – COMPLETE – advanced interface	1- variable in is already defined in method main	None
	PROGRAM 5 – COMPLETE – advanced interface	None - full program once	None

User	SCAFFOLDED ERROR PROMPTS – Number of times	NUMBER OF RUN-TIME ERRORS	User enabled Scaffolding techniques used during run-time error correction
	PROGRAM 6 – COMPLETE – advanced interface ***Main class attempt to add extra line:***	None - Scaffolding techniques used - full program - 2, statement dialog, instructions, tutorial, keywords enabled	None
User 12	PROGRAM 1 – COMPLETE	1 - error: not a statement x * 2; 3. illegal start of expression x =*2;	None
	PROGRAM 2 – COMPLETE	None – example suggestion accepted, full program	None
	PROGRAM 3 – COMPLETE	1- error: '(' or '[' expected Scanner s = new Scanner; 2- cannot find symbol value = s.nextline;	None
	PROGRAM 4 – COMPLETE	None - statement dialog	None
	PROGRAM 5 – COMPLETE – advanced interface	None – full program	None
	PROGRAM 6 – COMPLETE – advanced interface	None – statement dialog	None
User 13	PROGRAM 1 – COMPLETE *Main class attempt to add extra line: - 2*	1- error: ';' expected int x 2- error: ';' expected double x=10	Full program - 3 Instruction
	PROGRAM 2 – COMPLETE	None – Examples	None
	PROGRAM 3 – COMPLETE *Main Class Error: If there are only two words in the declaration:* *Main Class Error: Classname does not begin with uppercase:* *Main Class: Error: Line in wrong publi class classname format:*	None – Use of examples	None
	PROGRAM 4 – InCOMPLETE – Advanced	None	None
User 14	PROGRAM 1 – COMPLETE ***Main class attempt to add extra line: - 4***	1- no suitable method found for println(String,int) 2- cannot find symbol sum=x*2;	Full program instructions

User	SCAFFOLDED ERROR PROMPTS – Number of times	NUMBER OF RUN-TIME ERRORS	User enabled Scaffolding techniques used during run-time error correction
	Main Class Error: Classname does not begin with uppercase:		
	PROGRAM 2 – COMPLETE	None	None
	PROGRAM 3 – InCOMPLETE	1 - unclosed string literal System.out.println("enter your nam	Scanner example rejected Full program
	PROGRAM 4 – COMPLETE	None	None
User 15	PROGRAM 1 – COMPLETE	None 1- <identifier> expected int x=10, double;	Statement dialog
	PROGRAM 2 – COMPLETE	1- error: ')' expected System.out.println("i": +i)	None
	PROGRAM 3 – COMPLETE	1- ')' expected System.out.println("Your input is": + words);	Hints, Statement dialog, instructions, examples, Full program
	PROGRAM 4 – inCOMPLETE	1 - error: ';' expected y=in.next line();	Examples
User 16	PROGRAM 1 – incomplete *Main class attempt to add extra line:*	1- error: ';' expected system.out.println(x)	None
	PROGRAM 2 – incomplete *Main class attempt to add extra line:*	1- unclosed string literal Sytem.out.println("Your input is + input)	None
	PROGRAM 3 – COMPLETE	Noen – full program	None
User 17	PROGRAM 1 – inCOMPLETE	1- error: ';' expected int x=10	Instructions Full program
	PROGRAM 2 – inCOMPLETE	Not completed chunks	none
	PROGRAM 3 – incomplete ***Main Class Error: Classname does not begin with uppercase:***	Not run	none
	PROGRAM 4 – inCOMPLETE	None – statmetn dialog, examples, full program	None

User	SCAFFOLDED ERROR PROMPTS – Number of times	NUMBER OF RUN-TIME ERRORS	User enabled Scaffolding techniques used during run-time error correction
	Main Method Error : public,void, return, static statements in main method:	Not run	
User 18	PROGRAM 1 – COMPLETE ***Main Class: Error: Line contains special character:*** ***Main class attempt to add extra line: - 2***	None	None
	PROGRAM 2 – COMPLETE	None – full program	None
	PROGRAM 3 – Complete	5 Scanner s= new Scanner(System.in);	Full program
User 19	PROGRAM 1 – COMPLETE *Main class attempt to add extra line:*	3. error: '.class' expected System.out.println(double (x));	Instructions
	PROGRAM 2 – COMPLETE	1- error: ';' expected System.out.println (", ")	none
	PROGRAM 3 – COMPLETE	None - statement dialog	None
	PROGRAM 4 – COMPLETE – advanced interface	None - Hints, statement dialog	None
	PROGRAM 5 – COMPLETE – advanced interface	None - Statement dialog, full program	None
	PROGRAM 5 – inCOMPLETE – advanced interface	Did not contain any output	None
User 20	PROGRAM 2 – COMPLETE *Main class attempt to add extra line:*	1 - : error: variable natural_numbers might not have been initialized	Full program
	PROGRAM 3 – COMPLETE	None – statement dialog	
	PROGRAM 4 – inCOMPLETE	1- unclosed string literal System.out.println	Full program
User 21	PROGRAM 1 – COMPLETE *Main class attempt to add extra line:*	2- ';' expected System.out.println("x?x")	Full program
	PROGRAM 2 – COMPLETE	1- cannot find symbol System.out.println (I);	none
	PROGRAM 3 – COMPLETE	None – statement dialog, full program	none

User	SCAFFOLDED ERROR PROMPTS – Number of times	NUMBER OF RUN-TIME ERRORS	User enabled Scaffolding techniques used during run-time error correction
	PROGRAM 4 – inCOMPLETE – advanced interface	Not run	none
	PROGRAM 5 – inCOMPLETE – advanced interface *Main class attempt to add extra line:*	1- error: ';' expected Output a=new Output ()	none
User 22	PROGRAM 1 – COMPLETE	None – full program	None
	PROGRAM 2 – COMPLETE Main Class Error: If there are only two words in the declaration:	None – full program	None
	PROGRAM 3 – COMPLETE	None- statement dialog	None
	PROGRAM 4 – COMPLETE – advanced interface *Main Class: Error: Line contains special character:* *Main Class: Error: Line in wrong publi class classname format:*	Program is nor run	
User 23	PROGRAM 1 – COMPLETE	1- error: not a statement int x=10;!	None
	PROGRAM 2 – COMPLETE	2. not a statement for(int 0;i<=9;i++)	none
	PROGRAM 3 – COMPLETE Main Class Error: Classname does not begin with uppercase:	2- error: cannot find symbol string a;	Statement dialog
	PROGRAM 4 – COMPLETE – advanced interface *Main class attempt to add extra line:*	1- error: ';' expected MethSt a=new MethSt () 2- cannot find symbol string x;	Full program
	PROGRAM 5 – inCOMPLETE	Not run	None
User 24	PROGRAM 1 – inCOMPLETE	1 error: variable in is already defined in method main	none