A TANGIBLE PROGRAMMING ENVIRONMENT MODEL

INFORMED BY

PRINCIPLES OF PERCEPTION AND MEANING

by

ANDREW CYRUS SMITH

Submitted in accordance with the requirements

for the degree of

Doctor of philosophy

in the subject

Computer Science

at the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: PROF JH GELDERBLOM

February 2017

DECLARATION

Name: Andrew Cyrus Smith

Student number: 7075006

Degree: PhD (Computer Science)

A TANGIBLE PROGRAMMING ENVIRONMENT MODEL

INFORMED BY

PRINCIPLES OF PERCEPTION AND MEANING

I declare that the above thesis is my own work and that all the sources that I have used or quoted have been indicated and acknowledged by means of complete references.

___________________________ _________________________

SIGNATURE DATE

Dedicated to Michelle, Angelique, and my parents.

Acknowledgements

Oh, the places you'll go.
(Dr Seuss)

I am back!

It has been a long journey that took me through unchartered waters, and now I have returned with new knowledge.

It was not easy. I prepared well by consulting literature, researchers, colleagues, and family. Yet, when my research ship left the harbour and I sailed off to find that hidden gem I could not anticipate that my hometown would be annexed and my crew change.

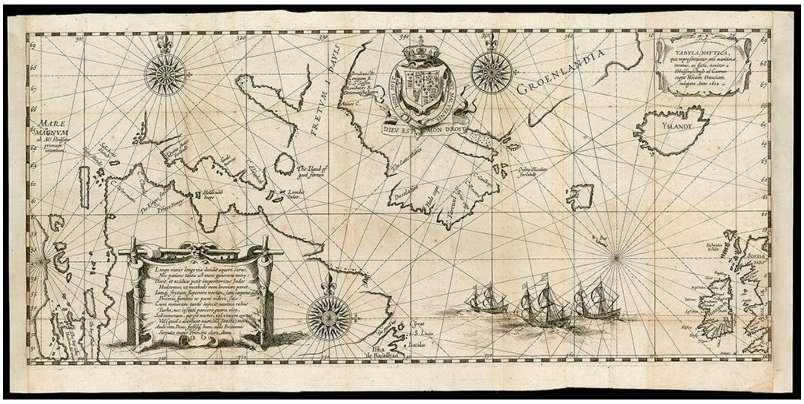
My supervisor Professor Gelderblom understood my curiosity and supported me to the end. For this, I will be forever grateful. Thank you, Helene!

I wish to thank Professor Paula Kotze who helped set my compass and see me set sail, Professor Adele Botha for probing my logbook, and Professor Marlien Herselman for reminding me that I'm defending the stake I had set in the ground.

My friends, colleagues, participants and their parents, and fellow researchers queried my work and thereby helped me construct a usable mental model. They are, in no particular order, Mario Marais, Anita van Deventer, Thato Foko, David Johnson, Nomusa Dlodlo, Alta van der Merwe, Erkki Sutinen, Johannes Cronje, Michael Horn, Brygg Ullmer, Ron Beyers, Elise van den Hoven, Chris Krause, Grant Cambridge, Ajay Makan, Rensie van Rensburg, Laurie Barwell, Karel van der Walt, George Sibiya, Tshepo Toloane, Ntsibane Ntlatlapa, Seugnet Blignaut, Henrik Hautop Lund, Corne Kruger, Mduduzi Mudumbe, Mpho Nkuna, Michaella Janse van Vuuren, Azwifarwi Vele and Barend Taute.

I am particularly thankful to Merryl Ford who gave me the resources to participate in the rewarding world of research. Quentin Williams encouraged me to embark on this journey while Hina Patel and Mathew Chetty kept the ship afloat. Three CSIR Meraka Institute groups funded this study and they are ICT for Education Youth and Gender, Young Researchers, and Internet of Things. Thank you all.

Finally, my eternal appreciation goes to the loves of my life, Michelle and Angy, who sacrificed so much to give me the time and space to complete this work.



Map: Hessel Gerritsz, *Tabula nautica qua repraesentantur orae maritimae, meatus, ac freta, noviter a H Hudsono, Anglo ad caurum supra Novam Franciam ndagata anno 1612, Amsterdam, 1612,* Library and Archives Canada NMC 19228

SUMMARY

It is a fundamental Human-Computer Interaction problem to design a tangible programming environment for use by multiple persons that can also be individualised. This problem has its origin in the phenomenon that the meaning an object holds can vary across individuals. The Semiotics Research Domain studies the meaning objects hold. This research investigated a solution based on the user designing aspects of the environment at a time after it has been made operational and when the development team is no longer available to implement the user's design requirements.

Also considered is how objects can be positioned so that the collection of objects is interpreted as a program. I therefore explored how some of the principles of relative positioning of objects, as researched in the domains of Psychology and Art, could be applied to tangible programming environments. This study applied the Gestalt principle of perceptual grouping by proximity to the design of tangible programming environments to determine if a tangible programming environment is possible in which the relative positions of personally meaningful objects define the program. I did this by applying the Design Science Research methodology with five iterations and evaluations involving children.

The outcome is a model of a Tangible Programming Environment that includes Gestalt principles and Semiotic theory; Semiotic theory explains that the user can choose a physical representation of the program element that carries personal meaning whereas the Gestalt principle of grouping by proximity predicts that objects can be arranged to appear as if linked to each other.

Keywords:

Gestalt principles, grouping by proximity, perception, personally meaningful, programming, programming languages, Semiotic theory, Signs, tangible program, tangible user interface.

PUBLICATIONS FLOWING FROM THIS RESEARCH

I published research results during the course of this project. Papers included in peer reviewed conference and workshop proceedings since 2010 are listed here and cited in the thesis body.

Reitsma, L., Smith, A. & Hoven, E. van den, 2013. StoryBeads: Preserving indigenous knowledge through tangible interaction design. In International conference on culture and computing. IEEE, pp. 79–85.

Smith, A.C., 2010a. Dialando: Tangible programming for the novice with Scratch, Processing and Arduino. In 6th International workshop on technology for innovation and education in developing countries (TEDC). Available at: http://hdl.handle.net/10204/4048.

Smith, A.C., 2010b. Tangible interfaces for tangible robots. In E. Hall, ed. Advances in robot manipulators. Croatia: InTech, pp. 607–624. Available at: http://www.intechopen.com/books/advances-in-robot-manipulators/tangible-interfaces-for-tangible-robots.

Smith, A.C., 2014a. Cluster-based tangible programming. In Fourth international conference on digital information and communication technology and it's applications (DICTAP). IEEE, pp. 405–410. Available at: http://ieeexplore.ieee.org/.

Smith, A.C., 2014b. Rock garden programming: Programming in the physical world. In Fourth international conference on digital information and communication technology and it's applications (DICTAP). IEEE, pp. 430–434. Available at: http://ieeexplore.ieee.org/.

Smith, A.C., Dlodlo, N. & Jere, N., 2016. Towards an internet of things tangible program environment supported by indigenous African artefacts. In Proceedings of the first African conference on human computer interaction. AfriCHI'16. Nairobi, Kenya: ACM, pp. 176–181. Available at: http://doi.acm.org/10.1145/2998581.2998599.

Smith, A.C. & Gelderblom, J.H., 2013a. The building is the program. In Peripheral interaction: Embedding HCI in everyday life, A volume in the workshop proceedings series of the INTERACT 2013 conference. Workshop at INTERACT 2013. Available at: http://www.peripheralinteraction.com/interact/paper/Workshop_PI_Smith.pdf.

Smith, A.C. & Gelderblom, J.H., 2013b. Towards a tangible web: Using physical objects to access and manipulate the Internet of Things. In Proceedings of the 15th annual conference on World Wide Web applications. Available at: http://hdl.handle.net/10204/7367.

Smith, A.C. & Gelderblom, J.H., 2016. End user programming with personally meaningful objects. In L. Church, ed. Proceedings of the 27th annual workshop of the psychology of programming interest group - PPIG 2016. St. Catharine's College, University of Cambridge, UK. Available at: https://drive.google.com/file/d/0B-7G3GOHucdiMTNjdEtyMC1qWjQ/view.

Smith, A.C. & Kotzé, P., 2010. Indigenous African artefacts: Can they serve as tangible programming objects? In IST-Africa 2010 conference proceedings. IEEE Conference Publications. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5753043.

Smith, A.C., Kotzé, P. & Gelderblom, H., 2011a. General design methodology applied to the research domain of physical programming for computer illiterates. In Design, development & research conference. Faculty of Informatics and Design, Cape Peninsula University of Technology. Available at: http://hdl.handle.net/10204/5423.

Smith, A.C., Reitsma, L., Hoven, E. van den, Kotzé, P. & Coetzee, L., 2011b. Towards preserving indigenous oral stories using tangible objects. In 2011 Second international conference on culture and computing. Kyoto, Japan: IEEE Conference Publications, pp. 86–91. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6103215.

Smith, A.C., Springhorn, H., Mulligan, S.B., Weber, I. & Norris, J., 2011c. tactusLogic: Programming using physical objects. In IST-Africa 2011 conference proceedings. IEEE Conference Publications. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6107337.

Table of contents

CHAPTER 1 INTRODUCTION 1

1.1 Introduction 2

1.2 Problem statement and purpose of this research 3

1.3 Rationale of this study 5

1.4 Research thesis statement 5

1.5 Research aim 6

1.6 Research objectives 6

1.7 Research questions 6

1.8 Definition of terms 7

1.9 Assumptions 7

1.10 Methodology overview 7

1.11 Literature summary 9

1.11.1 Objects and their meaning 9

1.11.2 Personally meaningful objects 10

1.11.3 Gestalt 10

1.11.4 Tangible objects and computing systems 11

1.11.5 Tangible programming 12

1.12 Delineations and limitations 13

1.13 Organisation of the remainder of the thesis 13

1.14 Conclusion 14

CHAPTER 2 THEORETICAL BACKGROUND 15

2.1 Introduction 16

2.2 Perceptual organisation 16

2.2.1 The general Gestalt principles of perceptual organisation 19

2.2.2 Gestalts 20

2.2.3 Parts and wholes 21

2.2.4 Gestalt principle of Prägnanz 22

2.2.5 Gestalt principle of figure-ground separation 23

2.2.6 Gestalt principles of perceptual grouping 24

2.2.6.1 The principle of perceptual grouping by proximity 25

2.2.6.2 The principle of perceptual grouping by similarity 26

2.2.6.3 The principle of perceptual grouping by good continuation 26

2.2.6.4 The principle of perceptual grouping by closure 26

2.2.6.5 The principle of perceptual grouping by common fate 27

2.2.6.6 The principle of perceptual grouping by synchrony 27

2.2.6.7 The principle of perceptual grouping by parallelism 28

2.2.6.8 The principle of perceptual grouping by symmetry 28

2.2.6.9 The principle of perceptual grouping by uniform connectedness 28

2.2.6.10 The principle of perceptual grouping by common region 28

2.2.7 Conclusion to this section 28

2.3 Signs 29

2.3.1 Semiotics 30

2.3.2 Saussure and Peirce 30

2.3.3 Peirce’s model 31

2.3.3.1 The sign-representamen 31

2.3.3.2 The semiotic object 32

2.3.3.3 The interpretant 33

2.3.3.4 Phenomenological categories of experience 34

2.3.3.5 Ontological categories of being 34

2.3.3.6 Peirce’s first trichotomy 35

2.3.3.7 Peirce’s second trichotomy 36

2.3.3.8 Peirce’s third trichotomy 36

2.3.4 Conclusion to this section 37

2.4 Programming 37

2.4.1 Psychology of programming 38

2.4.2 Classification of programmable systems 39

2.4.3 Generations of languages 40

2.4.3.1 First-generation languages 41

2.4.3.2 Second-generation languages 42

2.4.3.3 Third-generation languages 42

2.4.4 Language format 43

2.4.5 Conclusion to this section 44

2.5 Perceptual organisation, signs, and programming 45

2.5.1 Perceptual organisation and semiosis 46

2.5.2 Saussure's linguistic sign model applied to program code 47

2.5.3 Gestalt principles and program code 48

2.5.4 Peirce's sign model and Gestalt principles applied to tangible program elements 48

2.5.5 Conclusion to this section 49

2.6 Conclusion 50

CHAPTER 3 LITERATURE REVIEW: TANGIBLE OBJECTS 51

3.1 Introduction 52

3.1.1 Nomenclature 52

3.2 Tangible objects and computer interaction 53

3.2.1 Ishii's tangible user interface model 53

3.2.2 The meaning that objects hold 57

3.2.3 Supportive technologies 59

3.2.3.1 Technologies that support untethered passive tangible systems 61

3.2.3.2 Technologies that support untethered active tangible systems 62

3.2.4 Supportive gesture modalities 63

3.2.4.1 Gestures for binding data 64

3.2.4.2 Gestures for unbinding data 64

3.2.4.3 Gestures for modifying data 65

3.2.5 Tangible objects and data 66

3.2.5.1 Tangible objects for generating data 67

3.2.5.2 Tangible objects for interacting with data 67

3.2.5.3 Tangible objects as containers of data 70

3.2.5.4 Tangible objects as pointers to data 71

3.2.5.5 Tangible objects as representations of data 71

3.2.6 Tangible objects and digital models 72

3.2.7 Conclusion to this section 72

3.3 User-created tangible objects 73

3.3.1 Motivation for user-created tangible objects 74

3.3.2 Craft materials for, and methods of, constructing tangible objects 75

3.3.3 Environments for creating personally meaningful tangible objects 76

3.3.4 Conclusion to this section 78

3.4 Discussion 78

3.5 Conclusion 79

CHAPTER 4 LITERATURE REVIEW: TANGIBLE PROGRAMMING 81

4.1 Introduction 82

4.2 Distinguishing between physical and tangible programming 83

4.2.1 Physical programming 83

4.2.2 Tangible programming 84

4.2.2.1 The tangible programming environment 84

4.2.2.2 Useful characteristics of tangible programs 85

4.2.2.3 Tangible programming styles 85

4.2.2.4 Tangible programming modalities 87

4.2.3 Conclusion to this section 87

4.3 Tangible programming environments 88

4.3.1 Environments that incorporate supplied objects 88

4.3.1.1 Objects utilised for their one dimensional property 88

4.3.1.2 Objects utilised for their two dimensional properties 96

4.3.1.3 Objects utilised for their three dimensional properties 97

4.3.2 Environments that incorporate objects with personal meaning 98

4.3.2.1 StoryRoom 98

4.3.2.2 Quilt Snaps 105

4.3.2.3 Diorama Table 107

4.3.3 Conclusion to this section 108

4.4 Discussion 109

4.4.1 The Gestalt principle of good continuation 109

4.4.2 The Gestalt principle of grouping by common region 110

4.4.3 The Gestalt principle of grouping by proximity 110

4.4.4 Personally meaningful objects 112

4.5 Conclusion 113

CHAPTER 5 RESEARCH METHODOLOGY 115

5.1 Introduction 116

5.2 Philosophical stance 116

5.2.1 Ontological nature of reality 116

5.2.2 Epistemological assumption 117

5.2.3 Human nature 117

5.3 Methodology and methods 118

5.3.1 The nature of Design Science Research outputs 119

5.3.2 The basic activities of Design Science Research 119

5.3.3 Vaishnavi and Kuechler’s process model overview 119

5.3.4 Vaishnavi and Kuechler’s Knowledge Flows and Process steps 121

5.3.4.1 Awareness of Problem 122

5.3.4.2 Suggestion step 122

5.3.4.3 Development step 123

5.3.4.4 Evaluation step 123

5.3.4.5 Conclusion step 126

5.3.5 Data collection and analysis 126

5.3.5.1 Science fairs 126

5.3.5.2 TekkiKids 128

5.3.5.3 Kindergartens 129

5.3.5.4 Interactive exhibition 129

5.3.5.5 Laboratory with children 129

5.3.5.6 Laboratory without children 129

5.3.6 Ethical procedures 130

5.4 Conclusion 131

CHAPTER 6 DESIGN, IMPLEMENTATION, AND EVALUATION 133

6.1 Introduction 134

6.2 Iteration one: GameBlocks I 137

6.2.1 System design 138

6.2.2 Tangibles design 138

6.2.3 Evaluation 139

6.2.3.1 Evaluation design 140

6.2.3.2 Evaluation results 141

6.2.4 Discussion 141

6.3 Second iteration: GameBlocks II 142

6.3.1 System design 143

6.3.2 Technical description 144

6.3.3 Evaluation 145

6.3.3.1 Evaluation design 145

6.3.3.2 Evaluation results 148

6.3.4 Discussion 148

6.4 Design iteration three: RockBlocks 149

6.4.1 Design considerations 150

6.4.2 System design 151

6.4.3 Tangibles design 151

6.4.4 Evaluation 152

6.4.4.1 Evaluation design 152

6.4.4.2 Evaluation results 153

6.4.5 Discussion 153

6.5 Fourth iteration: Dialando 155

6.5.1 System design 155

6.5.2 Tangibles design 157

6.5.3 Discussion 157

6.6 Design iteration five: T-Logo 158

6.6.1 Design process 158

6.6.2 Evaluation methodology 159

6.6.3 Tangible program concepts 160

6.6.3.1 Physical and digital constructions 160

6.6.3.2 Mapping between objects, actions, and parameters 161

6.6.3.3 Clustering, Cluster Marker, and Cluster Marker Zone 162

6.6.4 Clustering topologies 163

6.6.4.1 Topology #1: Grouping by common region 163

6.6.4.2 Topology #2: Grouping by proximity 166

6.6.5 Language and system usage 166

6.6.6 Programming rules 167

6.6.7 System components 167

6.6.8 Information transfer from the physical to digital domain 167

6.6.9 Software modes 167

6.6.9.1 Map mode 168

6.6.9.2 Program Construct mode 171

6.6.9.3 Program Interpret and Execute modes 171

6.6.10 Program example 172

6.6.11 Evaluation 173

6.6.11.1 Laboratory evaluation design 174

6.6.11.2 Laboratory evaluation results 175

6.6.11.3 Comparison to the TERN language 176

6.6.11.4 TERN language elements 178

6.6.11.5 T-Logo language elements 178

6.6.11.6 Logic expression using the T-Logo language 179

6.6.12 Critical discussion 181

6.7 Conclusion 182

CHAPTER 7 RESEARCH CONTRIBUTION: A MODEL FOR A TANGIBLE PROGRAMMING ENVIRONMENT 185

7.1 Introduction 186

7.2 Programming environment model 186

7.2.1 Language architect 187

7.2.2 Software developer 188

7.2.3 User 189

7.2.4 Computing system 191

7.3 Applying the model 192

7.3.1 First scenario 192

7.3.2 Second scenario 194

7.3.3 Third scenario 195

7.3.4 The derived language 196

7.4 Conclusion 196

CHAPTER 8 FINAL DISCUSSION AND CONCLUSIONS 197

8.1 Introduction 198

8.2 Summary of findings 199

8.3 Conclusions 200

8.4 Summary of contributions 202

8.4.1 New knowledge...202

8.4.2 Implications of the new knowledge 203

8.5 Suggestions for further research 204

REFERENCES 207

APPENDIX A Research ethics training 231

APPENDIX B Second iteration ethical clearance form: SciFest Africa 2008 233

APPENDIX C Participant instructions and consent form: SciFest Africa 2008 239

APPENDIX D TekkiKids: Statement regarding informed assent and consent 241

APPENDIX E TekkiKids: Invitation to participate and consent form 243

APPENDIX F Second iteration workshop invitation pamphlet: SciFest Africa 2008 245

APPENDIX G Second iteration partial evaluation form: Meraka Institute 2007 247

APPENDIX H Second iteration evaluation form: SciFest Africa and Science Unlimited 2007 249

APPENDIX I Third iteration user evaluation forms: CSIR and SciFest Africa 2008 251

List of figures

Figure 1-1 Document structure........1

Figure 2-1 Document structure........15

Figure 2-2 Relevant research domains and their relationships........16

Figure 2-3 An arrangement of coloured pencils is perceived as a butterfly when viewed from a particular angle........18

Figure 2-4 The general Gestalt principles of perceptual organisation, with examples........20

Figure 2-5 Juxtaposed parallelograms appear to differ in size........20

Figure 2-6 Parts and wholes........21

Figure 2-7 Perceiving a simple image........23

Figure 2-8 Grouping perception varies according to rotation angle........25

Figure 2-9 Peirce’s trichotomies of signs in context........35

Figure 2-10 Programmable systems........41

Figure 2-11 Solving a problem using first, second, and third-generation programming languages........43

Figure 2-12 A comparative example of a fixed-format language........43

Figure 2-13 Two examples of valid vertical allignment........44

Figure 2-14 Two examples of programmer-applied discretion in the case of a free-form language........44

Figure 2-15 Interplay between perceptual organisation and semiosis........46

Figure 2-16 The perceptual organisation of a single stimulus can lead to multiple interpretants........46

Figure 2-17 Saussure’s linguistic sign model applied to three generations of computer program languages........47

Figure 2-18 Examples that demonstrate perceptual grouping by common region, common fate, and closure........49

Figure 2-19 Coding a program element by grouping a qualisign with an icon sign........49

Figure 3-1 Document structure........51

Figure 3-2 Chapter outline........52

Figure 3-3 Ishii’s basic tangible user interface model........54

Figure 3-4 A selection of mappings between graphical user interfaces and tangible user interfaces........56

Figure 3-5 A continuum of the meaning that objects hold........57

Figure 3-6 The SurfaceWare drinking glass 58
Figure 3-7 Three objects represent three nouns 58
Figure 3-8 Digital Dream Lab objects interlock to form a tangible programming object 59
Figure 3-9 My stack and system perspectives on the relationships that exist in tangible interaction systems 60
Figure 3-10 My classification matrix of mechanisms in tangible systems mapped according to data exchange, and identity encoding/position encoding 61
Figure 3-11 The Siftables interaction language 64
Figure 3-12 Three SenseTable mechanisms to bind physical objects to data 65
Figure 3-13 A SenseTable puck and rotation knob combination 65
Figure 3-14 The distance between two physical objects alters data 66
Figure 3-15 A battery serves as a mental reminder to its user 66
Figure 3-16 Krippendorff's model of the user and the designer's view of an artefact 73
Figure 3-17 Krippendorff's adapted model reflects the case where and individual is both the designer and the user 74
Figure 3-18 An improvisational interface 75
Figure 3-19 A marker as an improvisational program element interface 75
Figure 3-20 A simple character design supports observer self-identification better than a realistic character design 76
Figure 3-21 Masks with little intrinsic character 77
Figure 3-22 StoryKits integrates high-tech and low-tech items 78
Figure 4-1 Document structure 81
Figure 4-2 Chapter outline 83
Figure 4-3 An example of a tangible database query 86
Figure 4-4 The Gestalt principle of perceptual grouping applied to the tangible database query style 86
Figure 4-5 Gestalt principles applied in an arrangement 90
Figure 4-6 The Turtan system incorporates the Gestalt principle of good continuation 93
Figure 4-7 The ReacTable and the Gestalt principle of perceptual grouping by proximity 94
Figure 4-8 The SiteView tangible programming environment 94
Figure 4-9 My application of the Gestalt principle of perceptual grouping by proximity to SiteView 95
Figure 4-10 The instance block is functionally an exact copy of the component block 97
Figure 4-11 A selection of StoryRoom physical programming elements 99

Figure 4-12 A combined iconic and symbolic depiction of the first program rule 100
Figure 4-13 A combined iconic and symbolic expression of program Rule 1 and Rule 2 100
Figure 4-14 Temporal, iconic, and symbolic grouping identified in Rule 1....................................... 101
Figure 4-15 Overlap implies relationships between objects.. 102
Figure 4-16 Alternative Rule 1 sequences 103
Figure 4-17 Alternative Rule 1 groupings 103
Figure 4-18 Three conceptual iconic sentences.. 104
Figure 4-19 My depiction of Montemayor's conceptual iconic sentence using non-descript symbols.. 104
Figure 4-20 My depiction of Montemayor's conceptual iconic sentence with tangible objects........ 105
Figure 4-21 A Quilt Snaps patch with its associated information flow 106
Figure 4-22 Three Quilt Snaps program examples and their information flow................................. 106
Figure 4-23 Diorama Table with everyday objects and projected animation 107
Figure 4-24 Grouping-by-temporal-common-region and grouping-by-temporal-proximity............. 112
Figure 5-1 Document structure... 115
Figure 5-2 The Design Science Research process model with associated cognition 120
Figure 5-3 The generic design iteration process as applied in the current research......................... 122
Figure 5-4 The role of children in this study 125
Figure 5-5 My evaluation strategy in context of the Framework for Evaluation in Design Science (FEDS) 125
Figure 6-1 Document structure... 133
Figure 6-2 The five design iterations and their properties 135
Figure 6-3 The five iterations in context of the Design Science Research methodology 136
Figure 6-4 The knowledge that informed the first iteration and the results..................................... 137
Figure 6-5 Cube, programming tray, and toy robot.. 138
Figure 6-6 The sign, colour, function, and magnet placement for each cube type 139
Figure 6-7 A seven-part program and the associated result of each instruction 139
Figure 6-8 Evaluations in Pretoria and Grahamstown 139
Figure 6-9 Evaluation aids and activity 140
Figure 6-10 Suggested design improvements... 142
Figure 6-11 The knowledge that informed the second iteration and the results............................. 143
Figure 6-12 Programs constructed using Tortis, Turtle Talk, and GameBlocks II 143
Figure 6-13 The sign set that defines the language elements 144
Figure 6-14 A program, its TURTLE TALK equivalent, and execution results 144

Figure 6-15 Foam cube, active square under construction, and toy car 145

Figure 6-16 The physical configuration, the challenge, and two solutions 147

Figure 6-17 Program design, construction, and debugging activities 148

Figure 6-18 “Turn-and-go” signs 149

Figure 6-19 “Turn-and-stop” signs 149

Figure 6-20 Zulu fertility doll and beaded apparel 150

Figure 6-21 The knowledge that informed the third iteration and the results 150

Figure 6-22 Natural rock and hand tools, the processed rock, programming objects, and Sensing Tiles 151

Figure 6-23 A program that exhibits the sign set 152

Figure 6-24 Constructing a RockBlock program 153

Figure 6-25 A tangible program object crafted from clay and the associated conceptual programming environment 154

Figure 6-26 System components 156

Figure 6-27 The fourth iteration artefact in use and a program example in execution 156

Figure 6-28 The knowledge that informed the fourth iteration and the results 157

Figure 6-29 The knowledge that informed the fifth iteration and the results 159

Figure 6-30 The four key steps involved in constructing a text-based computer program 160

Figure 6-31 Four steps in constructing a tangible program 161

Figure 6-32 The general case of defining the object’s position on the two dimensional construction surface 161

Figure 6-33 Physical clustering 162

Figure 6-34 The relationship between the Cluster Marker, its radius, and the Cluster Marker Zone 163

Figure 6-35 Hoops and markers 163

Figure 6-36 Algorithm ambiguity in the case of infinitely narrow rings 165

Figure 6-37 Three wide hoops with markers 165

Figure 6-38 Wide hoops eliminate the initial ambiguity 165

Figure 6-39 Apparent ambiguity in the case of three wide hoops 166

Figure 6-40 The T-Logo physical components 168

Figure 6-41 The four T-Logo software modes 169

Figure 6-42 An example of a T-Logo mapping activity 169

Figure 6-43 The on-screen appearance of the seven user-selectable mapping options 170

Figure 6-44 The sequence in which objects on the construction surface are processed 172

Figure 6-45 A program, the interpretation order, and the corresponding Processing language statements.......173
Figure 6-46 An example to illustrate how the logical values of two parameter objects are totalled 174
Figure 6-47 A Processing language software routine and resultant graphics 174
Figure 6-48 Objects and the commands and values they represent 174
Figure 6-49 A tangible program with annotations and the execution result 175
Figure 6-50 TERN puzzle and cubes examples.......178
Figure 6-51 Example of a linear program.......180
Figure 6-52 Example of a conditional statement in a loop 180
Figure 6-53 Example of the logic AND conditional statement.......181
Figure 6-54 Design iterations, their target groups, and the nature of their outputs 183
Figure 7-1 Document structure.......185
Figure 7-2 Model of a tangible programming environment.......187
Figure 7-3 Language architect in the model 188
Figure 7-4 Software developer in the model 189
Figure 7-5 User in the model 190
Figure 7-6 Computing system in the model.......192
Figure 7-7 Examples of objects and a program to control the temperature.......194
Figure 7-8 Examples of objects and a program to make coffee 194
Figure 7-9 Examples of objects and a program to control a security light 195
Figure 8-1 Document structure.......197
Figure 8-2 Figurines as programs.......205

List of tables

Table 1-1 Definition of terms ... 7

Table 1-2 Research objectives and applicable methods ... 8

Table 3-1 A selection of mappings between Graphical User Interfaces and Tangible User Interfaces ... 56

Table 4-1 Summary table of physical ... 89

Table 4-2 Summary table of physical strings ... 90

Table 4-3 Summary table of Electronic Blocks ... 91

Table 4-4 Summary table of TORTIS ... 91

Table 4-5 Summary table of Robo-Blocks ... 92

Table 4-6 Summary table of Turtan ... 93

Table 4-7 Summary table of ReacTable ... 94

Table 4-8 Summary table of SiteView ... 95

Table 4-9 Summary table of Media Cubes ... 96

Table 4-10 Summary table of Digital Dream Lab ... 97

Table 4-11 Summary table of roBlocks ... 98

Table 4-12 Summary table of StoryRoom ... 105

Table 4-13 Summary table of Quilt Snaps ... 106

Table 4-14 Summary table of Diorama Table ... 107

Table 6-1 Tern actions and T-Logo objects ... 176

Table 6-2 Tern and T-Logo control structures ... 177

Table 6-3 TERN sensing elements and their T-Logo equivalents ... 177

Table 6-4 Miscellaneous TERN and T-Logo elements ... 177

CHAPTER 1

INTRODUCTION

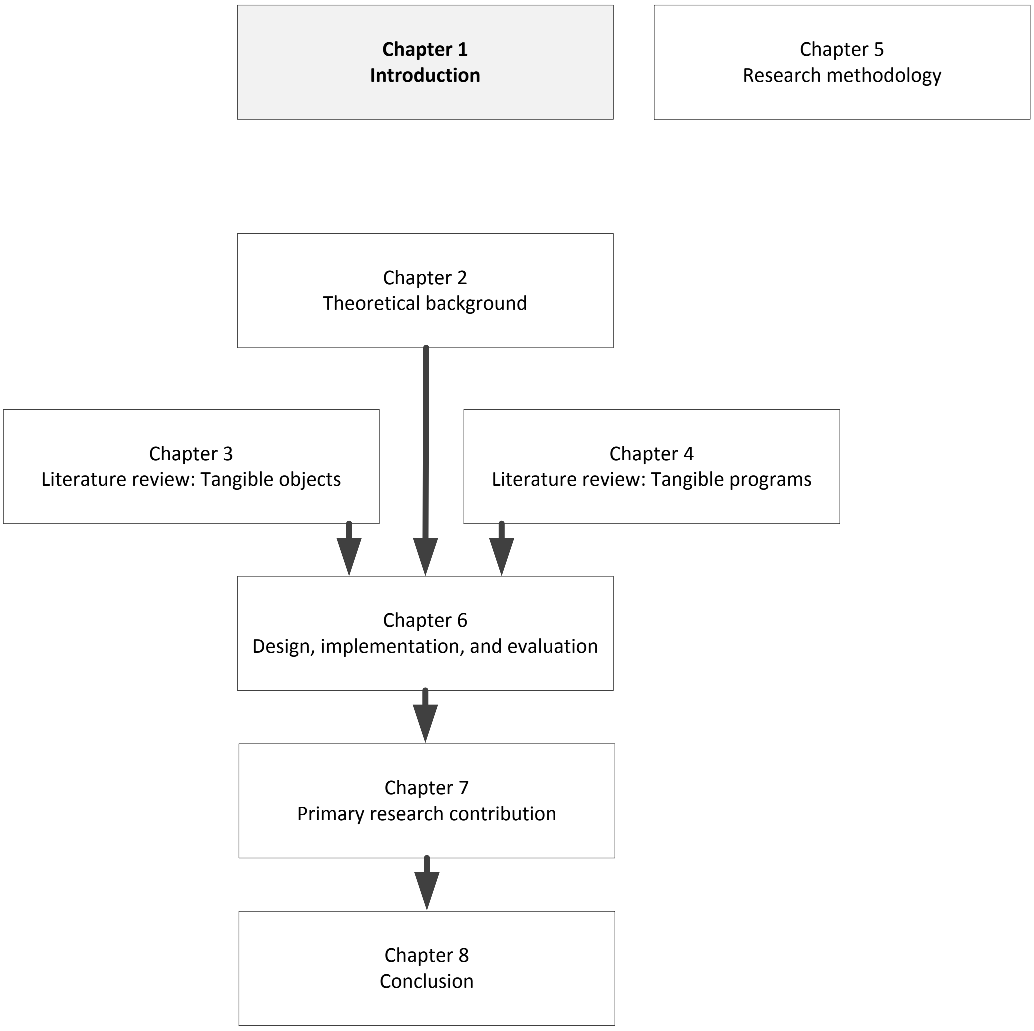

Chapter 1
Introduction
Chapter 5
Research methodology
Chapter 2
Theoretical background
Chapter 3
Literature review: Tangible objects
Chapter 4
Literature review: Tangible programs
Chapter 6
Design, implementation, and evaluation
Chapter 7
Primary research contribution
Chapter 8
Conclusion

Figure 1-1 Document structure

1.1 Introduction

Human beings have the ability to create almost any object to their liking and have practiced doing so for millennia. By the twentieth century, we had acquired sufficient knowledge to create electromechanical computational devices. Vacuum valve technology, the discreet transistor, and the integrated circuit have subsequently replaced most of the electromechanical components in the computer. The modern computational device is colloquially referred to as a computer and the logic that determines its operation is called a program. The act of creating a program is called programming and the person doing the programming is called a programmer.

The first tool with which the computer could be programmed was designed by the small community of scientists for their own use. As computer technology changed, so too did the tools for programming them. The collection of tools used in the creation of a program is called the programming environment. Programming in the 1940's was done by manually toggling electrical switches and changing wired connections on a panel. Switches were later replaced with the invention of the punch card. Using this environment, a single program instruction took the form of a symbol comprising holes in a sheet of card stock, and a program consisted of a collection of cards. By the early 1980's computer technology had evolved to the point where a combination of computer keyboard and visual display unit provided all the computer interaction a programmer needed to program. Symbols like those on a typewriter could now be projected onto a display and a program would comprise of a carefully constructed collection of these symbols. In contrast to the card-based programming environment, the keyboard and visual display programming environment did not require a tangible medium to contain the program; rather, the program had no physical form. An alternative programming environment was made possible with the introduction of the mouse and graphic display. This resulted in an expanded symbol set with line drawings. Again, the program had no intrinsic physical form. More recently, researchers started exploring programming environments in which the program takes the form of a static arrangement of tangible objects; such a program is called a *tangible program*. These programming environments all incorporate fixed, predefined signs with which the programmer constructs his program.

Instead of using signs designed by someone else, I am interested in an environment in which the programmer can use signs of which the designs are based on his personal experience. The Merriam-Webster (Merriam-Webster n.d.) and American Heritage (Patwell 1992) dictionaries describe the word "personally" as being something that involves only the individual's experience, excluding others. Using this adjective, I can state that I am interested in designs that involve the programmer personally. These dictionaries also define "meaningful" as something that conveys a significant idea.

By combing these two definitions, I can use the terminology “personally meaningful” to describe an idea that is the result of an individual’s experience.

This study explored the combined application of signs that are personally meaningful and the Gestalt principle of perceptual grouping by proximity to the design of tangible programming environments. My aim was therefore to develop a model that guides tangible programming environment designs in which the programmer is free to use signs of his own choice and based on his own experience.

1.2 Problem statement and purpose of this research

Fischer (2001) notes that the research domain of Human-Computer Interaction (HCI) not only studies the interaction between humans and computers but also the relationships that exist between humans and computers. In my study, this relationship takes the form of signs (realised as tangible programming objects) that hold personal meaning to the programmer. I refer to these as personally meaningful signs. Without this relationship, the computer is unable to correctly interpret a program that has been constructed using a collection of personally meaningful signs. Therefore, the combined application of personally meaningful signs and grouping by proximity in tangible programming environments is an HCI problem.

Fischer also argues that determining the design requirements for the typical user (or the programmer as is the case in this study) is not possible since such a user does not exist. Rather than designing for the non-existing typical user, Fischer reports that designers have developed techniques to gain an understanding of the user. Some of these techniques employ the preferences as set by the user, while other techniques draw inferences from the user’s actions. Fischer concludes that one of the fundamental HCI design problems is that of designing a system that will be used by multiple persons, but which must also seem to have been individually designed for each user. As an alternative attempt to address this fundamental HCI design problem, Fischer and Scharff (2000) propose that the user be afforded an opportunity to design aspects of the system at a time after the system has been made operational and when the product development team is no longer available to implement the user’s own design requirements. I call this *Fischer’s proposal* but the problem is that no tangible programming environment model exists that can guide its application.

Whereas Fischer discusses mechanisms for designing a system according to a user’s requirements, Boradkar (2010) considers the meaning an object holds for an individual. Boradkar observes that the meaning an object holds for an individual is not always the same meaning that the same object holds for someone else. I call this *Boradkar’s observation*. To illustrate the point, Boradkar recalls a personal interview he conducted with an executive who is responsible for product development.

This executive observed that his own perception of a product is influenced by his knowledge of the journey that the product travelled during the research, design, and development processes. The executive's view of the finished product is coloured by these processes and therefore differs from the view held by those individuals who are not aware and are not part of the production processes. As an example of how knowledge of the production process may influence an individual's perception of the product, consider one of the production steps in the production of deep fried chicken. Here, a naïve view of the product is that the deep fried chicken is a spicy meal. Yet, to a person knowledgeable of the steps followed in producing this product (Boyd 1994), it may be true that the deep fried chicken product does not bring to mind a spicy meal but instead an act of cruelty.

As explained in the preceding paragraph, the meaning an object holds for one person may be different to the meaning the same object holds for another person. Consequently, if one assumes that the object (deep fried chicken in this case) is a sign, then it would be reasonable to describe it as a personally meaningful sign because the meaning that the object holds for one person may differ from the meaning the same object holds for another person.

As is the case for keyboard, mouse, and screen–based programming environments, tangible programming environments are based on the working relationship that exists between the programmer and the computer, and specifically where this relationship is formed by the common understanding of the signs used in a program constructed by the programmer. Even though the available literature (Anderson, Frankel, Marks, Agarwala, Beardsley, Hodgins, Leigh, Ryall, Sullivan & Yedidia 2000; Blackwell & Hague 2001b; Buechley, Elumeze, Dodson & Eisenberg 2005; Camarata, Do, Johnson & Gross 2002; Do & Gross 2007; Gallardo, Julia & Jorda 2008; Horn 2009; Kitamura, Itoh & Kishino 2001; McNerney 1999; Perlman 1974; Schweikardt & Gross 2007; Sipitakiat & Nusen 2012; Suzuki & Kato 1995a; Wang, Zhang & Wang 2011; Wyeth & Wyeth 2001; Zuckerman 2004) indicate that there exists an interest amongst researchers to develop tangible programming environments, there seems to be little or no research conducted that simultaneously addresses Fischer's proposal and Boradkar's observation. Studies that do report on the design of programming environments in which tangible objects are used in the construction of a tangible program (I call such objects *tangible programming objects*) either do not explicitly report on the user's participation in the design of the objects, or rely on user group participation. No current literature guides the design of tangible programming objects when the individual programmer is given an opportunity to determine what the objects should be or what these should look like. Tangible programming objects that have been derived without the direct input of the particular user are often intended for use by multiple individuals and accompanied by explanations of their meaning. The problem therefore is that no tangible programming environment model exists in which the individual can decide what the objects should look like. Semiotics is a research domain that considers the meaning that objects hold for humans. I therefore addressed this problem by including aspects of Semiotics in my research.

My discussion thus far has only considered the creation of the tangible programming objects. Another dimension worthwhile considering in the design of tangible programming environments is how objects should be placed so that the collection of objects is viewed as a program. Literature on tangible programming environments treats the relative positioning of objects as an engineering challenge. Nonetheless, research domains that consider the significance that relative object positions hold for the individual do exist and these include Psychology and Art. The problem is that no guidelines exist on how to explicitly include relative object positions in the design of tangible programming environments. It would therefore be interesting to explore how some of the principles of relative positioning of objects, as researched in these domains, could be applied to tangible programming environments. Consequently, this study also explored the application of Gestalt principles to the design of tangible programming environments.

To conclude, this study applied Gestalt principles and Semiotic theory to address the problem of deriving a model that can guide the design of tangible programming environments that simultaneously implement Fischer's proposal and address Boradkar's observation.

1.3 Rationale of this study

Although tangible programming environments exist in which the objects are designed with input from users and in which objects are placed in close proximity to each other, no known models exist in which a) the programmer can independently decide on the nature of the objects and b) the Gestalt principle of perceptual grouping by proximity is explicitly applied to the design of tangible programming environments. A study that investigates how a programming environment can be created (within which programmers construct physical programs by applying aspects of the established Gestalt principle of perceptual grouping by proximity as they place objects) could benefit the domain of Computer Science because it will show by means of a model how designers of tangible programming environments can incorporate the Gestalt principle of perceptual grouping by proximity together with personally meaningful programming objects into their own designs.

1.4 Research thesis statement

With this study, I defend the following assertion:

It is possible to derive a model for a tangible programming environment in which the relative positions of personally meaningful objects define the program.

1.5 Research aim

I deduced the following research aim from the research thesis statement:

To derive a model of a tangible programming environment that guides a developer to create an environment in which an arrangement of objects that hold personal meaning to the user defines the program.

1.6 Research objectives

A set of research objectives were derived from the research aim. The research objectives are:

Research Objective **1** To determine a set of program elements suited to a programming environment that incorporates personally meaningful tangible objects.

Research Objective **2** To devise a mechanism by which a personally meaningful tangible object can be used as a program element.

Research Objective **3** To devise a method by which an arrangement of one or more personally meaningful tangible objects can define a program statement.

Research Objective **4** To devise a programming environment in which an arrangement of personally meaningful tangible objects can be interpreted as a program.

Research Objective **5** To develop a model that guides the creation of tangible programing environments where an arrangement of personally meaningful objects defines the program.

1.7 Research questions

I derived the following primary research question from the research objectives:

In the context of existing tangible programming environments and considering how tangible objects are currently used when interacting with data, what are the constructs to incorporate into a model for creating tangible programming environments in which the relative positions of personally meaningful objects define the program, and how do these constructs interact and relate to one another?

I next derived secondary research questions from the primary research question:

a. What program elements are suitable for a tangible programming environment in which the programmer can incorporate personally meaningful tangible objects?

b. How can a user associate personally meaningful tangible objects with program elements?

c. How can the arrangement of these personally meaningful objects be interpreted as program statements?

1.8 Definition of terms

The following is a description of terms used in this thesis that are unusual or not widely understood, including those I developed. Some terms may have meaning in this thesis that differ from that in common use. The list below is provided to ensure uniformity throughout the thesis.

Table 1-1 Definition of terms

Term	Definition
Developer	A person who creates a tangible programming environment.
Meaningful	Something that conveys a significant idea.
Personally	Something that involves only the individual's experience, excluding others.
Personally meaningful	Something that conveys a significant idea and involves only the individual's experience, excluding others.
Personally meaningful tangible object	A tangible object that holds personal significance to the user due to their perceptions and interpretations.
Tangible object	A physical object that has no digital counterpart.
Program element	A sign that represents an action, state, or parameter.
Tangible Program Element	A physical object that is programmatically meaningful and is realised when the object is linked to a digital counterpart.
User	The person who constructs a tangible program.
User-defined	The user is empowered to make certain decisions. For example, the user chooses which object to use and what information to associate with the object.

1.9 Assumptions

I assume that Gestalt principle of perceptual grouping by proximity, as well as the notion of a sign as described by the research domain of semiotics, are valid. Based on the notion of a sign as described by the Semiotics research domain, I therefore assume that the meaning an object holds for a person may differ from the meaning the same object holds for someone else. I also assume that the difference in these meanings is amplified by the object's situation relative to other objects. Semiotics theory and the Gestalt school of thought are discussed in Chapter 2.

1.10 Methodology overview

The study was conducted iteratively using a qualitative (Patton 2002) approach within the Design Science (Hevner, March, Park & Ram 2004) research paradigm, and specifically applied as described by Vaishnavi and Kuechler (2013). I completed five iterations (one in the laboratory and four outside the laboratory) and derived a model based on the new knowledge that emerged. I provide in Chapter 5 details of the research methodology.

The methods applied in achieving the various research objectives, as stated in Section 1.6 and repeated here, are given in Table 1-2.

Table 1-2 Research objectives and applicable methods

Research Objective **1**	To determine a set of elements suited to a programming environment that incorporates personally meaningful tangible objects.	A desktop literature survey of tangible programming paradigms identified generic program elements. Two tangible programming environments were designed, implemented, and evaluated to test the suitability of object materials and object sizes. Craft materials and methods of construction were identified from the literature on tangible programming environments and matched to generic program elements. Using the identified craft materials and methods of construction, I crafted physical objects to represent a subset of the generic program elements according to my interpretation.
Research Objective **2**	To devise a mechanism by which a personally meaningful tangible object can be used as a program element.	A programming environment consisting of a program and physical installation was created that allows an individual programmer to make an association between a tangible object and a program element.
Research Objective **3**	To devise a method by which an arrangement of one or more personally meaningful tangible objects can define a program statement.	A programming environment consisting of a software interpreter and physical installation was created that allows a user to make an association between the perceptual grouping by proximity of objects and a program action.
Research Objective **4**	To devise a programming environment in which an arrangement of personally meaningful tangible objects can be interpreted as a program.	A programming environment consisting of a software interpreter and physical installation was created. The environment can identify program elements and interpret them as statements. The environment was compared to another prominent tangible programming system.
Research Objective **5**	To develop a model that guides the creation of tangible programing environments where an arrangement of personally meaningful objects defines the program.	The theory and principles that describe and predict how individuals assign personal meaning to objects were identified and combined with the constructs that flowed from Research Objective 4.

1.11 Literature summary

I approached the literature review from five viewpoints. First, I considered how humans interpret the objects in their environment. The personal significance an object may hold was the second point of view. I then reflected why humans sometimes perceive objects as interacting with each other. For the fourth viewpoint, I deliberated how physical objects can represent, contain, and manipulate data. Finally, I studied systems in which physical objects are used to construct programs.

1.11.1 Objects and their meaning

Objects surround us and the study of how we interpret them is the research field called Semiotics. Peirce (1935) developed a three-component model to describe the relation between the object, the individual, and the meaning. He referred to these as the representamen, interpreter, and semiotic object respectively. Saussure (1959) proposed an alternative model consisting of a signifier and the signified. He called the object being observed the signifier and the meaning that results he called the signified. In contrast to Peirce, Saussure did not explicitly link the resulting meaning to an individual. Peirce's model is useful when the individual's background is relevant whereas Saussure's is more compact.

Peirce puts it that an object does not carry meaning in itself but meaning is instead constructed in the observer's subconscious mind based on his past experiences (Chandler 2007; Martin & Ringham 2000), experiences with the object (Fiske 1990) or experience with the object type (Palmer 1999). Yet the meaning can change (Souza 2005) over time. The individual's interpretation can also differ from other members of the same cultural group (Barthes 1982). The result is that even if the object has a common meaning within a cultural group, the individual may associate a different meaning with the object. This was my motivation for exploring how the user can make his objects and use them for programming.

Saussure's model is useful when there is little risk that meaning will differ between individuals. For example, when used to describe the C (Kernighan, Ritchie & Ejeklint 1988) programming language where a word such as 'while' is very well defined and no ambiguity can exist. In contrast, Peirce's model is useful when the research topic relates to how the meaning of words and objects can vary from one person to another. In this study, I used Saussure's model to illustrate that a single concept can be expressed in more than one way whereas Peirce's model was useful to show that the meaning conveyed by an object can vary across individuals.

Underkoffler and Ishii (1999) demonstrated that an object's meaning can include attributes, nouns, and verbs. Oh, Deshmane, Li, Han, Stewart, Tsai, Xu and Oakley's (2013) Digital Dream Lab objects are examples of nouns, adjectives, and verbs. I identified that objects can also serve to mean a quantity, adjective, and an adverb. Finally, objects can represent discreet quantities or continuous values as in Dietz and Eidelson's (2009) glass.

1.11.2 Personally meaningful objects

Objects are most often designed by persons other than the user and consequently the meaning objects hold can vary according to the user. Design methodologies such as Druin's (2002) do attempt to personalise the designed object but the result remains a compromise of all the participants' inputs. Another result is that the meaning of an object may not be obvious and Patten (2005) reports on an instance where the user modified the appearance of an object to better represent a given concept for him. Krippendorff's (1989) model illustrates the problem by highlighting that the user and the designer are often distinct persons. I addressed this problem by letting the user design personalised programming objects. To this end I found McCloud's (1994) comment useful. He states that a simple object design is better suited than a feature-rich option when the objective is for the user to identify with it. From this, I deduced that personally meaningful objects are best made using materials that hold little intrinsic meaning. I therefore based my final artefacts on wooden blocks, dowels, clay, and paper.

Published research describes systems in which the user designs his objects. For example, Sanders (2000) created tools with which the user may fashion tangible representations of ideas while Sherman, Druin, Montemayor, Farber, Platner, Simms, Porteous, Alborzi, Best, Hammer, Kruskal, Matthews, Rhodes, Cosans and Lal (2001) report on having users create objects with which to interact with data. Story Room (Alborzi, Druin, Montemayor, Platner, Porteous, Sherman, Boltman, Taxén, Best, Hammer, Kruskal, Lal, Schwenn, Sumida, Wagner, et al. 2000), Quilt Snaps (Buechley n.d.), and Diorama Table (Oizumi, Mikami, Sasada & Ubukata 2007) do the same for programming environments. My research differs in that I explicitly incorporate Gestalt principles into the programming environment.

1.11.3 Gestalt

What we perceive is sometimes different to what our senses detect and Gestalt principles help describe this phenomenon. For example, when we observe two objects close together we tend to group them together (principle of grouping by proximity). In addition, when we drive along a long road and there appears a junction to the right, we do not consider the path to the right as a continuation of the current one. Gestaltists call this the principle of good continuation and I incorporated this principle in my artefact designs.

Objects are not always observed separate from their environment, including other objects. The way they are viewed along with other objects can be explained by Gestalt principles. We often consider the placement of one object relative to another as significant, for example in board games including chess. The arrangement can also affect the way a system behaves. Some HCI systems are based on the notion that the meaning an object holds depends on their positions relative to other objects (Gorbet & Orth 1997b; Marco 2011). Beckmann and Dey's (2003) SiteView include examples of objects that do not hold individual value. The distance between objects can also hold meaning and Patten, Recht and Ishii (2006) use the distance between objects to represent a numerical value. Yet some objects can hold value independent of others. For example, Dietz and Eidelson‘s (2009) SurfaceWare drinking glass indicates a numerical value and Mazalek's (2001) genie bottles can function on their own or be combined with others.

Two objects can be used together to modify data and Patten, Ishii, Hines and Pangaro (2001) and Patten et al. (2006) combine a data container with a rotating knob to modify digital data. Also, a physical object can be associated with its digital counterpart or another object by considering the position of objects relative to projected text, an area, or the other object (Oh et al. 2013; Patten et al. 2001). Associations like these are predictable using the Gestalt principle of grouping by proximity and my final artefact design is based on this property. Finally, physical gestures can also be used to group data associated with multiple objects and assign data from various sources to a single object (Merrill, Kalanithi & Maes 2007).

Gestalt principles can be identified in most tangible programming environments but none of the reviewed environments was designed explicitly with these principles is mind. Of the environments reviewed, most only incorporate one Gestalt principle. Only Story Room and Diorama table combine more than one principle with these being grouping by proximity and good continuation. My research explicitly consideres the relationships between objects and I use Gestalt principles to explain the relationships. Consequently, my work differs from others in that I make explicit reference to Gestalt principles to describe my programming environment.

1.11.4 Tangible objects and computing systems

As far as it concerns using physical objects to represent data or manipulate data, Ishii (2009) developed a model that both illustrates tangible objects as being distinct to digital data and also how one can represent the other in the digital and physical domains, respectively. According to the model, data can have both digital and physical representations. This is similar to a computer program that can take form as text on paper and also be a series of 1's and 0's in a computer's electronic store. Ishii's model therefore proposes that a dataset's representation can vary according to the domain in which it is applied. My research takes Ishii’s model one step further by demonstrating that a dataset’s physical representation can vary according to the individual. I also identified that Ishii’s model does not address the design of the physical representation. In contrast to Ishii’s model, my model highlights the origin of the object’s design.

Machine readable identification markings can help establish a link between the data and the object (Ullmer 2002). My literature analysis of how data and their representations are linked concluded that the identification may be classified as either electrically active or passive whereas the identification information can be transferred between the digital and physical worlds using mechanisms that are either tethered or untethered. I captured these options as a two dimensional model.

A common design approach is to develop functional objects without concern what meaning a user may attach to the object. Krippendorff (1989) proposes an alternative approach in which the form of the object also conveys meaning. My research is aligned with Krippendorff’s proposal and considers how individuals may choose diverse physical representations for data.

1.11.5 Tangible programming

Tangible programming environments rely on supporting technologies that range from active circuitry to passive sensing mechanisms. Passive sensing offer benefits that include a wider selection of materials to choose from, reduced cost, improved robustness, and more design options (Horn, Solovey & Jacob 2008). These benefits inspired me to design artefacts based on passive sensing technology.

Some tangible programming environments like SiteView (Beckmann & Dey 2003) dictate the object placement order along with their positions. These constraints are similar to those found in textual programming environments such as those developed for the C (Kernighan et al. 1988) language. In these environments the sequence in which symbols may be placed is often fixed and prescribed. To illustrate the point, consider the scenario in which a user assigns the numerical value of 10 to a symbol named “decade”. The correct sequence to do this is “decade = 10” and not “10 = decade”. I considered this an unnecessary constraint on the user and addressed it using the Gestalt principle of grouping by proximity.

SiteView places similar constraints on the user. For example, when the user wants to indicate the three conditions ”rain”, morning”, and “Monday” in a program rule he has to place each representative objects at prescribed positions. I argue that these conditions represent distinct data types (being the weather state, time of day, and day of the week). I developed software that can make this distinction without human intervention and interpret the objects appropriately. The result is that the three conditions in this example can now equally well be placed in different sequences including the two arrangements “Monday, morning, rain” and “rain, morning, Monday”. From this, I concluded that the symbol sequence is not important; instead, it is the combination of words that should be considered. Therefore, as long as the user places the words in close proximity to each other he does not have to be concerned about their order. This is another example of the Gestalt grouping by proximity principle.

Programming language designs often include the option for the user to determine how actions and parameters are represented. Yet, with these systems, the user has to base his designs on language elements such as parameters and actions previously determined by the language designer. For example, in the C (Kernighan et al. 1988) language, the user can choose to use the sign ”decade” to represent the quantity 10 and does this using the symbol sequence “#define decade 10”. Some tangible programming environments also include this feature (along with its limitation) and examples are Story Room (Alborzi et al. 2000), Quilt Snaps (Buechley n.d.), and Diorama Table (Oizumi et al. 2007). In Story Rooms, the user is constrained to using the designer’s physical signs but has the choice to combine these with personally meaningful objects. With Quilt Snaps, the user can embellish squares yet the size and shape are determined by the system designer. Finally, when choosing his objects for the Diorama Table, the user has to keep in mind that the way the shapes will be interpreted by the system has also been predetermined by another person. It is, therefore, possible for the user to choose what programming signs look like but this must be done within constraints determined by a system designer. I considered a different approach and demonstrated a system in which the user is free to choose the physical representations of program actions and parameters.

1.12 Delineations and limitations

This study explored the design of programming environments that support personally meaningful objects positioned according to the Gestalt principle of perceptual grouping by proximity. The scope of this study was limited to the creation of stored (as opposed to interactive) programs. Due to the interpretative epistemological assumptions made in this study, the results may not be useful in programming environments in which more than one person collaborate on the same program. I discuss my epistemological assumptions in Chapter 5.

1.13 Organisation of the remainder of the thesis

The remainder of this thesis is organised as follows. Chapter 2 provides the theoretical foundations that support this thesis. Chapter 3 captures the results of a literature study on tangible objects and I discuss Tangible Programming Environments in Chapter 4. Chapter 5 (Research methodology) presents the research design and details on the Design Science Research methodology as applied in this study. Chapter 6 (Design, implementation and evaluation) describes the design cycles of the tangible programming environment developed in support of the thesis statement as presented elsewhere in this chapter. I completed five design cycles of which three were evaluated with the help of children. Design decisions are discussed and the completed designs presented. Chapter 6 also documents the field evaluation data and the results of the data analysis. I present in Chapter 7 (Research contribution) a model of tangible programming environments that includes the user, language architect, system developer, tangible objects, Gestalt principles, and Semiotic theory. Chapter 8 (Conclusion) summarises my findings and contributions and suggests how the current study can be extended.

1.14 Conclusion

My literature search revealed that only three tangible programming environments allow the user to select the objects that represent programmatic elements. It also emerged that no model exists to inform the design of such environments. This thesis reports on a study that shows how designers of tangible programming environments can incorporate both personally meaningful objects and the Gestalt principle of perceptual grouping by proximity into their designs.

My two primary contributions to the body of knowledge are as follows: First, I demonstrate a programming environment that incorporates the Gestalt principle of perceptual grouping by proximity and Semiotic theory that allows the user to select personally meaningful objects to represent program elements. Second, I present a model for this environment that highlights the relationships between the language architect, system developer, user, tangible objects, Gestalt principles, and Semiotic theory.

CHAPTER 2

THEORETICAL BACKGROUND

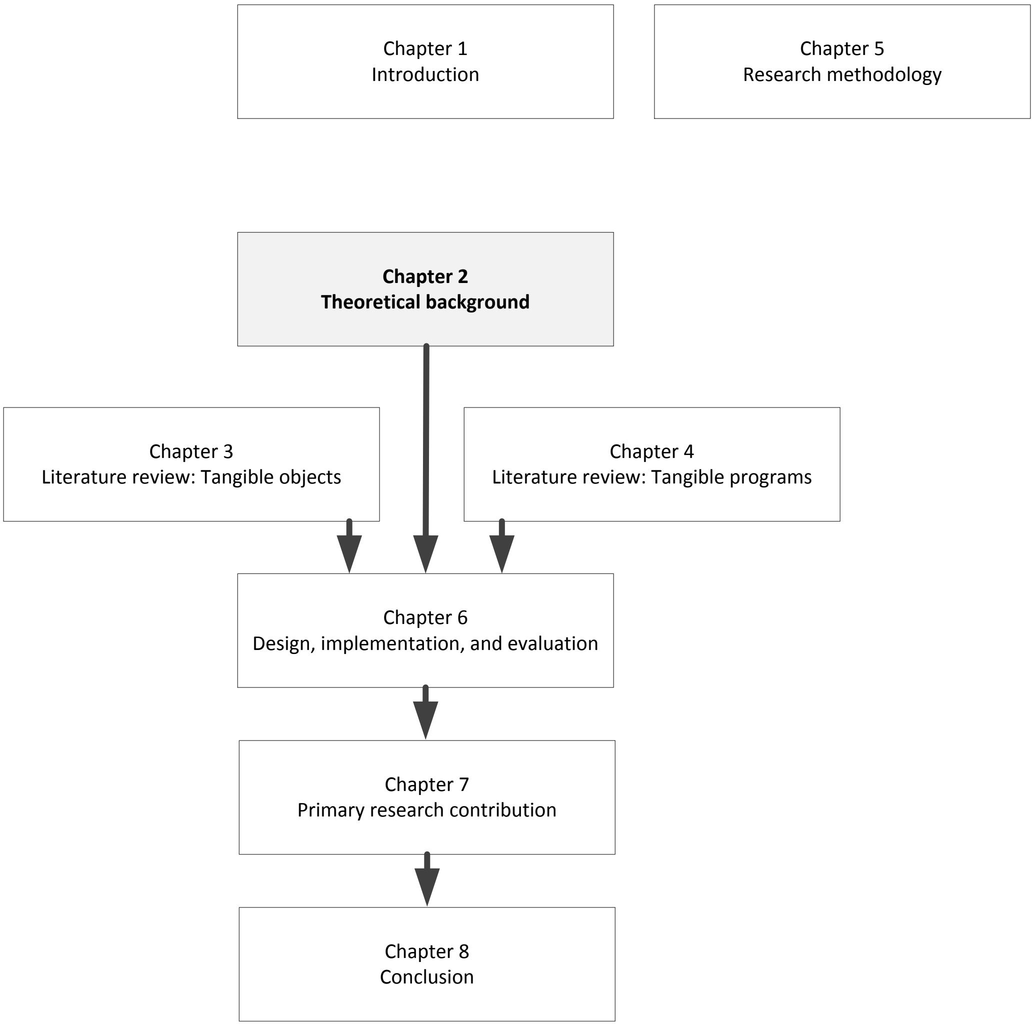

Chapter 1
Introduction
Chapter 5
Research methodology
Chapter 2
Theoretical background
Chapter 3
Literature review: Tangible objects
Chapter 4
Literature review: Tangible programs
Chapter 6
Design, implementation, and evaluation
Chapter 7
Primary research contribution
Chapter 8
Conclusion

Figure 2-1 Document structure

2.1 Introduction

In this chapter, I provide a theoretical framework to this study based on visual semiotics and the Gestalt principle of perceptual grouping. To recap, this study develops a model to guide the creation of tangible programming environments in which an individual can arrange objects in such a manner that, when the objects are combined, a product emerges that represents to its creator something more than the sum of the individual objects and at the same time constitutes a computer program. The model will combine certain Gestalt principles of perceptual organisation and Semiotic theory with programming. This chapter is guided by the scoping of Chapter 3 (Literature study: Tangible objects), Chapter 4 (Literature study: Tangible programs) and Chapter 5 (Research methodology). Figure 2-2 illustrates the hierarchical relationships that exist between the four main sections in this chapter.

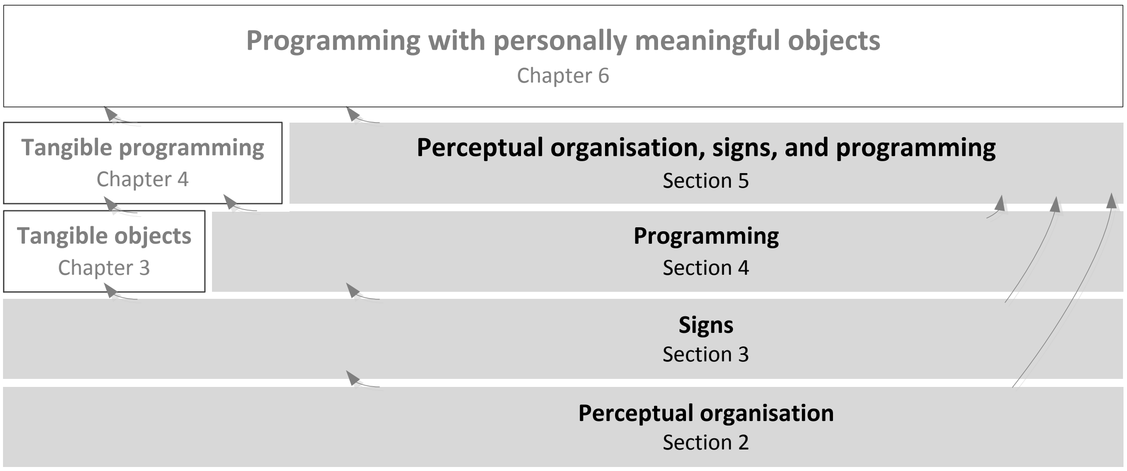


Figure 2-2 Relevant research domains and their relationships

The three research domains of perception, signs, and programming are discussed in this chapter for the following reasons: Perceptual organisation (Section 2.2) explains how individuals interpret a collection of elements, the section on signs (Section 2.3) considers how meaning is assigned to elements, while programming (Section 2.4) is discussed because the goal of this research is to derive a tangible program model. Section 2.5 illustrates the close relationships that exist between perception, signs, and programming. I conclude this chapter with Section 2.6.

2.2 Perceptual organisation

A group of early twentieth-century German psychologists developed the Gestalt principles in an effort to explain why stimulated discrete sensory receptors sometimes result in strong connected perceptions (Bregman 1994). According to Henle (1985), Wertheimer's (1912) publication on the perception of motion is the official onset of the Gestalt movement. Although Steinman, Pizlo and Pizlo (2000) and others cite Wertheimer as the founder of the Gestalt School of Psychology, Shepard and Levitin (2002) consider Wertheimer to be only one of three principal founders of this school. According to Blackburn (1996) and Gordon (2004), the other two principles founders of the Gestalt School of Psychology are Koffka and Köhler. The Gestalt theory of perception did not receive much attention from the research community after the demise of some of the original conceptualisers in the early 1940's; instead, the theory attracted negative critique in the 1950's and 1960's (Blackburn 1996). However, Blackburn states that the research community did continue to support the particular Gestalt theory proposal that states that higher-level cognitive processes (such as remembering and interpretations) are responsible for the way we experience the world. Blackburn's statement is corroborated by Matlin (1988) who emphatically puts it that psychologists with an interest in perception are still supportive of Gestalt psychology. According to Blackburn, the philosophical importance of the Gestalt theory lies in revealing the complexity of how we perceive three-dimensional spatial objects. The late 1970's and early 1980's witnessed a revived interest in Gestalt and its influence towards the understanding of visual perception (Wagemans, Elder, Kubovy, Palmer, Peterson, Singh & Heydt 2012). Henle (1989) confirms the renewed interest in the 1980's by Gestalt psychologists. This renewed interest in Gestalt is evidenced by a number of articles that make direct reference to the Gestalt theory of perception, for example, Tonder and Lyons' (2005) study on visual perception in Japanese design, Leyton's (2006) study on the structure of paintings, Desolneux, Moisan and Morel's (2008) study on computer image analysis, and Overvliet, Krampe and Wagemans' (2012) study on perceptual grouping in haptics.

The meaning that results from the perception process is due not only to individual sensations (Kasschau 2003; Wagemans et al. 2012) and their configuration (Palmer 1980), but is also influenced by the individual's knowledge, experience, assumptions, and understanding of the world (Bernstein & Nash 2008; Gregory & Zangwill 1987). The Gestalt theory of perception holds that sensations are not perceived in isolation but are assembled by the human brain to result in something that is different to the individual sensations, nor is the result (as assembled by the brain) the sum of these sensations; instead the result is a different perceptual experience called a *Gestalt* (Kasschau 2003). To illustrate this perceptual experience, Figure 2-3 (a) presents a stack of coloured pencils that can be perceived as a butterfly (Figure 2-3b) when viewed from an appropriate angle.

Although the German word "Gestalt" can be translated as *pattern* or *shape* in the English language, Rock and Palmer (1990) suggest that the word *configuration* better describes the intended meaning. In addition to the word Gestalt, Gordon (2004) also uses the word *wholes* to describe the result of the spontaneous tendency for humans to organize sensations.

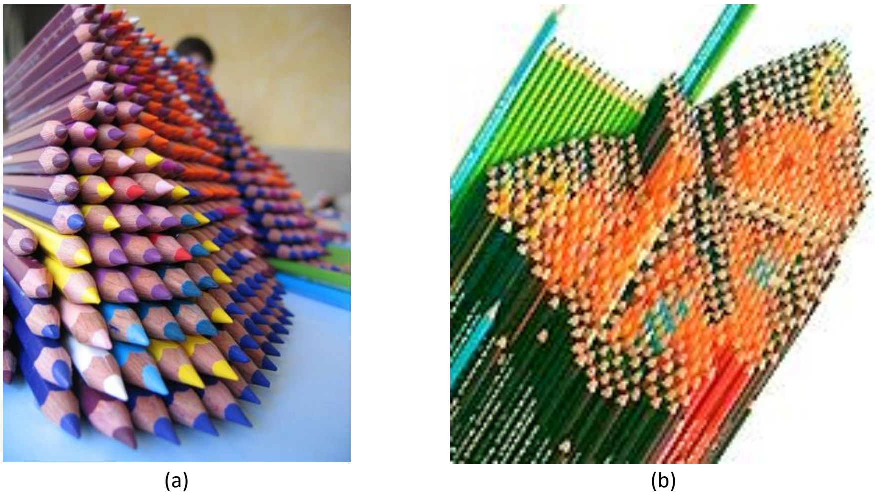


Figure 2-3 An arrangement of coloured pencils is perceived as a butterfly when viewed from a particular angle

Chandler (2007) puts it that the Gestalt theory of perception supports the notion that reality, as perceived by an individual, is not objective but is instead constructed by his perception process. Asch (2002) adds that Gestalt psychologists argued that the experience produced by complex patterns cannot always be predicted by considering their parts in isolation. According to Asch, the relations that exist between patterns are essential when considering the resultant experience.

In this study, I consider the above notion that reality (as perceived by the individual) is constructed by the individual's perception process. I further consider, as stated by Asch above, the relations that exist between patterns due to the optical observation of tangible objects (as opposed by patterns that form due to auditory inputs), and specifically those tangible objects that are in close proximity to each other. The relations that exist due to tangible objects in close proximity to each other are discussed in Section 2.2.6.1.

According to Leymarie (2006), Gestalt psychologists noted that humans sometimes perceive visual relationships in situations where no optical relationships exist and at other times visual patterns are perceived where no optical patterns exist. Here, "optical" refers to the phenomenon of light rays falling in on the optical sensors in the eye, and "visual" refers to the physiological processing of the electrical signals that result from the light rays falling in on the optical sensors in the eye. In addition to a Gestalt that may result from visual processes, Köhler (1938) states that a Gestalt may also result from the summation of spatial, auditory, and intellectual processes. Wagemans et al. (2012) corroborate Köhler's statement and adds that Gestalt principles are also applicable to virtually all perceptual experiences, including auditory (Bregman 1994) and tactile perception, amongst others.

Textual, graphical, and (as will be shown in Chapter 4), tangible programming environments rely on visual elements to represent the program. The balance of this section therefore focusses on Gestalts that result from visual processes.

2.2.1 The general Gestalt principles of perceptual organisation

Matlin (1988) comments that researchers do not agree on the number of Gestalt laws, with this number ranging between one and 114. Gestalt laws are also referred to as *Gestalt heuristics, Gestalt principles* (Goldstein 2010) and *Wertheimer's Laws*. Goldstein (2010) motivates that the Gestalt principles should not be referred to as laws because the predictions they make are not sufficiently 'strong'. Instead, Goldstein proposes that Gestalt principles should be referred to as heuristics because the Gestalt principles provide rules that work most of the time but not all the time. Palmer (1999), too, prefers to use terms such as *principles* and *factors* when referring to Wertheimer's Laws. Palmer defends his choice of terminology based on the idea that the laws described by Wertheimer are weaker than those usually considered to be scientific laws. Finally, Verstegen (2005) refers to Wertheimer's Laws as *general Gestalt principles of organization*. Based on Goldstein's motivation, Palmer's preference, and Verstegen's terminology, I will refer to Wertheimer's Laws as principles.

In this thesis I discuss 12 Gestalt principles with these being the principle of Prägnanz, the principle of figure-ground perception, the seven classical principles of perceptual grouping, and three additional classifications of perceptual grouping that have been proposed towards the end of the 1990's. The seven classical principles of perceptual grouping are proximity, similarity, common fate, symmetry, parallelism, continuity, and closure. The three additional perceptual grouping classifications are enclosure/common region, connectedness, and synchrony. Figure 2-4 illustrates examples of the nine general Gestalt principles of perceptual organisation in addition to examples of the three additional classifications of perceptual grouping.

Shepard and Levitin (2002) state that the Gestalt principles of *perceptual grouping by proximity*, *similarity*, *symmetry*, *good continuation*, and *common fate* are innate to humans and need not be learned. Figure 2-4 illustrates examples of the Gestalt principles of perceptual grouping by proximity, similarity, symmetry, good continuation, and common fate. I now give an overview of Gestalts in general and focus on the Gestalt principles of parts and wholes, *Prägnanz*, figure-ground separation, and perceptual grouping.

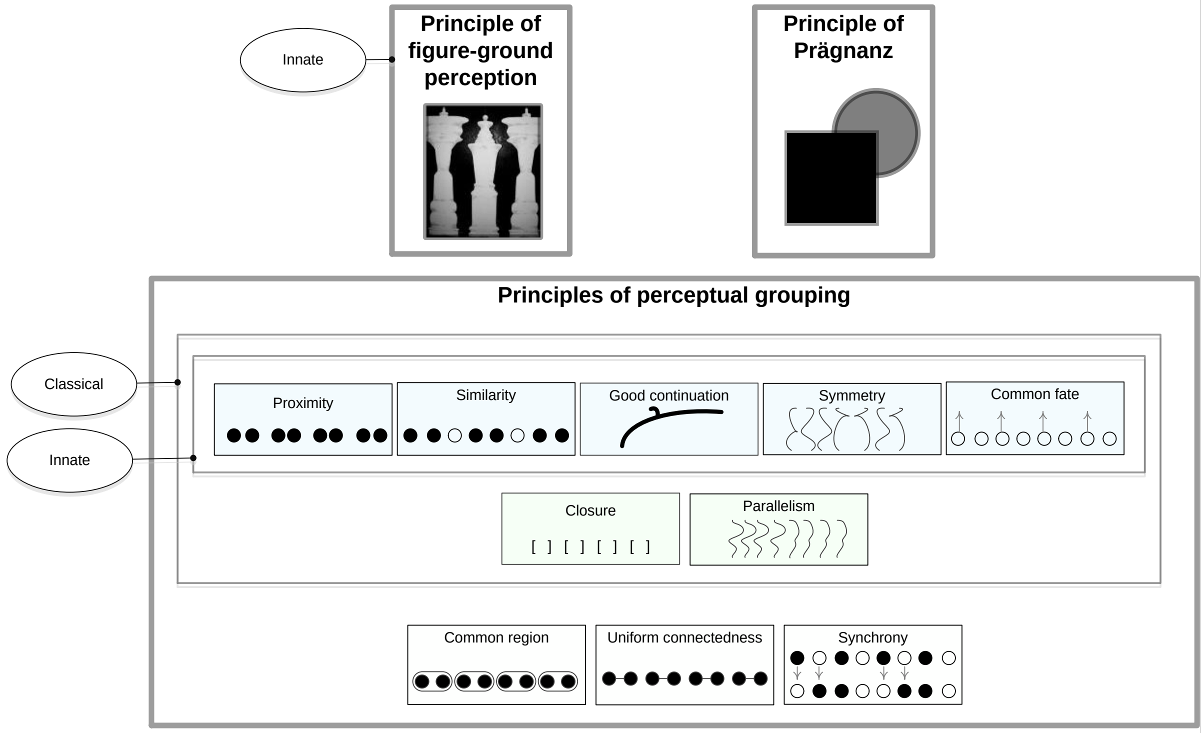


Figure 2-4 The general Gestalt principles of perceptual organisation, with examples

Based on Palmer (1999, 2002a, 2003) and Rock and Palmer (1990)

2.2.2 Gestalts

Physiologically, the human eye receives light to form an optical image on the retina at the back of the eye (Young, Freedman & Ford 2007). The way we perceive the visual world is not the same as the collection of various light intensities and colours projected onto the retina (Kimchi, Behrmann & Olson 2003). Kimchi, Behrmann and Olson (2003) add that we instead perceive the world as consisting of individual objects that are logically located in space. They therefore argue that an internal process is responsible for transforming the collection of various light intensities and colours into objects that are perceived to be related to each other. Shepard and Levitin (2002) give an example of how the optical image received from the world is not the same as the way we perceive the world. In their example (Figure 2-5), a parallelogram is copied, rotated, and then placed next to the original parallelogram. Table legs were then added to the parallelograms to result in a completed drawing.

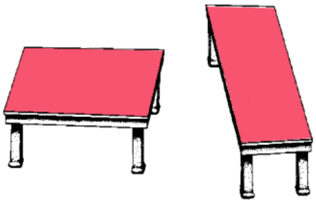


Figure 2-5 Juxtaposed parallelograms appear to differ in size

Based on Shepard and Levitin (2002)

When viewed, two tables of differing sizes and orientation are perceived. As explained by Shepard and Levitin, the pattern formed on the retina consist only of a series of lines and dots (I colour to highlight the parallelogram), yet the representation in the brain is that of two tables that differ in size.

According to Kimchi et al. (2003), the Gestalt school of thought attributes perceptual organization to processes that group and segregate the stimuli. Blackburn (1996) states that the Gestalt quality is separate from the individual sensations. From the preceding, I deduce that perception is not simply the result of physiological mechanisms (these being electrical pulses sent to the brain), but that some mechanism inside the brain influences the experience. Indeed, according to Helm (2014), a neuro-cognitive process known as perceptual organisation enables humans to interpret an optical image on the retina as objects that are arranged in physical space. Helm continues that we interpret some of these objects as being part of structured wholes. In his discussion on perceptual organisation, Palmer (2002b) excludes Helm's cognitive component of the process; Palmer instead ascribes this perceptual organisation to an activity of the visual nervous system. In my study, it is not important whether the perceptual organisation is due to the visual nervous system as described by Palmer, or Helm's neuro-cognitive process. What is important is that some mechanism does exist by which Gestalts are formed when light enters the retina but I will not investigate the mechanism.

2.2.3 Parts and wholes

When an individual interprets an experience the experience itself changes (Blackburn 1996). Gordon (2004) offers a set of two dimensional graphic lines as an example of a visual experience that is changed when it is interpreted. In his example of Figure 2-6, the experience comprises a combination of circles and straight lines. Considered individually, no interaction seems to exist between these entities and they simply represent themselves (the *parts*).

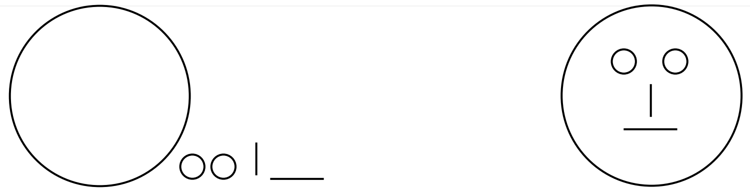


Figure 2-6 Parts and wholes

(Gordon 2004)

However, when these are assembled in a specific manner they are no longer experienced as individual sensations but are instead experienced as part of a *whole* (a pictorial representation of a face in this example). In addition, the circles and lines no longer represent themselves; they now acquire new meaning (a whole) in that they now represent two eyes, a nose, and a mouth. Another interesting example of how the experience changes can be found in Saussure's (2011) discussion on how we read the written word. He highlights that we read a word that we have not previously encountered by spelling out each letter in sequence, but once the same word has become common to the individual the group of letters are no longer experienced individually but rather as a single symbol.

Yet another illustration of how the combined individual parts results in something more than sum is Saussure's (2011) reference to the game of chess. Chess is a board game in which two opponents take turns in positioning objects on the board (Mason 1946). The significance (Saussure(2011) calls it *value*) of a chess piece is determined by its placement relative to other pieces. As an example of changing significance according to position, consider the pawn piece: the pawn is perceived as having considerable significance when positioned immediately and diagonally in front of the opposing king chess piece. In contrast, when the same piece is positioned behind the opposing king, the pawn is perceived as having little significance. Its relation to other pieces therefore determines the significance of a chess piece.

According to Gordon (2004), Wertheimer applied the *phi* phenomenon to demonstrate the part-whole interaction (the phi phenomenon is the perceived - but unpredictable - interaction between two light sources determined by the spatial and temporal relationship between the sources.) By varying the switching times of the two light sources, Wertheimer created a perceived movement located in the space between the two light sources with the 'parts' being the two light sources and the 'whole' being the combined flashing lights and perceived movement.

2.2.4 Gestalt principle of Prägnanz

The principle of Prägnanz was formulated by Koffka (1935). This principle is also referred to as the *law of Prägnanz* (see for example Helm (2014) and Rock & Palmer (1990)), *goodness* (Gordon 2004), *pattern goodness* (Kubovy, Holcombe & Wagemans 1998), and the *minimum principle* (Palmer 1999). As applied to perceptual organisation, the principle of Prägnanz holds that when the human visual system is stimulated, the visual system tends to settle into states that reflect the stimulation as a combination of the simplest and most stable shapes (Helm 2014) that are based on sensory information received from the retina (Rock & Palmer 1990). According to Helm, the principle of Prägnanz was inspired by a principle in physics that holds that dynamic systems tend to adjust in order to minimise the energy in the system. Indeed, Tyler (2002) reports that it is assumed that the human visual system shows preference for interpretations that are the simplest of all alternative interpretations. Rock and Palmer (1990) describe Prägnanz as being, when faced with an ambiguous shape, the tendency for observers to perceive the most basic organization of shapes. Palmer (1999) presents Figure 2-7 as an example. In this figure, the circle is most often perceived to be behind the square. This perception is explained by the Gestalt principle of Prägnanz as follows: When one considers the likely alternatives (a closed circle versus a three-quarter circle), the closed circle is the simplest alternative since it is more regular and symmetrical when compared to the three-quarter circle. Put another way, the closed circle is perceived as a more basic shape when compared to the three-quarter circle. Gordon (2004) suggests yet another description of Prägnanz: According to Gordon, Prägnanz implies that the retinal image is processed so as to remove details that may be distracting. Gordon describes the principle of Prägnanz as a process that is inclined to simplify the perception by maximising the resultant symmetry and wholeness.

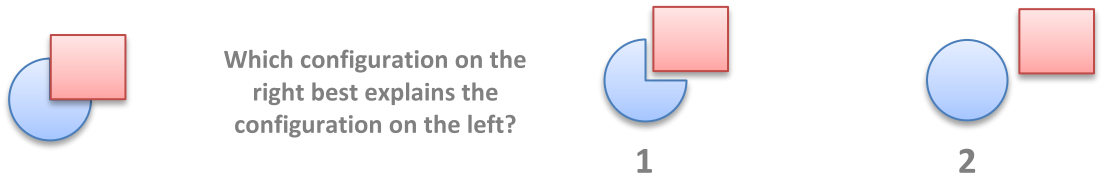

Which configuration on the right best explains the configuration on the left?
1
2

Figure 2-7 Perceiving a simple image

Based on Palmer (1999)

2.2.5 Gestalt principle of figure-ground separation

Chandler (2007) states that individuals tend to separate what they perceive to be a *figure* from what they perceive to be the *background.* He calls the processes *foregrounding* and *backgrounding* respectively and the combination of these two processes *figure-ground separation*. According to Chandler, individuals seem to identify a dominant shape from the background when presented with a visual image. This dominant shape is called a figure and it often has a definite outline that can be discerned from the rest of the image, called the *ground* (Chandler 2007). Although the Gestalt view is that figure-ground perception is innate (Wagemans et al. 2012), Matlin (1988) comments that it is prone to influence from hints and suggestions. Figure-ground separation is not limited to visual experiences but is also applicable to inputs from other sensors such as auditory input from the ears and touch sensations from the skin. Gordon (2004) provides two examples of figure-ground separation in auditory and touch sensing modalities respectively as being an individual's ability to extract the voice of a single speaker amidst multiple voices at a noisy cocktail function, and to perceive an insect crawling on the arm. He also states that all human sensory modalities are subject to figure-ground separation. Chandler (2007) puts it that we tend to perceive smaller areas (that have been placed against a larger background) as figures against the ground. He refers to this as the *principle of smallness*. Finally, Chandler refers to the *principle of symmetry* as being an individual's tendency to perceive a symmetrical area as a figure when this is observed against asymmetrical backgrounds. The principle of symmetry is also discussed in Section 2.2.6.8 in the context of the Gestalt principles of perceptual grouping.

2.2.6 Gestalt principles of perceptual grouping

According to Wagemans et al. (2012), the concept of perceptual grouping was first investigated and reported in Wertheimer's seminal work as published in 1923. An English translation (Wertheimer 1938) is also available. Wagemans et al. (2012) discuss seven principles of perceptual grouping that they refer to as the *classical principles of perceptual grouping*. These are perceptual grouping by proximity, similarity, common fate, symmetry, parallelism, continuity, and closure. Figure 2-4 illustrates examples of the classical principles of perceptual grouping.

Following an extended period of limited advances in the field of perceptual grouping after the passing away of the original Gestalt researchers, Palmer and others were the first to suggest three additional classifications of perceptual grouping (Rock & Palmer 1990). The first suggestion was what Palmer called *the law of enclosure/common region.* The second is known as the *law of connectedness* (Rock & Palmer 1990). Finally, Lee and Blake (1999) suggested a third new principle of perceptual grouping called *temporal synchrony*. Figure 2-4 illustrates the three additional classifications of perceptual grouping as suggested by Rock and Palmer (1990) and Lee and Blake (1999).

Gordon (2004) describes perceptual grouping as being spontaneous and reliable phenomena whereas Hochberg (2007) puts it that perceptual grouping follows organizational rules. Palmer (1999) states that the visual system mixes multiple grouping factors and it is therefore not easy to predict which perceptual grouping will result when an individual is exposed to a particular scenario; rather, the perceived grouping depends on the scenario. To illustrate the problem of predicting perceptual grouping, Palmer and Rock (1994) present a scenario comprised of coloured dots uniformly dispersed along a line while their colour and spacing are adjusted. In this example, the dots are perceived to be grouped by proximity when the distance between the dots is large and the difference in colour small. Conversely, when the difference in colour is increased and the difference in spacing is almost uniform, the perceived grouping was by colour.

The outcome due to perceptual grouping may vary (Palmer 1999). For example, at times the perceptual grouping may appear to be a confederation of objects in which the confederation appears not to be a strong one and even though the elements in the group are interrelated, they maintain a level of perceptual independence. Palmer (1999) calls this *element aggregation*. Perceptual grouping operations that often produce element aggregations are proximity, similarity, common region, and certain common fate operations. In addition to the phenomenon of element aggregation, Palmer (1999) states that *unit formation* is an alternative phenomenon due to perceptual grouping. Unit formation means that multiple elements could be perceived as a single object. Perceptual grouping operations that may result in unit formation are element connectedness, good continuation, and certain common fate operations.

The perceptual grouping of line-like elements is influenced by factors such as symmetry, parallelism, good continuation, and closure. In general, predicting the outcome due to grouping is not a trivial exercise. However, when only a single grouping factor is relevant in a scenario, the outcome of grouping can indeed be predicted with certainty (Palmer 1999). In the following discussion on perceptual grouping, the assumption is that only a single grouping factor is relevant and where I do compare perceptual grouping, the assumption is that everything else is equal. The strength of grouping is not constant and can be influenced by various factors. One factor is the angle through which elements are rotated while others remain unchanged. Palmer offers a set of L-shaped elements in Figure 2-8 to illustrate the point with the perceived grouping being the strongest in the second row. I next describe Wertheimer's seven classical principles of perceptual grouping and three recently proposed principles.

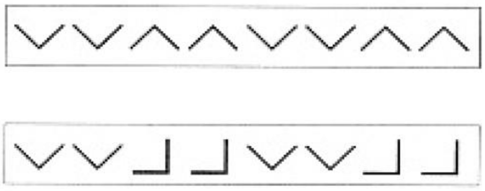


Figure 2-8 Grouping perception varies according to rotation angle

(Palmer 1999)

2.2.6.1 The principle of perceptual grouping by proximity

Even though Wertheimer (1938) used the terminology *factor-of-proximity* to describe perceptual grouping by proximity, I shall instead use Wagemans et al.'s (2012) contemporary phrase *perceptual grouping by proximity*. Palmer (1999) puts it that *relative closeness* describes the intended meaning well. Figure 2-4 applies dots to illustrate perceptual grouping by proximity. According to Kubovy et al. (1998), grouping by proximity is the most fundamental form of perceptual grouping. Wagemans et al. (2012) report that it is possible to measure the strength of perceptual grouping by proximity without the measurement being influenced by another form of perceptual grouping. Bregman (1994) comments on the strength of grouping by proximity by stating that the strength of the grouping tends to be inversely proportional to the distance between the visual elements; that is, the closer the elements, the stronger the tendency is to group them perceptually. Palmer (1999) argues that perceptual grouping by either connectedness or common region is stronger than perceptual grouping by proximity.

Kubovy et al. (1998) conducted a study to measure the strength of perceptual grouping by proximity and to determine the effect that the element configuration has on the strength of this grouping. Their study required an instrument in which the proximity and configuration of the stimuli could be varied independently. Kubovy et al. chose a dot pattern for this purpose. The results revealed that the relative grouping strength between the dots could be modelled as a decaying exponential function of distance.

2.2.6.2 The principle of perceptual grouping by similarity

Wertheimer (1938) uses the terminology *factor-of-similarity* to describe perceptual grouping by similarity but I will use the more contemporary phrase (Wagemans et al. 2012) of *perceptual grouping by similarity*. According to Chandler (2007), humans associate features with each other if they look similar and consequently group them together. Wagemans et al. (2012) give examples of similarity measurements and these are colour, size, and orientation. Figure 2-4 illustrates perceptual grouping by similarity with the aid of dots. Because similarity covers multiple properties, Palmer (1999) suggests that similarity can be considered to be the general principle of grouping. Wagemans et al. (2012) suggest that common fate and proximity (two classifications of perceptual groupings that I also discuss) may be viewed as special instances of grouping by similarity. In the case of perceptual grouping by common fate, the property to be deliberated is the object velocities whereas the object positions are relevant to grouping by proximity.

2.2.6.3 The principle of perceptual grouping by good continuation

Elements arranged in such a manner that they appear to continue on each other tend to be perceptually grouped together. Even though Wertheimer (1938) uses the terminology *factor-of-good-curve* to describe perceptual grouping by good continuation, I shall use Wagemans et al.'s (2012) more modern phrase *perceptual grouping by good continuation*. Good continuation is sometimes also referred to as *continuity*, see for example Palmer (1999). Chandler (2007) states that contours that exhibit gradual direction changes are preferred over ones that are sudden. Figure 2-4 illustrates perceptual grouping by good continuation with the aid of two intersecting lines.

2.2.6.4 The principle of perceptual grouping by closure

Gordon (2004) describes perceptual grouping by closure as the tendency for individuals to perceive a completed figure even if it is not. Closure is sometimes referred to as *closedness* (Palmer 1999) while Weiten (2011) prefers the term *completeness*. He describes completeness as the tendency for individuals to group parts so that, together, they create the perception of a completed object.

Weiten adds that individuals are able to perceptually ignore gaps in a figure. Figure 2-4 illustrates perceptual grouping by closure with the aid of right and left facing square brackets. In this figure, perception tends to pair right facing brackets with those facing left. Chandler (2007) states that our interpretations favour “closed” as opposed to “open” figures. Palmer (1999) offers an alternative explanation by stating that we tend to group elements together if they form a closed region. He also gives an example illustrating that closure supersedes continuity.

2.2.6.5 The principle of perceptual grouping by common fate

Even though Wertheimer (1938) uses the terminology *factor-of-uniform destiny* to describe perceptual grouping by common fate, I shall use *perceptual grouping by common fate which is* Wagemans et al.’s (2012) more modern terminology. Shepard and Levitin (2002) state that it is very improbable that things that move in a highly correlated way in the physical world are not connected to each other in some way. Figure 2-4 illustrates perceptual grouping by common fate with the aid of alternating dots where the alternating dots are shown as moving in the same direction. Wagemans et al. (2012) state that the perceptual grouping by common fate does not seem to be limited to motion in three dimensions, but also applies to luminance changes. As an example, Wagemans et al. recall an experiment conducted by Sekuler and Bennett (2001) that points to a tendency for observers to group elements of a visual scene perceptually when the elements in the scene become brighter or darker at the same time, irrespective of the individual luminances of the elements. According to Shepard and Levitin (2002), perceptual grouping by common fate is a much stronger grouping principle than the principles of perceptual grouping by proximity, similarity, symmetry, and good continuation. If the speed and direction of element movement is considered, Palmer (1999) argues that grouping by common fate is a special case of perceptual grouping by similarity.

2.2.6.6 The principle of perceptual grouping by synchrony

Palmer (1999) puts it that perceptual grouping by synchrony is not limited to visual events but also applicable to auditory perception. According to Palmer, the phenomenon of synchrony is similar to that of common fate in that both phenomena are due to the dynamics of the observed elements. However, the phenomenon of synchrony differs from common fate in that synchrony does not require the elements to change in the same way. Another difference is that synchrony does not have to include motion; it only requires synchronised changes (Palmer 2003). To illustrate grouping by synchrony that does not involve motion, Palmer (2002a) offers two rows of dots in which the dots alternate in colour between black and white. Irrespective of the colour change in a particular dot, all dots that change at the same time are perceived as belonging together. Figure 2-4 illustrates perceptual grouping by synchrony using dots that change in colour as indicated by the arrows.

2.2.6.7 The principle of perceptual grouping by parallelism

Palmer (1999) observes that parallelism (parallel contours) in a physical object is not affected when it is moved. He explains that parallelisms between objects are primarily by coincidence and are often eliminated when the arrangement is disturbed.

2.2.6.8 The principle of perceptual grouping by symmetry

All stimuli humans receive are subject to symmetry detection and the symmetry need not be perfect to be detected (Helm 2011). According to Shepard and Levitin (2002), if objects are not related to each other then it is unlikely that there are any symmetric relationships between them. Therefore, if a symmetric relationship between objects is observed then we tend to perceptually group these objects together (Shepard & Levitin 2002).

2.2.6.9 The principle of perceptual grouping by uniform connectedness

According to Rock and Palmer (1990), the visual system tends to perceive connected uniform areas as a single unit. They call this *connectedness*. Examples of uniform areas include spots, lines, and larger areas. In 1994, Palmer and Rock published a comprehensive description of perceptual grouping by connectedness and called this *uniform connectedness*. According to this description, the initial perception of closed regions with no variation in their visual properties are at first perceived as a single unit. Visual properties include lightness, chromatic colour, motion, and texture (Goldstein 2010; Palmer & Rock 1994). Refer to Figure 2-4 for an illustration of perceptual grouping by uniform connectedness. In this figure, two spots that are joined by a horizontal line are perceived as a single unit.

2.2.6.10 The principle of perceptual grouping by common region

Palmer (1999) states that perceptual grouping by common region is the phenomenon that elements could be perceived as belonging together if they are enclosed within a common spatial region. Figure 2-4 illustrates an example of the principle of perceptual grouping by common region. In this figure, solid dots are distributed in a manner similar to those in the perceptual grouping by proximity example. However, the closely spaced dots are not perceived as belonging together and the common regions “rearrange” the perceived grouping so that it appears as if the dots in the common region belong together. This illustration demonstrates that the principle of perceptual grouping by common region supersedes the principle of perceptual grouping by proximity.

2.2.7 Conclusion to this section

Psychologists have observed that humans do not perceive sensory inputs as discrete entities but rather that perceptions are created based on multiple simultaneous inputs from various senses. Sensory inputs therefore influence how we experience the world. The way we experience the world differs from one person to the next. Experience is therefore subjective. Reality is also not universal but constructed by the individual based on sensory inputs. Gestalt describes the experience that results from multiple concurrent inputs. This experience does not equate to the sum of the inputs but instead is something different. It is also known that humans organize sensations with little or no conscience effort and the way in which we organise sensations are often predictable. Hence, we are constantly subjected to experiences based on our sensory inputs and the experiences differ between individuals even when the inputs are the same. The sensory inputs therefore hold a personal meaning for an individual. For individuals to function in a society it is necessary that its members agree to common meanings for specific sensory inputs. Road users should for example agree on road markings and programmers on program symbols. Section 2.3 considers sensory inputs and how their associated meanings are determined.

2.3 Signs

Communication in all its forms rely on signs (Hall 2007) and we interpret our world through a structure mediated and supported by signs (Deely 1990). The context in which we observe signs is also central to the way we interpret them; in addition, as the context changes so too does our interpretation of a sign (Hall 2007). Of particular relevance to my study is Andersen's (1997) view that a sign is afforded its property by virtue of the way it is treated by someone; that is, a sign has personal meaning.

A sign is any element that carries meaning; however, whatever the sign represents need not actually exist (Bopry 2002; Eco 1976). Signs can take form as the artificial, the natural, the verbal, the non-verbal, an artefact, and an act (Fiske 1990; Larsen 1994). Examples include visual text, literature, images, colours, sounds, utterances, physical objects, pictures, facial expressions, gestures, body language, odours, flavours, mathematical equations, whatever is on the stage in a theatre performance, and works of art including movie pictures and television programs (Andersen 1997; Berger 2012; Chandler 2007; Danesi 2004; Larsen 1994).

Physical objects that serve as signs are central to this study. An object on its own can be a sign or the object itself can be a combination of signs. For example, a book can be a sign in itself or it can be considered to be a collection of signs (Larsen 1994). Another example is a city that is either a sign itself or a collection of signs. Of particular interest to my study is Larsen's view that an exhibition of physical objects can be a sign. Based on this view and guided by certain Gestalt principles, in Chapter 6 I interpret an exhibition of personally meaningful objects as a collection of signs that together define a program.

2.3.1 Semiotics

Semiotics is an interdisciplinary research domain (Morris 1964) that studies sign systems (Fawcett 1992), sign processes (Krampen, Oehler, Posner, Sebeok & Uexkull 1987) and the creation of signs (Aghaei 2015); in particular, it studies the meaning that signs hold for humans. Semiotics consists of sign and semiosis components where the sign component represents something other than itself and the semiosis component describes how signs are created and used (Andersen 1997).

Almost all human artefacts incorporate some aspect of semiotics (Fawcett 1992). Because semiotics encompasses all things that are made, used, or adopted by people with the purpose of conveying meaning, the following are relevant to semiotics: artefacts, architecture, art forms, visual communication, linguistics, tone of voice, body posture, gestures, a person's attire and the vehicle he drives (Chandler 2007; Danesi 2004; Fawcett 1992).

Semiotics considers inter and intra communication involving persons, machines, and natural living things such as organisms, plants, and animals (Krampen et al. 1987). In particular, a computer is also a semiotic system (Rapaport 2012). Computer semiotics is a branch of semiotics concerned with computer-based signs and Nadin (1988) put it that the science of the computer interface is a science of semiotics. Referring to Peirce's model that I will discuss in Section 2.3.3, Nadin considered the interface to be the sign representamen, the type of computer system as the sign object, and context and the values that result to be the sign interpretant (Andersen 1997).

Of specific interest to my research is the study of how humans and machines communicate. Fawcett (1992) refers to this as computational semiotics. In Chapter 6 I develop a tangible programming environment that allows the user to communicate her intensions to the computer using personally meaningful objects.

2.3.2 Saussure and Peirce

Saussure (2011) and Peirce (1935) are the founders of modern semiotics (Chandler 2007; Danesi 2004) and even though their approaches differed, both are applicable to computer systems (Andersen 1997). Saussure's *sign* is a dyadic model consisting of a *signifier* and a *signified* (Saussure 1916, 1959). Independent of Saussure (Mick 1986), Peirce developed his triad that consists of a *representamen*, its *semiotic object*, and an *interpretant*.

Although both models describe a stimulus that leads to an accompanying meaning, Peirce's description of the mechanism by which this happens is more detailed (Cohn 2013). The added detail provides a sound theoretical foundation for my subsequent argument on the meaning personal objects hold for an individual; consequently, the discussion below focusses on Peirce's triadic model.

Even though the Saussurean model is not as comprehensive as Peirce's, it is compact which makes it useful in discussions regarding multiple representations that hold the same meaning. To this end and in Section 2.5.2, I will apply Saussure's linguistic sign model to three program language generations.

2.3.3 Peirce's model

Semiology literature refers to the word *sign* in its various incarnations (for example *SIGN*, *Sign*, and *sign*) along with their significantly diverse associations. It would therefore be prudent to clarify this word before discussing Peirce's model.

First, confusion may ensue in the use of the Saussurian *signifier*, the Peircean *representamen*, and the word *sign* (Chandler 2007). To clarify, Chandler explains that s*ign vehicle* independently refers to both the Saussurean *signifier* and the Peircean *representamen*. Second, the word *sign* can be ambiguous since it refers to the Saussurean *signifier/signified duality* while in the Peircean framework it refers to the *representamen correlate* (Chandler 2007). Therefore, it follows that in the Saussurean model *the sign vehicle* and the *sign* are distinct correlates whereas both refer to the Peircean representamen. Finally, Souza (2005) uses *SIGN* to refer to Peirce's triad and to differentiate it from the *representamen* that is sometimes also referred to as a *sign* (Section 2.3.3.1 elaborates on this). Therefore, in addition to adopting Souza's *SIGN* I will use the term *sign-representamen* when referring to the *SIGN representamen correlate*.

Having clarified the various meanings of the word sign and their associations, I next discuss the three correlates of Peirce's model. This is followed by an explanation of the *phenomenological categories of experience* and the *ontological categories of being*. Finally, these two categories support the subsequent description of Peirce's three trichotomies.

2.3.3.1 The sign-representamen

Peirce composed 76 definitions of the sign-representamen (Marty 2015). For the purpose of this study I consider the following definition to be suitable:

> "A sign, or representamen, is something which stands to somebody for something in some respect or capacity. It addresses somebody, that is, creates in the mind of that person an equivalent sign, or perhaps a more developed sign. That sign which it creates I call the interpretant of the first sign. The sign stands for something, its object. It stands for that object, not in all respects, but in reference to a sort of idea which I have sometimes called the ground of the representamen. " (Peirce 1935)

From this it is evident that Peirce considered the sign-representamen as something that represents something to somebody (Sebeok 2001). In particular, a sign-representamen is anything that brings about a triadic relation between itself, the associated object, and the associated interpretant (Lee 1997; Merrell 1997). The representamen is also referred to as a *sign vehicle* (Chandler 2007) and *representation* (Souza 2005).

Sign-representamens are not limited to physical objects (Chandler 2007) but are most often thought processes (Peirce 1935). Although anything can be a representamen (Chandler 2007) not all representamens are correlates in Peirce's (1935) model. This is because a sign-representamen is a specific type of representamen that is interpreted and results in an interpretant known as *a cognition of the mind* (Peirce 1935) or a *mental interpretant* (Lee 1997). I discuss Peirce's interpretant in Section 2.3.3.3.

With the semiotic object initially veiled and the interpretant non-existent, the representamen is the first SIGN correlate that is noticed; therefore, a sign-representamen is both the origin of a SIGN and its form (Chandler 2007; Merrell 1997). The semiotic object is subsequently revealed and the interpretant formed (Chandler 2007). At this point it becomes appropriate to refer to the representamen as a sign-representamen.

It is not in all respects that the sign-representamen stands for the semiotic object but only in the context (Sheriff 1989) in which the sign-representamen relates to the interpretant. This context is called the sign-representamen's *ground* (Peirce 1935) or the *source of meaning* (Cantor 2003). In relating the above to Saussure's signifier/signified model, Sheriff (1989) states that it is due to both the sign-representamen's quality and the sign-representamen's relations that the sign-representamen serves as a signifier.

2.3.3.2 The semiotic object

The semiotic object (also referred to as *referent* (Chandler 2007; Souza 2005)) has three attributes (Peirce 1935). First, the object may exist, or have previously existed, or be expected to exist, or be a combination of these conditions. This includes abstract ideas, physical things, and things that do not actually exist (Chandler 2007). Second, the object may be a known quality, a relation, or a fact. Finally, the object may be a collection of things or the result of parts assembled to form a single entity.

For the interpreter, the sign-representamen does not stand for the whole semiotic object but only in certain aspects (Peirce 1935) and to fully understand the semiotic object one needs to consider a collection of multiple sign representamens (Sáenz-Ludlow 2007). Peirce (1935) referred to this as a *complex object*. Also, the relation between the sign-representamen and the semiotic object is not necessarily one-to-one; instead, a single semiotic object may be associated with multiple sign-representamens and conversely a single sign-representamen may have multiple semiotic objects associated with it (Peirce 1935; Sáenz-Ludlow 2007). From this I deduce that both the sign-representamen and semiotic object may be correlates in multiple SIGNs.

2.3.3.3 The interpretant

Peirce used the term *interpreter* when referring to somebody while others use *person* (Souza 2005), *sign-user* (Sebeok 2001) and *translator* (Liszka 1996). Of particular interest is that anything or anybody that does the interpretation is called an interpreter (Liszka 1996). Liszka uses as example a bee and a person that each interprets the same flower differently. In this research I investigate the application of a computer program to interpret an arrangement of personally meaningful objects.

The meaning that results when the interpreter responds to (or perceives (Sheriff 1989)) the sign-representamen within his social, personal, and contextual environment is called the *interpretant* (Danesi 2009; Sabre 2012; Sebeok 2001) and this process is called *interpretation* (Souza 2005) or *semiosis* (Larsen 1994). The result is that the interpretant "binds" the representamen to the corresponding semiotic object (Souza 2005). Alternative terminology for the interpretant include *mental effect* (Fiske 1990), *a sign of the mind* (Sheriff 1989) and *meaning* (Nam & Kim 2010; Souza 2005).

A sign-representamen does not inherently hold meaning or transmit meaning; rather, an observer subconsciously constructs meaning and the result is influenced by the observer's cultural background (Chandler 2007; Martin & Ringham 2000). I therefore deduce that meaning varies amongst observers due to differing cultural backgrounds.

In addition to the environment, the interpreter's previous experiences with the semiotic object also determines the interpretant (Fiske 1990). Even when previous experience is not with the particular object but rather with the same object category, the interpretant can respond appropriately based on stored experiences with objects from this category (Palmer 1999). Souza (2005) explains that the meaning changes according to the individual and the available information; that is, as the information available to the interpreter changes so too does the meaning. The interpreter's interpretive process eventually stabilises on a final meaning (Souza 2005).

Also, an individual belonging to a certain culture may interpret a sign-representamen in a manner different to the general cultural interpretation (Barthes 1982). Computer programmers have their own culture (Raymond 2000) and Andersen (1997) suggests that meanings intended by these system developers are not necessarily the same meanings experienced by the users. Therefore, semiosis during system development may differ from semiosis when used. Barthes (1982) introduces the terms *stadium* and *punctum* to describe these differences, with the stadium being the cultural interpretation of the sign-representamen and the punctum being the individual's interpretation. The distinction between stadium and punctum is highlighted in Chapter 6 where I discuss my RockBlocks tangible programming environment. The above supports Liszka's (1996) comment that the interpretant is specific to the interpreter and I therefore conclude that a SIGN is *personally meaningful* to the interpreter.

2.3.3.4 Phenomenological categories of experience

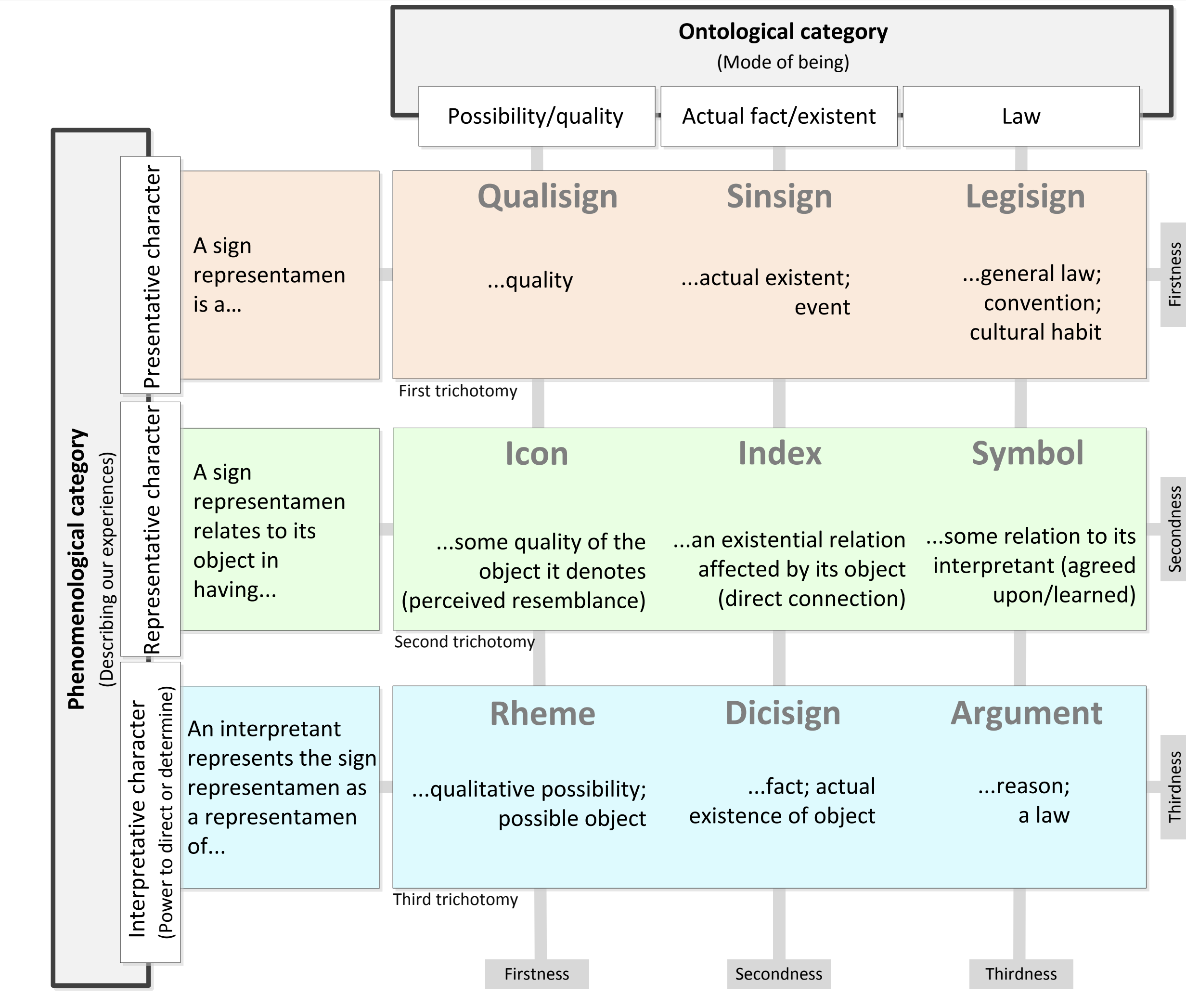
Phenomenology is the thinking style (Merleau-Ponty 1962) in which the subject is considered to be a person that lives inseparably in the world, himself, and with other persons (Farina 2014). Describing our experiences this way assumes that everything's perceived value is dependent on the person's lived experience (Merleau-Ponty 1962). In applying this thinking style to the representamen, Peirce (1935) identified three categories with these being the nature of the representamen in itself, its relation to the semiotic object, and the relation to its interpretant. These are respectively also referred to as the *presentative*, the *representative*, and *interpretative character* of the representamen (Liszka 1996). Additional categories were later added (Sheriff 1989) but Peirce's will suffice for this discussion.

2.3.3.5 Ontological categories of being

Peirce (1935) categorised the nature of being according to the way one entity relates to others. An entity can be an idea, a physical object, or a phenomenon and the three categories are *Firstness*, *Secondness*, and *Thirdness*.

For Peirce, Firstness is an entity's mode of being that is independent of another. It is simply the appearance, an idea, a quality, a sensation, a sentiment, a perception, or a "gut feeling" (Merrell 2015; Peirce 1935; Souza 2005). Secondness exhibits a property of actual existence and is a mode of being where one entity is in relation to another. Finally, Thirdness exhibits the property of a general law and is the mode of being in which an entity brings a second and a third entity into relation with each other and itself. The following example illustrates the three categories of being: The quality of pain is an example of Firstness, the association of pain with a tooth is Secondness and Thirdness is the thought that a dentist must be visited (Sarbo & Farkas 2013).

Peirce applied his three ontological categories to each of his phenomenological categories by defining three trichotomies. Figure 2-9 supports the following discussion on Peirce's trichotomies.



Phenomenological category (Describing our experiences)		Ontological category (Mode of being): Possibility/quality	Actual fact/existent	Law	
Presentative character	A sign representamen is a...	**Qualisign** ...quality	**Sinsign** ...actual existent; event	**Legisign** ...general law; convention; cultural habit	Firstness
		First trichotomy			
Representative character	A sign representamen relates to its object in having...	**Icon** ...some quality of the object it denotes (perceived resemblance)	**Index** ...an existential relation affected by its object (direct connection)	**Symbol** ...some relation to its interpretant (agreed upon/learned)	Secondness
		Second trichotomy			
Interpretative character (Power to direct or determine)	An interpretant represents the sign representamen as a representamen of...	**Rheme** ...qualitative possibility; possible object	**Dicisign** ...fact; actual existence of object	**Argument** ...reason; a law	Thirdness
		Third trichotomy			
		Firstness	Secondness	Thirdness	

Figure 2-9 Peirce's trichotomies of signs in context

Based on Merleau-Ponty (1962), Sheriff (1989) and Liszka (1996)

2.3.3.6 Peirce's first trichotomy

Peirce's first trichotomy considers the presentative character of the sign-representamen itself without considering how it relates with its semiotic object (Liszka 1996). These features can be known without requiring mental interpretation.

A sign-representamen is a *qualisign* due to a feature it possesses that is known directly without requiring cognition; for example, a red object is a qualisign simply because it is red (Liszka 1996). A *sinsign* is a sign representamen that is unique in its once-off (Jappy 2013) occurrence in time or in space; for example, a buzzer flashes and makes a sound at a particular instance of it being observed (Liszka 1996). In this example, it is not important that the buzzer is red and can make a sound but rather that it makes the sound and flashes at a particular instant in time. Finally, a sign representamen is considered to be a *ligisign* due to its predictive tendency, a developed convention, or a lawlike property bestowed upon it; for example, the colour red signals danger (Liszka 1996).

2.3.3.7 Peirce's second trichotomy

This trichotomy considers the extent to which the presentative properties of the sign-representamen correlates with the semiotic object (Liszka 1996) according to likeness, connection, and convention. Peirce named these *icons*, *indices*, and *symbols* respectively and Bopry (2002) states that they are not mutually exclusive; rather, all representamens include elements of the three classes and their application depends on the context in which the observer encounters the sign-representamen.

If some character of the sign-representamen coincides with a character of the semiotic object, it is an *icon*. For example, a photograph of Churchill is an icon of this individual. When the primary correlation between the representamen and the semiotic object is due to a reference that the representamen makes to an actual, a physical, or an imagined contiguous property of the semiotic object then the representamen is called an *index* (Liszka 1996; Merrell 1997). For example, smoke is an index of a fire. Finally, when the relation between the sign-representamen and the semiotic object is not primarily due to a presentative property of the representamen but instead due to either convention, being natural, or determined by an authority, then the representamen is called a *symbol*. The interpretation of a symbol is determined a-priori by social convention or posteriori by cultural habit and therefore the reference is by virtue of a law where law refers to social convention or cultural habit (Everaert-Desmedt 2011). The wagging tail of a dog is an example of a natural symbol of friendliness (Liszka 1996).

Unlike the index where a natural link exists between the representamen and its semiotic object and interpretant, and unlike the icon where there exists similarity or resemblance, only a cognitive link exists between a symbol, its semiotic object, and interpretant. Merrell (1997) explains that due to the socially attributed relation between the symbol, its semiotic object, and the interpretant communication within the social group will be functional but communication outside the group could be problematic. Chapter 6 discusses an encounter with this problem when I evaluated my GameBlocks tangible programming environment.

2.3.3.8 Peirce's third trichotomy

The third trichotomy considers the power of the sign-representamen to guide the formation of the interpretant (Liszka 1996). When the qualitative properties (and not the existential or lawlike qualities) of a sign-representamen tend to dominate the focus of the interpretant then the sign-representamen is called a *rheme* (Liszka 1996). A rheme brings an image to the interpreter's mind that consists of generic characteristics of the semiotic object and yet no specifics of the semiotic object (Peirce 1935). According to Liszka, the term *human being* is an example of a rheme that produces an interpretant that brings to mind some general properties of the semiotic object but does not refer to a specific instance of the object.

Peirce (1935) described a *decisign* as a unambiguous instance of a rheme. To clarify, whereas the rheme refers to a generic class of semiotic object, a decisign brings to the interpreter's mind a specific object that actually exists; that is, an existential object. All sign-representamens in Chapter 6 are decisigns.

The sign-representamen as *argument* has a lawlike effect on the interpreter to apply one of three mental argumentative processes that Peirce referred to as *deductive*, *inductive*, and *abduction* arguments. Peirce' deductive argument is the making explicit of something that already exists in a collection of connected sign-representamens but nonetheless remains unnoticed (Liszka 1996). For example if 'S' is 'M' and 'M' is 'P', then 'S' is 'P' (Peirce 1935). Liszka explains that both the induction and the abduction arguments add to the information that is already present in a SIGN system; specifically, the abduction argument introduces new hypotheses based on unexpected events in a system of SIGNs whereas the induction argument produces a conclusion that can be drawn from observed results. Finally, the result of the abduction argument is a theory that can explain the observation.

2.3.4 Conclusion to this section

Individuals can attach different meanings to the same perceived inputs. To survive in a community, individuals must communicate and in order to do so, the parties must agree on how meanings should be represented. The combination of representation and the associated meaning is referred to as a SIGN. The representation can be categorised according to the mechanism by which meaning is associated with it. For example, the icon is a representation that requires little cognitive effort to recall the associated meaning. In contrast to the icon, the symbol is a representation that is agreed on by society and typically requires significant cognitive effort when the associated meaning is recalled. SIGNs are not only used when individuals and societies communicate but also when individuals communicate with computers. Section 2.4 considers how SIGNs are applied when programming.

2.4 Programming

A computer is a reconfigurable tool that can serve diverse purposes (Kay 1984; Morgado 2006) with its behaviour determined by the program being executed (Newell, Shaw & Simon 1958). Programming is both a specialist and non-specialist human activity that changes the functionality of a system (Roy & Haridi 2004). A program consists of language symbols arranged according to predetermined rules and together they describe the algorithm that will be interpreted (Brookshear 2012; Kelleher & Pausch 2005; Ko, Abraham, Beckwith, Blackwell, Burnett, Erwig, Scaffidi, Lawrance, Lieberman, Myers, Rosson, Rothermel, Shaw & Wiedenbeck 2011; Morgado 2006; Touretzky 1984).

A language is a means to express intent (Bentley 1986) and a programming language can take several forms including text, spatial movements, and temporal events (Bentley 1986). In particular, a programming language is a communication system (Tanaka-Ishii 2010) with which a user expresses intent (Fernaeus 2007; Morgado 2006) in the form of a program. In Chapter 6, I explore the application of personally meaningful tangible objects in this communication system.

The program instructions that a programmer passes to a computer is the start of a process consisting of multiple steps that are often not visible to the programmer. This is because the programmer uses an environment that shields him from the details of how the program is converted into a form suitable for execution. The mechanism that accomplishes this is called *abstraction*. Abstraction helps the programmer by presenting a model that suits the programmer's operating level, thereby separating the programmer from lower-level details (Philipose, Fishkin, Perkowitz, Patterson, Fox, Kautz & Hahnel 2004).

A program is a collection of instructions that are to be processed by the central processing unit (CPU) and the program is often in text format. When textual, a program is an arrangement of well-selected alphanumeric characters. At the time when a programmer composes a computer program, he relies on a design by which the programmer and the computer have settled on both the meaning of each symbol as well as the rules by which the symbols may be combined. My T-Logo programming environment in Chapter 6 provides mechanisms to make each symbol personally meaningful and to identify where prescribed Gestalt-based rules have been applied.

2.4.1 Psychology of programming

Computer programming involves significant abstract thinking effort because it requires abstraction at three levels. These are the programming language, the computer program, and the algorithm (Katai, Juhász & Adorjáni 2008). Blackwell (2002) lists three cognitive features of programming: First, the programmer loses direct manipulation because programming abstracts situations, entities, and time. Second, the user applies abstraction as a means to express common features of system behaviour. Finally, notational elements represent abstract program concepts. I discussed notational elements in Section 2.3.

In the human brain, each of the two cerebral hemispheres tends to facilitate different functions (Eysenck & Keane 2000; Hugdahl & J.Davidson 2003; Kimura 1993; Kosslyn 1996). For right-handed persons the left hemisphere tends to be analytical whereas the right side tends towards the perceptual and intuitive (Franks 2006; Groome, Dewart, Esgate, Kemp, Towell & Gurney 1999; Katai et al. 2008). In order to reduce the significant cognitive load that abstract thinking places on the programmer's working memory, Katai et al. (2008) proposes a multiple-senses approach to programming called *dual coding*. Dual coding includes text, images, and sensations in the coding activity to balance the cognitive load between the two brain hemispheres (Katai et al. 2008).

The Psychology of Programming research domain focusses on the cognition of the programmer (Sajaniemi 2008) and it identified that individual computer programmers have unique cognitive styles (Weinberg 1998). Blackwell (2006) observed that usability of both the programming language and the programming environment are relevant to professional and end-user programmers alike. Programming is a creative process (Dollery 2003) that includes a wide range of cognitive resources and multiple programming environments have consequently been engineered to support varying cognitive styles (Blackwell 2006). Based on Sajaniemi, Weinberg, Blackwell, and Dollery's observations, my research on tangible programming environments aims to address individual cognition styles by providing a mechanism through which the user can incorporate personally meaningful elements. Research indicates that certain parts of the brain process certain information better than do other parts; in particular, the left hemisphere is better at analytical processing and the right is better suited to spatial information (Banich & Heller 1998). At the onset of this study, my programming elements relied almost exclusively on the "analytical" left hemisphere. These evolved over numerous design iterations to the point where the artefact design accommodates the user's creativity, thereby also incorporating the "perceptual" and "intuitive" right hemisphere. Consequently, my T-Logo incorporates dual coding and the user determines the distribution of the programming cognitive load to suit his own style. The design evaluation discussion in Chapter 6 begins with GameBlocks and concludes with the T-Logo programming environment.

2.4.2 Classification of programmable systems

In contrast to general-purpose languages (GPL's) such as C and Java that are not optimised for a particular application domain, Domain Specific Languages (DSL's) address specific domains including domestic appliance programming (Bentley 1986; Mernik, Heering & Sloane 2005). Even though DSL's are not as feature rich as GPL's and therefore not as universally applicable, a DSL offers a number of benefits (Mernik et al. 2005). For example, it simplifies the programming activity in the associated application domain, provides greater domain specific expressiveness and it makes the applicable domain more accessible to users. The FORTRAN and COBOL programming languages are examples of DSL's designed for scientific and business computing, respectively (Bentley 1986). More recent examples are the statistical computation language R (Gardener 2012), Processing (Greenberg 2007) for data visualisation, and the Hyper Text Markup Language (HTML) (Aronson 2011) and TeX (Syropoulos, Tsolomitis & Sofroniou. 2003) for visual effect.

Programming has diversified from its original focus of solving mathematical problems to include programmable domestic appliances (Blackwell 2002), each with their own DSL. Blackwell (2002) puts it that, in general, the programming actions of the user who fully exploits these devices are similar to those of a professional programmer. He explains that both face challenges that include deriving the correct program specification, using notations, and the risk of the program not executing as expected due to programming errors. Also, both share abstraction over time and class with abstraction over time referring to the programmer configuring an action to take place in the future and abstraction over a class is the ability of a programmable system to address multiple entities simultaneously (Blackwell & Hague 2001a).

Although somewhat dated, Nardi's (1993) classification remains a useful instrument to plot programmable systems according to their expressiveness level of interactive construction. Although Nardi does not define "expressiveness" and "interactive construction", I deduce from Patwell's (1992) descriptions that expressiveness is a measure of how effectively an idea can be conveyed whereas interactive construction refers to how direct and continual the system's response is when a user interacts with it. I have adapted Nardi's classification representation Figure 2-10 to include tangible programming environments and programmable domestic appliances. Tangible programming environments are discussed in Chapter 4. I highlight programmable domestic appliances since that application domain will benefit when personally meaningful objects are part of the programming activity. The position of my T-logo tangible programming environment (discussed in Chapter 6) on this graph reflects the user's ability to express his individuality by using personally meaningful objects as program elements.

2.4.3 Generations of languages

Certain languages separate the programmer from the central processing unit's (CPU) peculiarities. The degree to which languages address this separation can be plotted on a discrete linear scale (Brookshear 2012) where the lower end of this scale are first-generation languages and the upper end are higher order generations. Brookshear explains that when a programmer applies a first-generation programming language he must conform to the computer's characteristics whereas higher generations progressively separate him from CPU constraints.

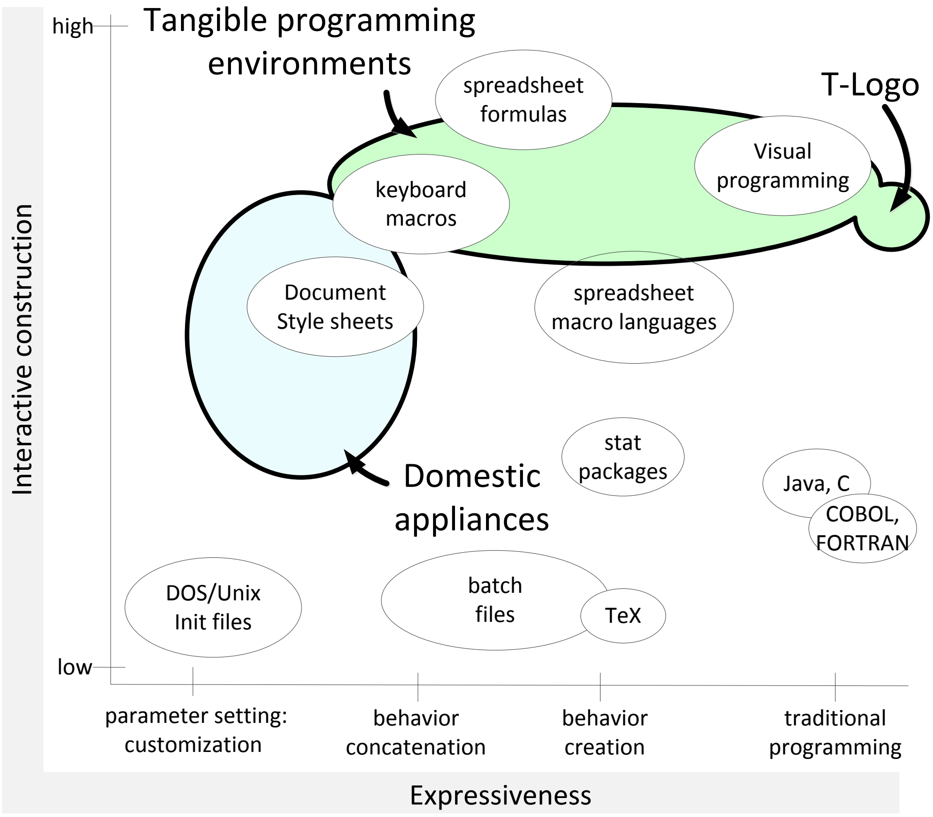


Figure 2-10 Programmable systems

Based on Nardi (1993)

While first and second-generation programming languages consider the CPU to be a collection of unchangeable elements (data pathways and data registers are examples), third-generation programming languages view it as an abstract device of which the detailed operation and elements can be disregarded. My research approach shares the latter view. Therefore, the programmer does not have to burden himself with the CPU's specific characteristics. Although Brookshear (2012) does not mention tangible programming languages in his discussion, I consider these to be an extension of his scale beyond third-generation languages. My motivation is that first, second, and third-generation languages are limited to symbols that can be created using a keyboard. Tangible programming languages overcome this constraint by including physically instantiated symbols. My work as presented in Chapter 6 extends this by making it possible for the programmer to use personally meaningful symbols and other sign-representamens.

2.4.3.1 First-generation languages

A program is comprised of numeric symbols (Brookshear 2012) and a modern CPU can directly interpret these symbols. The language that prescribes the symbols and their valid combinations is called *machine language*. Machine language is often referred to as a *first-generation* programming language. By convention, these symbols are written and displayed using hexadecimal notation. An example of a numeric symbol that can be directly interpreted by a particular CPU is the four-symbol sequence [4056]. For this CPU, this sequence indicates that the contents of register six will be copied to register five. A different CPU will interpret this symbol sequence in another way. Consequently, a program written using a first-generation programming language may not always produce the same result on another CPU. A first-generation programming language is therefore CPU specific.

2.4.3.2 Second-generation languages

Creating programs using machine language can be slow, error prone and tedious (Aho, Lam, Sethi & Ullman 2007) and not suitable for a CPU other than the original. A mnemonic system was developed in the 1940s to address these problems (Brookshear 2012). The result was a descriptive character sequence to replace hexadecimal symbols. For example, the symbol sequence [MOV R5,R6] is a more legible representation of the [4056] hexadecimal sequence. However, the CPU cannot interpret mnemonic sequences and therefore the sequence of mnemonics is changed into machine language instructions using a converter. Such a converter is also called an *assembler program* and the system of mnemonics is called an *assembly language*. Assembly language is generally considered a *second-generation language*. For a certain CPU, the difference between a first generation programming language and a second-generation programming language lies primarily in the symbols used with a one-to-one mapping between the symbols of the two languages. To illustrate the mapping principle, consider the above example where [40] maps to [MOV], [5] maps to [R5], and [6] maps to [R6]. Even though better suited to the user, a second-generation language remains CPU dependant.

2.4.3.3 Third-generation languages

Programming using either a first or a second-generation programming language requires the programmer to explicitly state each step the CPU has to take by using the elementary CPU primitives such as [MOV]. Brookshear (2012) argues that program design is simpler when high-level primitives replace the lower-level CPU ones found in first and second-language generations. To illustrate the difference, compare the solutions to the typical problem of determining the total cost of a shipped product. In this example, the total cost equates to the price of the item plus the cost of shipping. In pseudo code and using high-level primitives, the calculation can be expressed as [TotalCost = Price + ShippingCharge]. Languages that include high-level primitives like these are called *third-generation programming languages* of which the C (Kernighan et al. 1988) language is an example.

Figure 2-11 illustrates how a computation problem is solved using first, second, and third-generation languages. From this, it is clear that comprehension improves in tandem with the generation of the language.

Problem statement
"Calculate the total cost, being the price of the item plus the cost of shipping the item to the buyer."

Solution 1
Using a first generation language:

```
156C
166D
5056
306E
C000
```

Solution 2
Using a second generation language:

```
LD R5,Price
LD
R6,ShippingCharge
ADDI R0,R5 R6
ST R0,TotalCost
HLT
```

Solution 3
Using a third generation language:

```
int TotalCost (int Price, int ShippingCharge)
{
   return (Price + ShippingCharge);
}
```

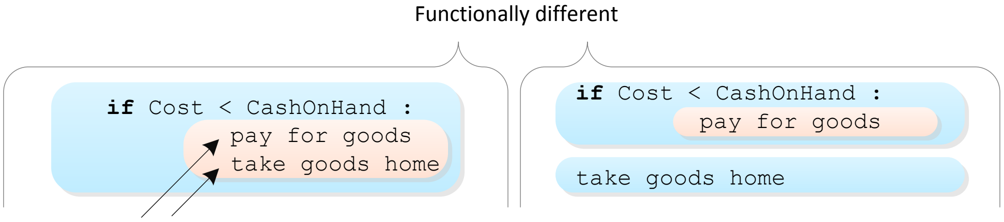
Figure 2-11 Solving a problem using first, second, and third-generation programming languages

Based on Brookshear (2012)

2.4.4 Language format

A *lexical analyser* determines which symbols in the source program constitute a meaningful sequence and it then classifies the meaningful sequence as being, for example, an arithmetic operator or a numeric value. These meaningful sequences are called *lexemes* (Aho et al. 2007) and *grammar* is the set of rules that prescribe the way these may be combined (Hein 1996). For text-based source-programs, the symbols are alphanumerical characters and a lexeme is a grouping of alphanumerical characters separated by spaces. In Chapter 6, I apply Gestalt principles of perception to identify and classify lexemes in a program written using my T-Logo programming language.

To simplify the parsing process, third-generation language designers often delineate lexemes according to a fixed format. These are called *fixed-format languages* (Brookshear 2012) of which Python (Gift & Jones 2008; Rossum 2012) is an example. When writing a program using this language, consecutive line indentations indicate that statements belong together. For example, the two instructions in Figure 2-12 (left) are executed as a set. In contrast, the same instructions on the right are not interpreted as a set. Shading in this figure highlights the difference.

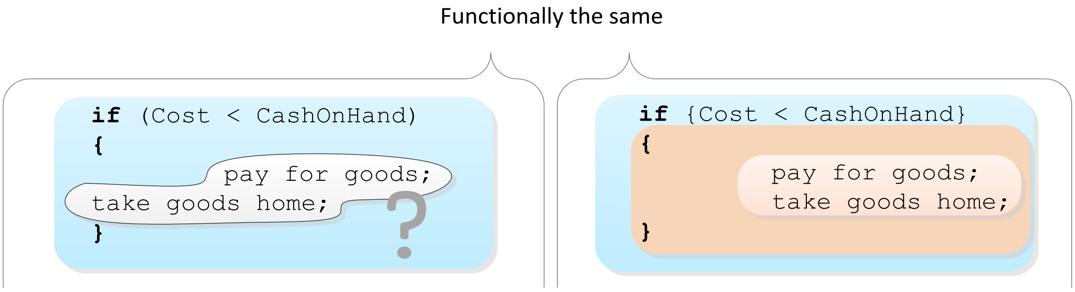


```
if Cost < CashOnHand :
        pay for goods
        take goods home
```

```
if Cost < CashOnHand :
        pay for goods
take goods home
```

Figure 2-12 A comparative example of a fixed-format language

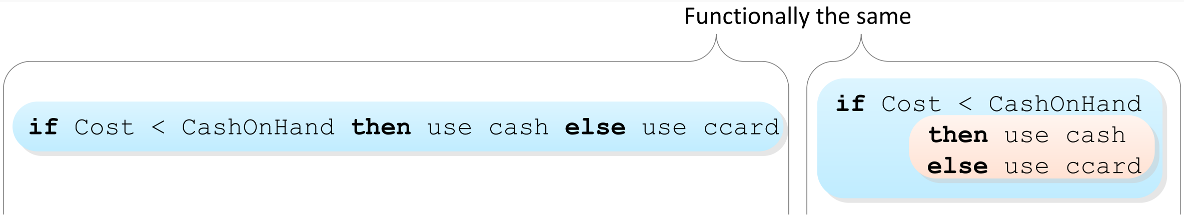
Free-format languages use special symbols such as punctuation and key words for parsing which means that the programmer has discretion to manipulate the visual appearance according to his interpretation of the task at hand (Brookshear 2012). Indeed, a recent study found that variations in horizontal and vertical spacing in written programs as illustrated in Figure 2-13 and Figure 2-14 affect a programmer's comprehension (Hansen, Goldstone & Lumsdaine 2013). The two sequences in each figure are functionally equivalent yet those on the right are more comprehensible. The sequence on the right in Figure 2-13 is easily perceived as one group nested within another but the grouping on the left is not so obvious. In addition, the perception is that the sequence on the left in Figure 2-14 is a single group while that on the right is one group encapsulated within another. The two Gestalt principles of perceptual grouping by common region and proximity explain this phenomenon. In Chapter 6, I apply these Gestalt principles to my T-Logo programming environment.



```
if (Cost < CashOnHand)
{
              pay for goods;
take goods home;
}
```

```
if {Cost < CashOnHand}
{
          pay for goods;
          take goods home;
}
```

Figure 2-13 Two examples of valid vertical allignment



```
if Cost < CashOnHand then use cash else use ccard
```

```
if Cost < CashOnHand
     then use cash
     else use ccard
```

Figure 2-14 Two examples of programmer-applied discretion in the case of a free-form language

Based on Brookshear (2012)

2.4.5 Conclusion to this section

I have shown that a computer is a general-purpose tool that can be programmed to perform certain tasks. However, for the computer to perform as expected the user must use SIGNs that both the computer and the individual agree on. To achieve this, layers of abstraction provide an interface between the computer's discrete electrical signals and the user's tangible representation of the task. These layers therefore make it possible to use personally meaningful objects for this purpose. Section 2.5 ties together the above discussion on perceptual organisation, signs, and programming.

2.5 Perceptual organisation, signs, and programming

For Saussure (2011), the sign has no inherent value yet it gains value when considered in relation to other signs. The relation between signs can be sequential (as for written texts) or concurrent as observed in visual images, dance movements, and orchestral performances (Chandler 2007).

To illustrate the value of signs and how these values can change, Saussure used as example positions that chess pieces occupy on the playing surface. The value of the pawn game piece is due to the threat it poses to the opponent's pieces and its defensive property to shield pieces from attack. In addition, this value changes according to the position the piece holds relative to other board pieces. Not only does one piece hold a position relative to another piece, but concurrent relations exist between all game pieces. Therefore, when the player contemplates his next move he does not consider each piece in isolation but instead studies the board holistically.

Not only are spatial relations important in board games, but according to Chandler (2007) they are also key in drawings, paintings, photography, and written language due to the way text is laid out in shape poems, magazines, newspapers, and notices. Chandler adds that these relations communicate culturally important concepts. Smith (1952) explains the importance of spatial relations in African cultures by using the example of positions relative to the human body; that is, the right side of the body signifies strength and the left side signifies weakness. Additional examples of spatial relations include direction (North, South, East, and West) and position (in front of, behind, close to, far away from, to the left of, to the right of, at the centre of, at the peripheral of) (Chandler 2007).

The axis and the direction in which members of a culture interpret texts influence the sequence in which visual images are interpreted (Chandler 2007). For example, societies that generally use the English language for communication interpret text from left to right and from top to bottom (I refer to these as Western societies). Other societies include Arabic, Hebrew, and Chinese and their interpretations differ from those of Western societies. To Western societies, past events that include something old or already given, or something that the reader already knows are usually associated with a position to the left of the centre (Chandler 2007). Chandler contrasts this with future events (for example something new or not yet known) that are usually often with a position to the right of the centre. He adds that a position to the right of the centre could also represent something yet to be contested, something that could be a problem, something surprising, or something that the viewer has not yet agreed upon. The vertical dimension is also significant: Up or higher positions along the vertical axis signify the ideal, having control, being rational, or the concept of "what may be". Lower positions are often associated with someone being subjected to power, death, emotions, being practical, something informative, and reality.

2.5.1 Perceptual organisation and semiosis

Tangible programming environments are based on an arrangement of physical objects. How the individual and the interpreting program view objects respectively determine the way the individual will arrange the objects and the program executes.

The Gestalt principles of perceptual organisation have a direct bearing on the individual's semiosis process and it is only after perceptual organisation has stabilised that semiosis can happen. A representamen is the result of perceptual organisation while an interpretant forms due to semiosis. Figure 2-15 illustrates this concept.

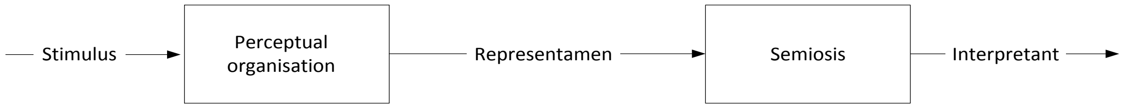


Figure 2-15 Interplay between perceptual organisation and semiosis

To illustrate this interplay, consider the visual stimulus (Chandler 2007) in Figure 2-16 (left). The viewer perceives either a white goblet on a black background (labelled Representamen A), or the silhouette of two faces (Representamen B).

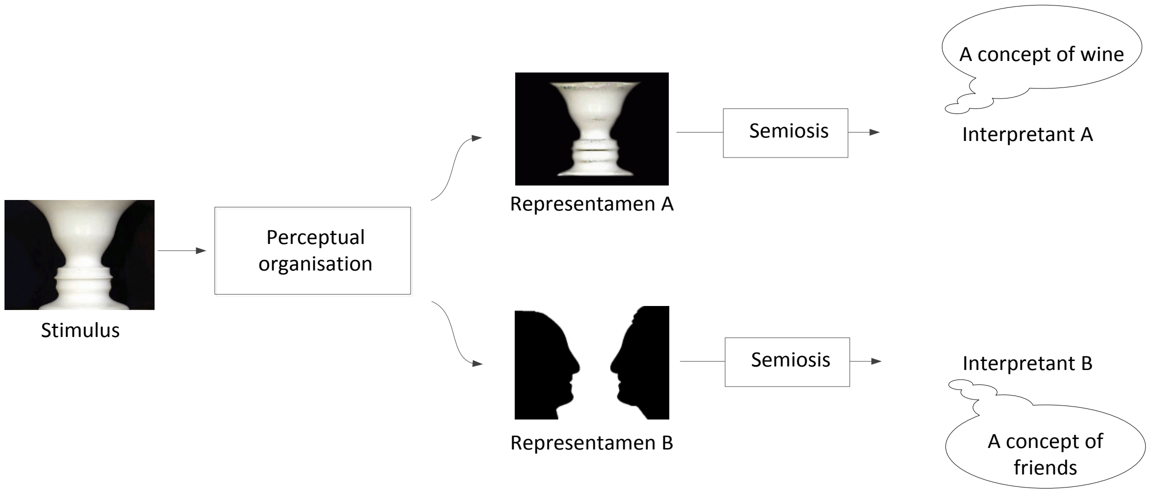


Figure 2-16 The perceptual organisation of a single stimulus can lead to multiple interpretants

Based on Chandler (2007)

As I discussed in Section 2.2.5, Gestalt psychologists describe this phenomenon as figure and ground perceptual organisation. Only once the perceptual organisation is complete can semiosis begin and ultimately produce an interpretant. In this example, Representamen A could result in an interpretant that reminds the interpreter of wine whereas Representamen B could result in an interpretant that reminds the interpreter of two men. Chandler (2007) explains that the result depends on the context within which the stimulus is perceived.

2.5.2 Saussure's linguistic sign model applied to program code

Section 2.4.3 demonstrated how first, second, and third-generation programming languages can be applied to produce three equivalent coded solutions to a problem. For each language demonstrated, a segment of the solution calculates the answer to the question "what is the cost?".

In the context of Saussure's (2011) linguistic sign model, the phrase "what is the cost?" is the signified and the signifier is the code segment. The signifier varies according to the language generation. In the case of the first-generation language, the signifier is `5056`. For the second, it is `ADDI R0,R5 R6` and for the third the signifier is `Price + ShippingCharge`. Figure 2-17 depicts Saussure's linguistic sign model applied to these code segments.

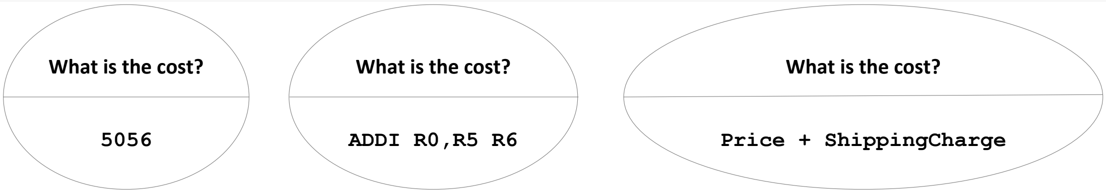


Figure 2-17 Saussure's linguistic sign model applied to three generations of computer program languages

Another example is the design of a programming language where designers are individually or collectively responsible to determine the elements that constitute program language signifiers. In the case of the third-generation language the individual has almost unlimited choice to determine these elements. This choice is possible because third-generation compilers can process user-created program elements as-if they are native to the language itself. For example, using the C language the user can compose a sequence of characters to represent a program element of which a numerical value is an example. The user achieves this by combining predefined keywords and a custom sequence of characters. Predefined keywords include `int`, `char`, and `float.` Respectively, these indicate a whole number, a character, and floating-point elements. An example of a user-created element that stands for the price of a soft drink is `PriceOfSoftdrink` and the mechanism to create it is by adding the following line to the program code: `int PriceOfSoftdrink`. To illustrate the versatility of user-created program elements, consider the following example that serves the same purpose: `int price_of_softdrink`. This demonstrates that the user can choose program elements according to his preference.

The question "what is the cost?" can be phrased in multiple ways. Alternatives include "how much do I have to pay?" and "what must I budget?". In the context of Saussure's linguistic sign model, each alternative is a signified. Nine valid combinations result when the three generations are combined with the three English phrases. The alternatives are virtually unlimited and to some extent depend only on the user's expressiveness. The user is also not limited to the English language nor required to use Western characters. For example, three alternative symbol sequences to the question “how much does it cost” are “hoeveel kos dit” (Afrikaans), 它要多少钱 (Chinese), and এটা কত টাকা লাগে (Bengali). In addition to written text, the user could express the signified by means of a pencil drawing, a sculpture, or a woodcarving.

The preceding illustrates that no universal rule determines what the signifier or the signified is; rather, the user community decides on the Saussure sign composition. I extend this concept to the programmer community and put it that individuals within that community should be at liberty to choose the signifier. In Chapter 6, I contemplate what a tangible programming environment could look like if the individual constructs program elements using personally meaningful signifiers.

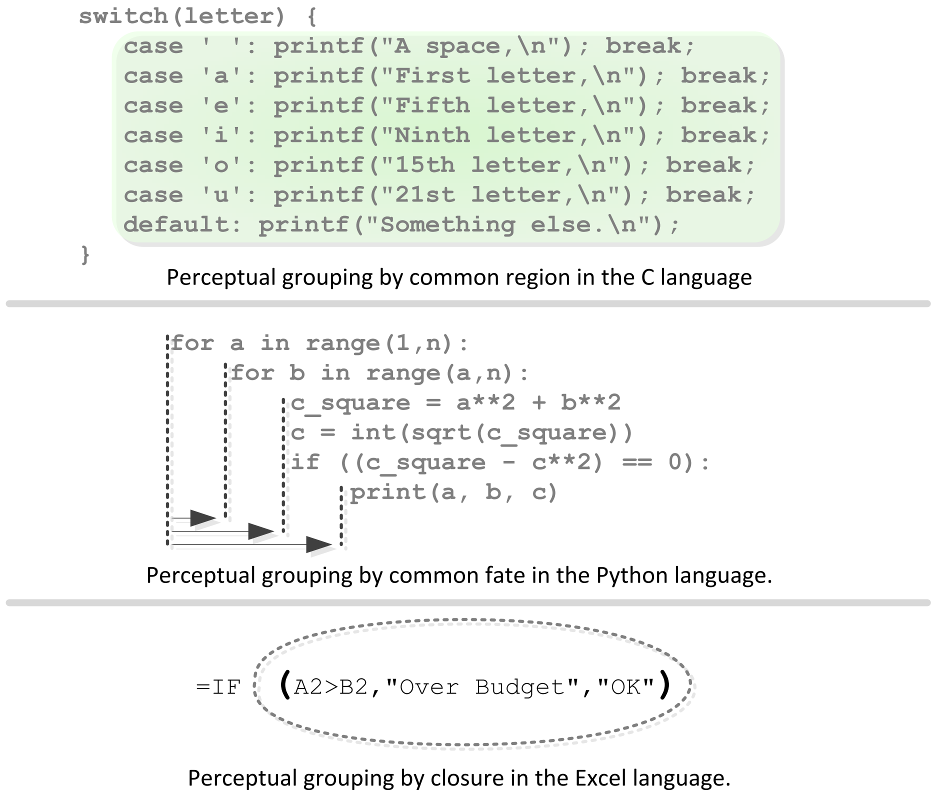
2.5.3 Gestalt principles and program code

Figure 2-18 depicts three Gestalt principles present in program segments written respectively in the C (Kernighan et al. 1988), Python (Rossum 2012), and Excel (Jones, Blackwell & Burnett 2003) languages. The first is perceptual grouping by common region where the horizontal offset on the left indicates the common region. The second example illustrates grouping by common fate. This grouping occurs because it appears to the observer that successive lines of code are increasingly shifted to the right. The last example is that of perceptual grouping by closure where the “( )” symbol pair seems to enclose the space in-between.

2.5.4 Peirce’s sign model and Gestalt principles applied to tangible program elements

My research considers tangible programming environments based on a combination of the representative and representational characters of representamens, the power of representamens to direct, and the Gestalt principles of perceptual grouping of tangible objects. For example, when the user groups an icon sign with a qualisign, she is coding a program element (the icon sign) that takes on the given property (the qualisign). Figure 2-19 illustrates how this is done in the case of a wooden toy car and a red cloth swatch.

An appropriate interpreter program can associate the car with the red swatch and process the pair as a single program element. The result is a coloured car. My T-logo tangible programming environment in Chapter 6 incorporates an interpreter that will deduce an appropriate program element when a qualisign is placed in close proximity to an icon sign.



```
switch(letter) {
   case ' ': printf("A space,\n"); break;
   case 'a': printf("First letter,\n"); break;
   case 'e': printf("Fifth letter,\n"); break;
   case 'i': printf("Ninth letter,\n"); break;
   case 'o': printf("15th letter,\n"); break;
   case 'u': printf("21st letter,\n"); break;
   default: printf("Something else.\n");
}
```

Perceptual grouping by common region in the C language

```
for a in range(1,n):
    for b in range(a,n):
        c_square = a**2 + b**2
        c = int(sqrt(c_square))
        if ((c_square - c**2) == 0):
            print(a, b, c)
```

Perceptual grouping by common fate in the Python language.

```
=IF (A2>B2,"Over Budget","OK")
```

Perceptual grouping by closure in the Excel language.

Figure 2-18 Examples that demonstrate perceptual grouping by common region, common fate, and closure

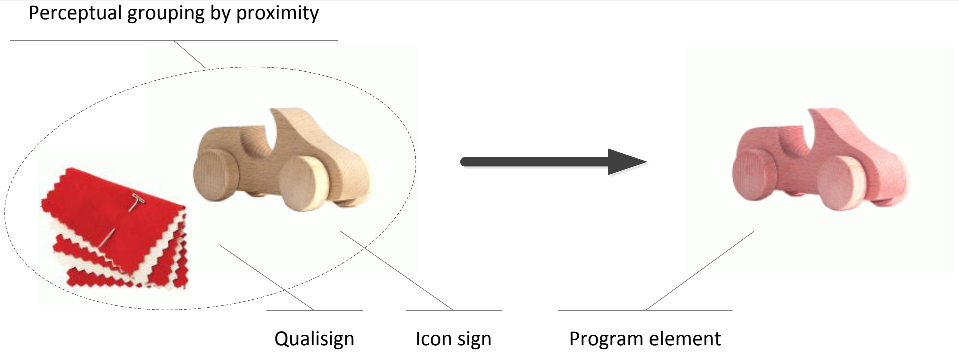


Figure 2-19 Coding a program element by grouping a qualisign with an icon sign

2.5.5 Conclusion to this section

This section illustrated the link that exists between visual perception, SIGNs, and program code. In particular, it is evident that perception must occur before a Peircian sign-representamen or a Saussurian signifier emerges. I also showed that program code is a collection of sign-representamens. I therefore conclude that program code relies on perception. However, perception is not sufficient to create usable program code; instead, a prior agreement on the meaning of the perceived phenomenon must also exist. As I showed, the role of the sign-representamen is to connect whatever is perceived with its associated program code.

2.6 Conclusion

This chapter established the theoretical foundation for this study and includes aspects of perception, signs, and programming. This research considers the development of a model that can be used to guide the design of tangible programming environments. The premise is that the Gestalt principle of perceptual grouping by proximity, together with user attributed tangible SIGN properties, can simultaneously be applied to an arrangement of tangible objects so that the arrangement represents a program. Therefore, by combining some of Saussure and Peirce's insights of signs (specifically the attributions a user makes to objects) with some of the Gestalt psychologists' insights (the phenomenon of perceptual grouping of objects in particular), my study investigates the use of physical objects as a representation of a program. My investigation requires the user to choose or construct objects of his choice and associate them with a predetermined list of program instructions. The user then groups the objects to form the program. In Chapter 6, I discuss a mechanism with which the user associates the objects with computer instructions and I present my model in Chapter 7.

CHAPTER 3

LITERATURE REVIEW: TANGIBLE OBJECTS

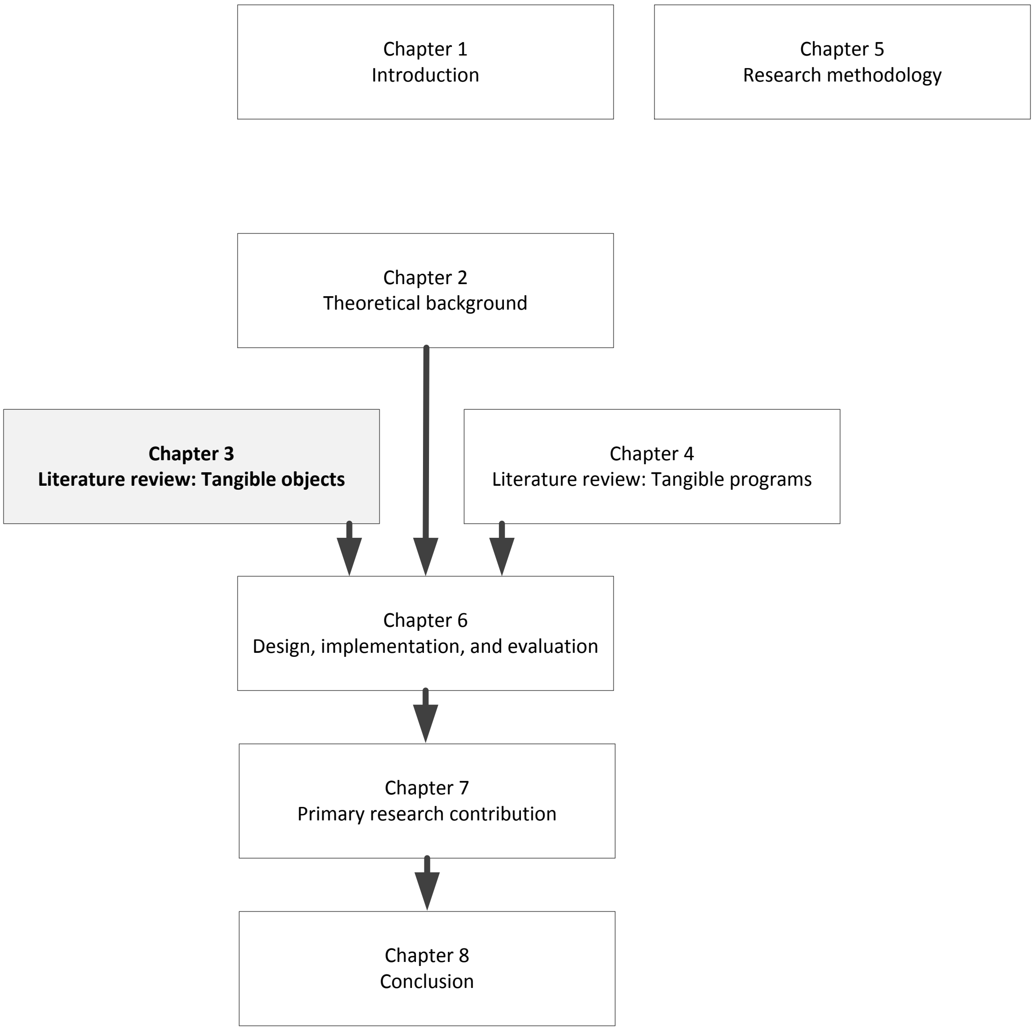

Chapter 1
Introduction
Chapter 5
Research methodology
Chapter 2
Theoretical background
Chapter 3
Literature review: Tangible objects
Chapter 4
Literature review: Tangible programs
Chapter 6
Design, implementation, and evaluation
Chapter 7
Primary research contribution
Chapter 8
Conclusion

Figure 3-1 Document structure

3.1 Introduction

This chapter discusses literature on user-created tangible objects and how tangible objects support interaction between humans and computers. Section 2 covers the concept of tangible objects as these relate to humans and computers. It also considers existing technologies shown to support the use of tangible objects in human-computer interaction activities. Particular attention is given to tangible objects that contain, point to, detect, measure, manipulate, and generate data. Systems are then discussed in which the relative positions of tangible objects are of particular significance. Finally, tangible object systems used in generating digital models are listed. Section 3 covers environments in which the user creates personally meaningful tangible objects. Motivation is also given for the user to create her own tangible objects. Having deliberated tangible objects in relation to data and computer interaction and having highlighted the need for user-created tangible objects, I then reiterate in Section 4 Ishii's (2009) omission of the object's origin in his basic TUI model. I also restate McCloud (1994) and Jacucci's (2007) opinions that can serve as system design guidelines when the objective is for the user to create personally meaningful objects. Section 5 concludes this chapter.

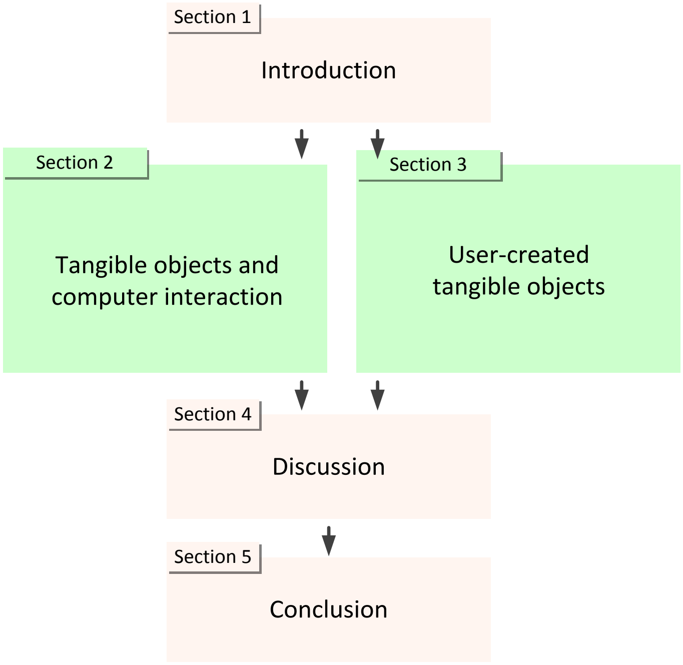


Figure 3-2 Chapter outline

3.1.1 Nomenclature

In the literature, one physical object may be referred to as a three-dimensional (3D) object and elsewhere another object may be referred to as a spatial object. The question I address here is:

"when should a physical object be referred to as a 3D object, and when should it be referred to as a spatial object?" The answer to this question depends on the context in which the reference is made.

When only the physical properties of an object are considered, referring to the object in terms of its 3D parameters is sufficient and the object itself as well as its position in space is then sufficiently described in terms of three orthogonal axes. However, it may be more appropriate to refer to the object as a spatial entity when the object is discussed in relation to living organisms. When an object is considered in the context of living organisms, it is not sufficient to only describe the relative position and orientation of the object to the subject. Other properties of the objects, such as its texture and colour, should also be described (Lannoch & Lannoch 1989). When the spatial nature of an object has been acknowledged, then, in the context of tangible user interfaces (TUIs), such an object is called a spatial TUI and thereby the importance of not only the shape of the object but also its location in space and its structure is highlighted (Jacoby, Josman, Jacoby, Koike, Itoh, Kawai, Kitamura, Sharlin & Weiss 2006).

Since the current study considers an object in relation to an individual, the object is mostly referred to as a spatial entity. Therefore, the answer to the question: "when should a physical object be referred to as a 3D object, and when should it be referred to as a spatial object?" is that in this study, it is appropriate to refer to a physical object as a spatial object.

3.2 Tangible objects and computer interaction

I discuss in Subsection 3.2.1 Ishii's (2009) TUI model that describes personal interaction with a computer using tangible objects. The varieties of meanings that objects hold are covered in Subsection 3.2.2. Subsection 3.2.3 differentiates between active and passive identity and encoding mechanisms and data exchange mechanisms as either tethered or untethered. Subsection 3.2.4 then gives an overview of gesture modalities that have been applied in support of human-computer interaction. Relationships that exist between data and tangible objects are discussed in Subsection 3.2.5. Subsection 3.2.6 concludes with a discussion on using tangible objects when creating digital models.

3.2.1 Ishii's tangible user interface model

Dix, Finlay, Abowd and Beale (2004) describe human-computer interaction as being concerned with the interaction that the user has with a computer in order to accomplish a task. According to Ghaoui (2005), and Sears and Jacko (2009), HCI is an interdisciplinary and multidisciplinary research domain that emerged from computing and includes contributions from (amongst others) computer science, psychology, cognitive science, ergonomics, sociology, engineering, education, graphic design, and industrial engineering. Ishii (2008a, 2009) reminds us that human-computer interaction design principles have progressed from requiring the user to remember commands and typing these commands (as is the case of the so-called command user interface, also written as CUI), to pointing at a visible rendition of the command using a computer mouse and selecting the rendition by the press of a mouse button (as is the case of the so-called graphical user interface, also written as GUI).

The TUI provides an alternative to both the CUI and GUI by taking form as tangible objects that both represent digital data and operate as tools for direct manipulation of digital data. The TUI is in sharp contrast to the GUI in that the GUI multiplexes the interface mechanism (mouse) in both space and time whereas the TUI makes provision for a dedicated interface for the data being manipulated. Another contrast is that the GUI is limited to intangible data representation whereas the TUI supports tangible representation.

Figure 3-3 depicts Ishii's basic TUI model. This model associates tangible representations of data, digital data, or digital computations with physical objects. The model also makes provision for changes to the associated digital entities through manipulation of the physical objects (Ishii 2009).

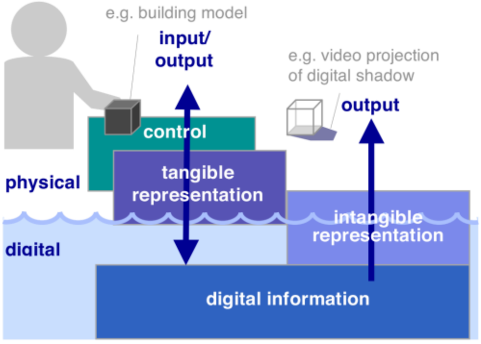


Figure 3-3 Ishii's basic tangible user interface model

(Ishii 2009)

He extended the model to include three feedback loops. The first is passive and exists between the user and the object that the user manipulates. This feedback is in immediate tactile form and exists independently of a computer. A second and active feedback loop is between the user and a computer program. The output of the computer program changes as the user manipulates the physical objects. Changes are fed back to the user in (for example) visual form. The third active feedback loop is between a physical object and data. This data may represent a digital model or it may be the result of computation and can be used to affect physical properties of the object (Ishii 2009).

What Ishii's (2009) TUI model does not reflect is how the tangible representation is conceived and by whom. Indeed, the majority of the literature discussed in this study does not indicate the user's involvement in determining the tangible representation. It is therefore reasonable to assume that, in the majority of TUI-based systems, the user is not involved in the design of the tangible representation and has to adapt to a predetermined representation.

An alternative approach extensively investigated in this thesis considers the scenario where the user decides on the tangible representation. To this end, Ishii (2008b) describes an “organic” tangible representation that allows the user to shape material supplied by a system designer. Systems that incorporate organic tangible representation make provision for the TUI system designer to prescribe the basic properties of the tangible representation yet also allow the user to change the tangible representation. Piper, Ratti, and Ishii’s (2002) Illuminating Clay and Ishii, Ratti, Piper, Wang, Biderman, and Ben-Joseph’s (2004) SandScape are examples of TUI systems that include endlessly malleable materials to facilitate organic representations.

Chapter 6 describes my T-logo tangible programming system that includes certain aspects of Ishii’s organic tangible representation. When using the T-logo programming environment, the user is free to decide what materials to use when constructing the programming objects.

Fitzmaurice, Ishii and Buxton’s (1995) Graspable User Interfaces and Ishii and Ullmer’s (1997) Tangible Bits explored mechanisms that allow a person to directly “touch” data using graspable media. Ishii and Ullmer’s research categorised a user’s attention as being in either a “foreground” or “background” state. They argued that when the user’s attention is in the foreground the user focusses on the task. Conversely, they argued that when a user’s attention is in the background then his attention is not centred on the task and the user remains aware of her immediate surroundings (the periphery). Ishii and Ullmer viewed the two states as being mutually exclusive. Their graspable media was designed to be used at the centre of a user’s attention and their ambient media was to be used at the periphery of the user’s attention. They also aimed to develop a type of human-computer interface that allows the user to “touch” data stored within the computer. They dubbed this type of human-computer interface a *tangible user interface* (Ishii & Ullmer 1997). Not only did they consider solid objects as potential tangible user interfaces but they also considered fluid-like mediums such as audio waves, visible light, flow of air, liquid, and gas that Ishii (Ishii 2009) called ambient media. These fluid-like mediums were recognised as possible TUIs and specifically for use in the background of a user’s attention. Ishii refers to objects that are both physical and graspable as *tangible objects* (Ishii 2009). My research considers a programming environment that requires focussed attention and the manipulation of solid objects.

A mapping between specific GUI elements and TUI elements emerged from research conducted during the 1990's. Table 3-1 gives a selection of these mappings while Figure 3-4 illustrates them.

Table 3-1 A selection of mappings between Graphical User Interfaces and Tangible User Interfaces

Graphical User Interface element	Tangible User Interface element	Description
window	lens	The GUI window is mapped to a tangible frame, called a *lens*. The lens can be positioned in physical space, with a display within the frame changing its projection under software control.
icon	phicon	The GUI icon is mapped to a tangible object that represents specific digital data and can be positioned in space.
menu	tray	The GUI menu is mapped to a physical *tray* that may contain one or more physical objects, with each object a selectable item.
handle	phandle	The GUI handle (used in changing the size of a GUI window) is mapped to a tangible object called a *phandle*.
widget	instrument	The GUI widget (that is often used in adjusting linear quantities) is mapped to a tangible slider mechanism called an *instrument*.

(Ishii & Ullmer 1997)

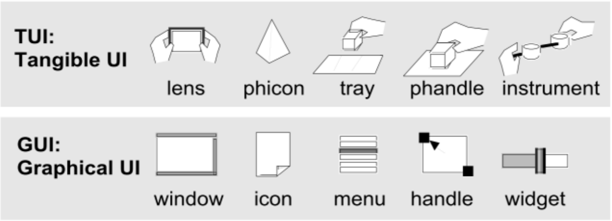


Figure 3-4 A selection of mappings between graphical user interfaces and tangible user interfaces

(Ishii & Ullmer 1997)

According to Ishii (2008b), researchers developed TUIs that incorporate tangible materials of which the shape is integral to the role of the TUI. An example of a TUI of which the shape is significant to its meaning is the tangible equivalent of the GUI widget (see Figure 3-4, bottom right). In addition to the research into fluid-like TUIs for use at the background of a user's attention, second generation fluid-like TUIs were also developed for applications at the centre of the user's attention. Examples of second generation TUIs include sand and clay. These fluid-like TUIs can be reshaped, with their digital representation changing at the same time. In addition to some TUIs that can be reshaped by the user, other TUIs can be reshaped by software through a process called actuation. Lumen (Poupyrev, Nashida, Maruyama, Rekimoto & Yamaji 2004) and Ohkubo, Ooide, and Nojima's (2013) "smart hairs" are examples.

Of particular interest to my own study is Ishii's consideration of objects that can be grasped and are found in the home or office environments. Ishii suggested that such objects can be used at the centre of a user's attention. He and Ullmer were particularly interested in exploiting the rich affordances that physical objects offer. Ishii also considered the potential of architectural surfaces serving as tangible user interfaces, dubbing these *interactive surfaces*. Such surfaces were to be used at the centre of a user's attention. Examples of interactive surfaces include those found as part of a building structure (an office wall is an example) and those that serve as furniture (the office desk, for example). Having now established that physical objects can represent and manipulate data, it is worthwhile to consider what meanings can be attributed to physical objects.

3.2.2 The meaning that objects hold

For a user, an object may hold personal meaning in that it facilitates the recall of a particular memory. Ullmer (1997) calls these “physical objects with embedded memories”. Ullmer and Ishii (2000) offer a seashell as an example of an personally meaningful object that helps its owner recall a holiday experience. They coined the term associative tangible user interfaces to describe objects used independently of others to represent digital information. The advantage of using a personally meaningful object as opposed to an object selected by another person is that the user does not have to form a new mental model associating the object with the information (Hoven & Eggen 2004). Streitz et al.’s (1999) InteracTable is an example of systems that represent information using personally meaningful objects.

To describe the multiple meanings an object may hold, Underkoffler and Ishii (1999) proposes the design space as illustrated in Figure 3-5. They view this object design space as a continuum. Positions along this continuum are identified where an object can be interpreted as being a pure object, an attribute, a noun, a verb, and a reconfigurable tool. In addition to these interpretations, I propose that an object may represent a quantity (a numeric value). Such an object represents more than itself and yet it does not exist in relation with another object. I therefore locate such an object in this design space at a position between the object as a pure object and the object as an attribute. This location describes an object that serves to represent a quantity that is optionally part of a set of discrete values. The following are examples of physical objects that represent discrete and continuous quantities, nouns, adjectives, and a verb.

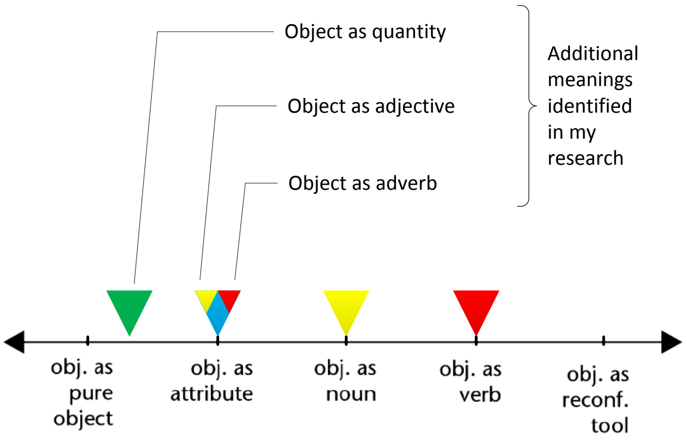


Figure 3-5 A continuum of the meaning that objects hold

Based on Underkoffler and Ishii (1999)

A glass marble is an example of an object that represents a quantity in a set of discrete values. Here, the set of discrete values consists of a bag filled with glass marbles and one or more marbles represent a quantity. A quantity described by an object may also be from a set of continuous values. An example of an object that represents a quantity from a set of continuous values is Dietz and Eidelson‘s (2009) SurfaceWare drinking glass. This drinking glass design varies the light reflected through the bottom surface according to the quantity of water in the glass. The system can determine the quantity of water when combined with an appropriate sensing surface. Figure 3-6 illustrates the operating principle: When the content is below a predetermined level, light that enters from the bottom reflects back to a sensor. Conversely, no light returns when the water is above this level.

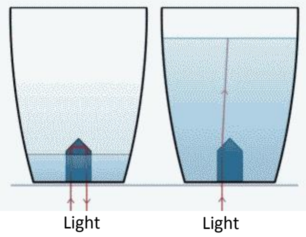


Figure 3-6 The SurfaceWare drinking glass

(Dietz & Eidelson 2009)

The three fictitious genies (Mazalek 2001) Opo, Junar, and Seala are examples of objects that represent nouns. Glass bottles in Figure 3-7 represent the genie characters. Each design highlights the colour, texture, and form of the respective genie personality to help the user associate a bottle with a genie. The short and round Opo object is coloured yellow and green with a matte finish. This form, texture, and colour combination reflects the dull and depressive Opo personality. Junar’s representation is angular and cackled with pink and bright orange colours that reflect his abrasive personality. Seala is a water genie and her physical representation is blue, smooth, and tall. This combination creates an impression of flowing water.

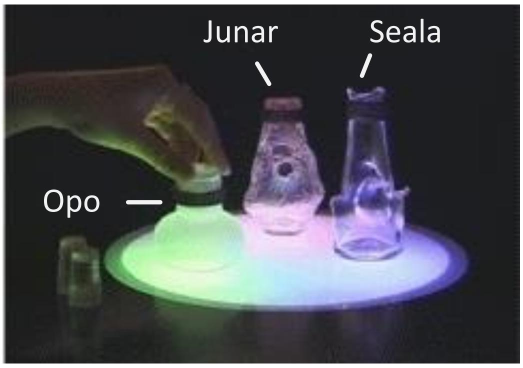


Figure 3-7 Three objects represent three nouns

(Mazalek 2001)

Tangible program elements interlock in Oh et al.’s (2013) Digital Dream Labs programming environment to express a program action. The elements on the left in Figure 3-8 represent program elements of a noun, two adjectives, and a verb. These can be mapped on a continuum as shown in Figure 3-5. A user constructs a program action by interlocking these elements as shown on the right and the physical constraints limit a program action to this combination of program elements. My T-logo programming environment as discussed in Chapter 6 removes this constraint by accommodating multiple simultaneous instances of an object that represents a quantity. For example, when a quantity of 10 is called for in a program, the user is free to use a combination of objects that add up to this number.

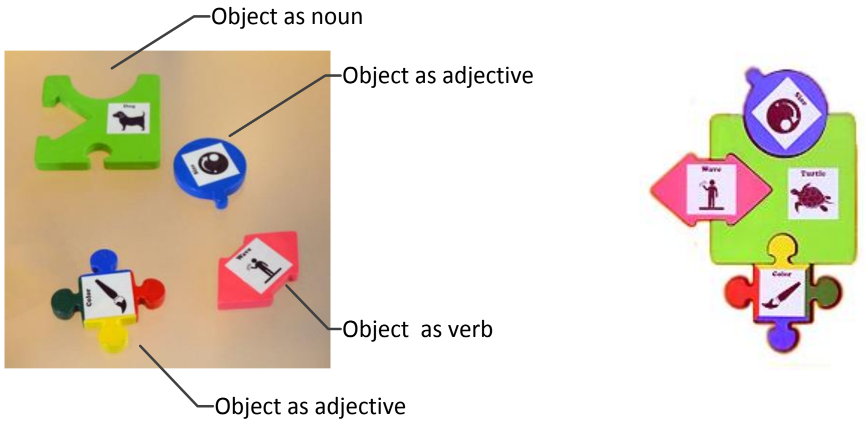


Figure 3-8 Digital Dream Lab objects interlock to form a tangible programming object

(Oh et al. 2013)

My research considers objects that hold personal meaning for the user and are applied to computer programming. The two preceding subsections considered how objects represent data and the meanings that objects hold for the user. Still missing from the discussion is how the computer determines the object position. To address this, Subsection 3.2.3 provides an overview of technologies that feeds this data to the computer.

3.2.3 Supportive technologies

Tangible interaction systems include mechanisms that exchange data between the system components and other mechanisms that encode both the identity and position of tangible objects. A range of technologies support these mechanisms. Figure 3-9 illustrates my stack and system perspectives on the role of supportive technologies within a three-component tangible interaction system. The data exchange mechanisms, and identity and position encoding mechanisms (as shown in Figure 3-9) are discussed below.

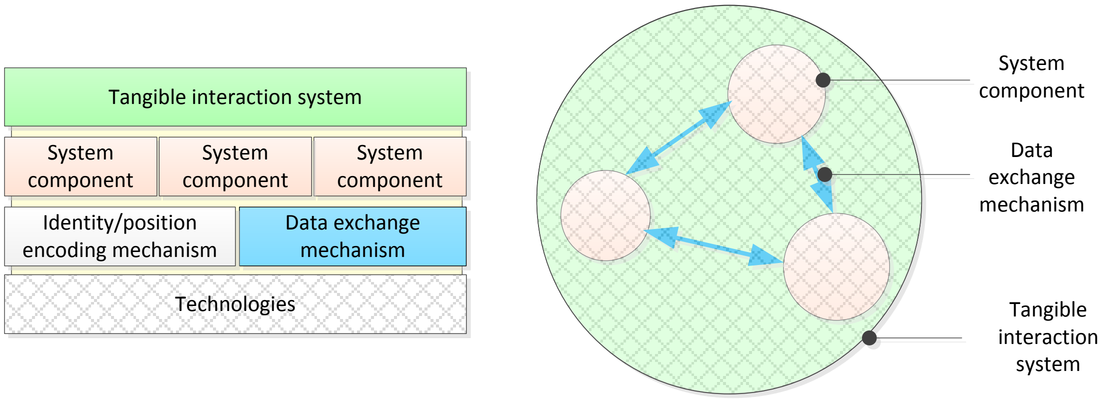


Figure 3-9 My stack and system perspectives on the relationships that exist in tangible interaction systems

I classify mechanisms by which system component data are exchanged as *tethered* and *untethered* and I refer to the respective tangible systems that incorporate them as *tethered* and *untethered tangible systems*. Tethered tangible systems are characterised by wires or direct physical contact to interconnect components. Examples of tethered tangible systems are Horn & Jacob's (2007) Tern and Suzuki & Kato's (1995a) AlgoBlock. Untethered tangible systems incorporate objects that have no wires leading to other objects and neither is it a requirement that an object be in physical contact with other objects. Examples of untethered tangible systems are Bricks (Fitzmaurice et al. 1995) and musicBottles (Ishii 2004) .

In addition to classifying the mechanisms according to how data are exchanged, I also classify mechanisms on whether or not either the identity or the position of an object is encoded. Horn et al. (2008) proposed *passive tangible interface* terminology to describe physical objects that neither require a constant source of electricity nor maintain a continuous link to a digital system. Applying their terminology, I classify mechanisms by which data is exchanged between system components as either *active* or *passive*. I interpret Horn et al.'s use of the word "passive" as comprising two independent mechanisms. The first is the provision (or absence) of electricity supply to the tangible objects whereas the second considers the provision (or absence) of a link to a digital system. Figure 3-10 illustrates my interpretation of the relationships between the mechanisms.

To conclude, I refer to the respective tangible systems that incorporate active or passive mechanisms as *active tangible systems* and *passive tangible systems.* By applying my classification scheme, I can state that an active tangible system relies on embedded electronics to encode either the identity or the position of a tangible object. Examples of active tangible systems include Schiettecatte and Vanderdonckt's (2008) AudioCubes and Reitsma's (2011) StoryBeads. Conversely, a passive tangible system does not rely on embedded electronics to encode either the identity or the position of a tangible object. An example of a passive tangible system is the Marble track music sequencer (Fischer & Lau 2006). These examples do not consider the mechanisms by which data are exchanged. Date exchange mechanisms as applied to untethered passive and untethered active tangible systems are discussed in the following sections.

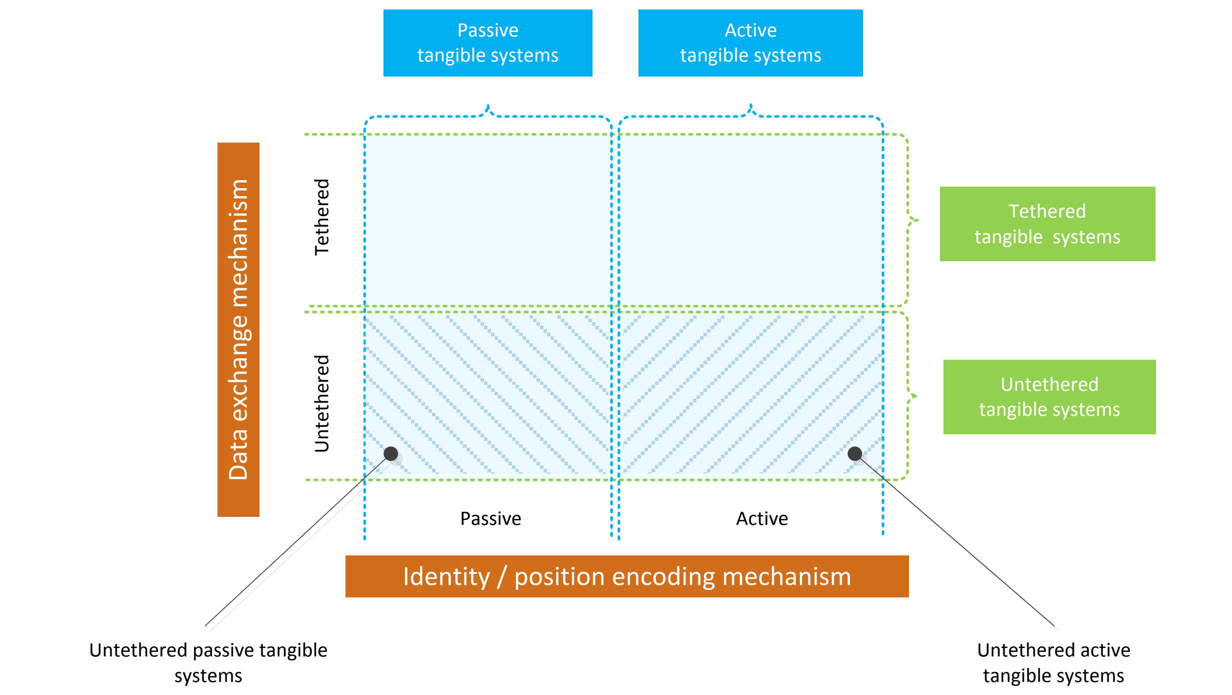

Passive tangible systems
Active tangible systems
Data exchange mechanism
Tethered
Untethered
Tethered tangible systems
Untethered tangible systems
Passive
Active
Identity / position encoding mechanism
Untethered passive tangible systems
Untethered active tangible systems

Figure 3-10 My classification matrix of mechanisms in tangible systems mapped according to data exchange, and identity encoding/position encoding

3.2.3.1 Technologies that support untethered passive tangible systems

I map technologies that simultaneously support untethered data exchange and passive identity/position encoding mechanisms of the bottom-left corner in Figure 3-10. As reported in the literature, magnet and vision based technologies are pervasive in untethered passive tangible systems. Examples of untethered passive tangible systems that include magnet and vision based technologies are described next.

The following illustrate magnet based technologies that support untethered passive tangible systems. My `GameBlocks` (Smith 2007b) and `Dialando` (Smith 2010a) systems use static magnetic fields to sense object position and orientation. Mazalek, Davenport and Ishii's (2002) `Tangible Viewpoints` applies electromagnetic resonant circuits to detect objects on a horizontal interaction surface while `Tangible Viewpoints` uses loop antennas inside the surface to determine the position and identity of objects on top. The latter is possible by embedding a coil and capacitor resonator circuit inside each object. `Actuated Workbench` (Pangaro, Maynes-Aminzade & Ishii 2002; Pangaro 2003) is an interaction system in which a table top contains embedded electromagnets and these move objects in two dimensions under software control. The software also determines when the user disturbs an object. Some systems, such as those developed by Ishii, Fletcher, Lee, Choo, Berzowska, Wisneski, Cano, Hernandez and Bulthaup (1999), Mazalek, and Lee (2001) and Ishii (2004) combine glass bottles and electromagnetic resonant circuits to detect the presence of bottle stoppers.

Examples of vision-based technologies that support untethered passive tangible systems are the following: Jorda, Kaltenbrunner, Geiger, and Bencina's (2005) reacTable incorporates Bencina, Kaltenbrunner and Jorda's (2005) reacTIVision vision software to detect and track optical markers in two dimensions. The system senses the position of objects placed on the table in real time and gives user feedback by means of a visual display projected onto the translucent surface. Tangible Object Placement Codes (TopCodes) (Horn 2007, 2009) is another vision-based detect-and-track system based on optical markers that generates data similar to that of Kaltenbrunner and Bencina's (2007) reacTIVision. Underkoffler and Ishii (1998), Underkoffler et al. (1999), and Underkoffler's (1999b) I/O Bulb integrates image capturing and projection technology into a single object. Using I/O Bulb, objects on a flat surface are sensed and processed with the result then projected onto the surface. Coloured dots on the objects enable tracking. In the Diorama Table (Takahashi & Sasada 2005; Takahashi 2007a), a camera and projector are mounted above the table on which the user positions everyday objects. The visioning system then detects and analyses the objects and projects animated images onto the table. Another vision-based tangible system is Tseng, Bryant and Blikstein's (2011) Mechanix that consists of magnetised tangible objects, a video camera and projector and a semi-transparent vertical screen with an embedded ferromagnetic mesh. The screen supports rear-projection and serves as a surface onto which magnetised tangibles can be attached. The camera detects optical markers on the tangibles and sends this data to a system for processing. Results are then projected onto the surface.

3.2.3.2 Technologies that support untethered active tangible systems

I map technologies that simultaneously support untethered data exchange mechanisms and active identity/position encoding mechanisms to the bottom-right quadrant in Figure 3-10. Inductive, electromagnetic, optical, and acoustic wave are examples of relevant technologies. Systems that incorporate these technologies are described next.

Topological data is exchanged between tangible cubes by means of infrared light in Schiettecatte and Vanderdonckt's (2008) AudioCubes. In addition to supporting inter-object communication, optical technologies can also be applied to position sensing. For example, infrared beacons can be used to associate a physical object to the room in which it is located (Want, Fishkin, Gujar & Harrison 1999).

Fernaeus (2007) also applies the electromagnetic spectrum in `Patcher` by integrating radio frequency identification (RFID) tags with a reader to track the positions of objects on a floor mat. In addition to tracking objects, RFID-based systems can also associate physical objects with data. Want et al. (1999) offer two examples of RFID-based systems that associate physical objects with data. First, photographs on the sides of a cube contain RFID tags with the addresses of associated web sites. Second, an RFID tag inside a wristwatch triggers the computer to display the wearer's diary.

Keeping to the electromagnetic spectrum, albeit at a much lower frequency, `SenseTable` (Patten et al. 2001) incorporates inductive coupling between a sensing surface and the objects placed on the surface. A wire grid below the sensing surface conducts low-frequency alternating current while objects contain resonant circuits. These circuits absorb maximum energy at a pre-set frequency and by varying the frequency of the signal within the wire grid, the identity of the object can be determined. The object coordinates are determined by constantly alternating the row and column in which the current flows.

Systems such as `TViews` (Mazalek 2005) rely on pulsed acoustic waves that travel along a horizontal surface from predetermined fixed positions. When the waves reach an object the detection circuitry inside the object sends infrared signals to a common receiver. In turn, the receiver applies triangulation techniques to calculate the position of the object on the surface.

I conclude this subsection on supportive technologies with a comment by Horn et al. (2008). They comment that (when compared to an active tangible system) a passive tangible system offers the system designer a wider choice of materials and designs with which to implement a solution. Additional benefits include improved durability, improved robustness, and reduced cost. In the research reported on in this thesis users partially assume the role of the system designer in that the users are encouraged to create their own tangible objects. Having considered the above, I base my designs in Chapter 6 on untethered passive tangible systems.

This section discussed mechanisms for connecting objects to the computer. Section 3.2.4 considers how gestures can manipulate data and associate objects with data.

3.2.4 Supportive gesture modalities

According to Pedersen, Sokoler and Nelson (2000), any physical object may represent a digital "object" and I consider data to be examples of digital objects. From this, I deduce that any physical object can represent data. I use the terms *binding* and *unbinding* to describe the mechanisms by which a physical object is respectively associated with data and disassociated from data.

Merrill et al.'s (2007) `Siftables` system uses gesture modalities to bind and unbind object to data. They use the term *interaction language* to describe these gesture modalities. The `Siftables` gesture modalities are illustrated in Figure 3-11. `SenseTable` (Patten et al. 2001, 2006) also demonstrates binding and unbinding mechanisms and the description of these are included in the following subsections along with `Siftables`.

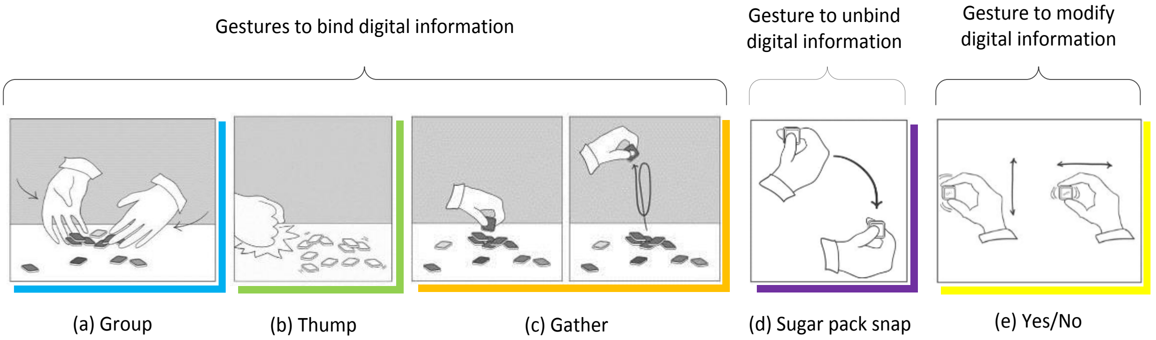

Gestures to bind digital information
Gesture to unbind digital information
Gesture to modify digital information
(a) Group
(b) Thump
(c) Gather
(d) Sugar pack snap
(e) Yes/No

Figure 3-11 The Siftables interaction language

Based on Merril et al. (2007)

3.2.4.1 Gestures for binding data

Discrete tangible objects in the `Siftables` (Merrill et al. 2007) system are called *squares*. The `Siftables` interaction language comprises of three gesture modalities by which a square interacts with data. Interaction includes binding, unbinding, and modification. Binding is discussed here while the unbinding and modification modalities are covered in Subsections 3.2.4.2 and 3.2.4.3.

Figure 3-11 depicts the binding gestures dubbed *group*, *thump,* and *gather*. Using the grouping gesture, the user gathers squares together to indicate that squares have something in common. The thump gesture assigns new data to all affected squares while the gather gesture associates a particular square with the aggregated data of other squares. In Chapter 6, I explore grouping tangible objects in my T-logo tangible programming environment.

Three mechanisms in the `SenseTable` (Patten et al. 2001) system bind a puck to data and Figure 3-12 illustrates these. The first is to place a puck nearby the projected data. An alternative is to position the puck within a demarcated area. Finally, the blue puck is associated with another puck when they are in close proximity to each other.

3.2.4.2 Gestures for unbinding data

To gesture that a particular `Siftable` (Merrill et al. 2007) should be disassociated from data the user holds the square and makes a swift downwards motion. Merril et al. refers to this as a s*ugar pack snap* motion and Figure 3-11 (d) illustrates the action.

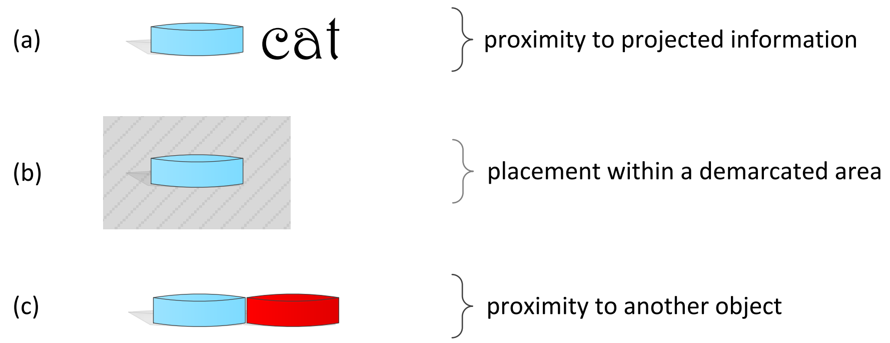

(a)
cat
proximity to projected information
(b)
placement within a demarcated area
(c)
proximity to another object

Figure 3-12 Three SenseTable mechanisms to bind physical objects to data

Using the `SenseTable` system, the user can use any of three actions to unbind a puck from data. The first is to execute a series of rapid left-right puck motions whilst keeping the puck in contact with the sensing surface. An alternative is to remove the puck from the sensing surface. The third mechanism is to bind the puck to alternative data; a puck is alternatively bound and unbound to data when the puck is moved into a demarcated position as shown in Figure 3-12 (b).

3.2.4.3 Gestures for modifying data

In addition to representing data, a physical object can also modify data. Examples are Merril et al.'s (2007) `Siftables` and Patten et al.'s (2001, 2006) `SenseTable` systems that can associate objects with data and use the same objects to modify data.

Using `Siftables`, a user modifies data by executing the *Yes/No* gesture illustrated in Figure 3-11 (e). In the `SenseTable` system, a combination of three modifier objects and gestures can change data that has previously been associated with a puck. The first modifier object is a rotation knob that plugs into a puck as illustrated in Figure 3-13. Data changes when the user turns the knob.

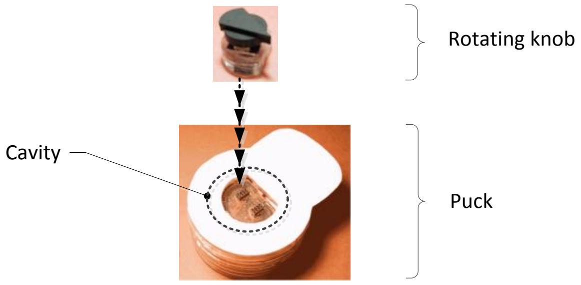


Figure 3-13 A SenseTable puck and rotation knob combination

Based on Patten et al. (2001)

A push button is the second modifier. Data changes when the user slides the puck-button combination across the input surface while depressing the button. This button also selects data. The third gesture is to vary the distance between two pucks. Figure 3-14 depicts two pucks that each represents a microphone and a music track. The music volume level changes according to the distance between the pucks.

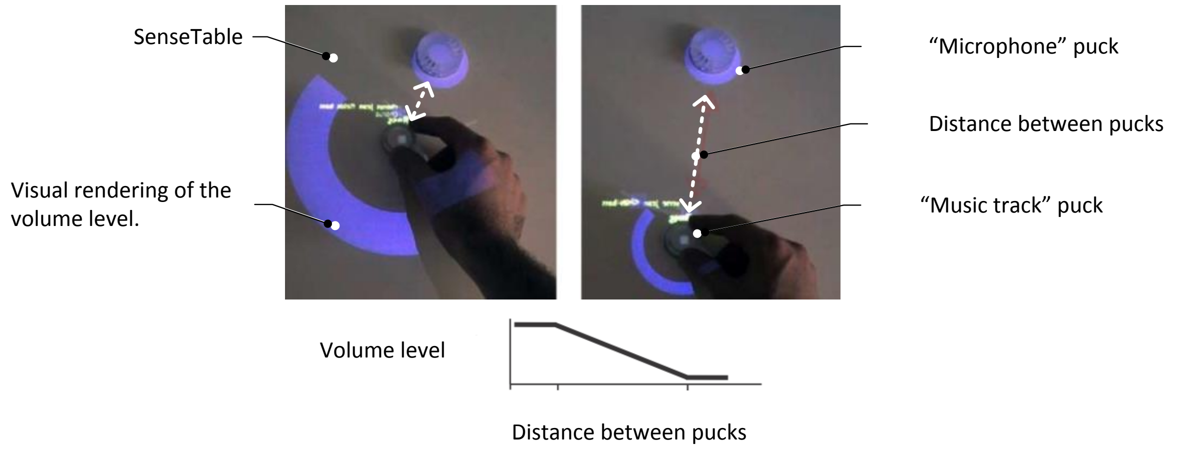


Figure 3-14 The distance between two physical objects alters data

Based on Patten et al. (2006)

Figure 3-15 is a photograph of a puck that a user has augmented with a battery. The added battery reminds this user what the function of this puck is. This modification inspired the design of my T-logo tangible programming environment in which the user assumes some of the responsibility to design the object. Having considered how gestures can be combined with physical objects to modify data, I next discuss the broader application of tangible objects to data representation.

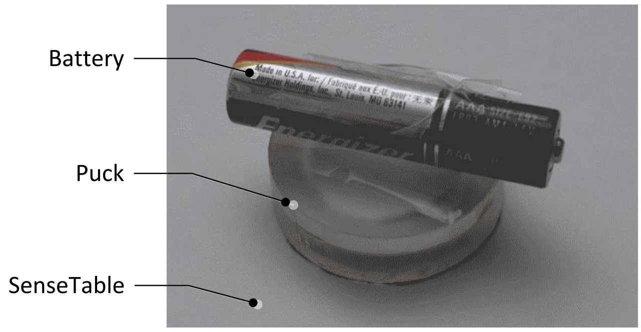


Figure 3-15 A battery serves as a mental reminder to its user

Based on Patten (2005)

3.2.5 Tangible objects and data

According to Holmquist, Redström and Ljungstrand (1999), the combination of tangible objects along with their relative positions convey meaning. I now explore this concept by considering the semantics of representational mappings, the use of tangible objects for generating data, the application of tangible objects for interacting with data, detection/measurement of data, and the manipulation of data.

Tangible objects can be used to interact with data and Holmquist et al. (1999) applies the following terminology to describe such tangible objects: *container* represents one or more datum, *token* describes an object that has a property that directly maps to data, and *tool* describes an object used to manipulate data. O'Malley and Fraser (2004) refer to these descriptions as the "semantics of the physical-digital representational mappings".

3.2.5.1 Tangible objects for generating data

In addition to tangible objects being metaphors of physical objects, they can also serve as metaphors of physical events and thus "generate" data. Underkoffler (1999b), Ishii (2008a, 2009), and Underkoffler and Ishii (1999) give examples of such metaphors in their Urban Planning Workbench (Urp). These are a "laser" object that generates a digital laser beam, a "wind tool" object that generates a digital wind, a "clock tool" object that sets the time of day in a digital model, and a "material wand" object that sets the "material" from which a digital model is constructed.

3.2.5.2 Tangible objects for interacting with data

According to Underkoffler (1999b), physical metaphors of mirrors, lenses, beam splitters, film, tweezers, wipers, and scrapers are examples of tangible objects that have either been applied to, or proposed for, affording interaction with data. In the following sections, I discuss systems that incorporate tangible objects that serve as data representations and systems that incorporate tangible objects with which to manipulate data.

Detecting/measuring data

Some tangible objects serve as metaphors of physical objects to "detect" and "measure" data. An example of such a tangible object is Ishii's (2009) "anemometer" that measures the projected "wind speed" of his table-top digital modelling system.

Manipulating data

Ishii (2009) states that Fitzmaurice et al. (1995) founded Tangible User Interfaces (TUI) research when they referred to their `Bricks` system as a *physical handle* with which to manipulate a virtual object. What follows are examples of systems that afford data manipulation.

Spatial compositions can be created by grouping two or more cubes, with Jacucci's (2007) `Videoblocks` system being an example. The system interprets the resultant spatial composition as an edited version of an original video. Jacucci uses the term *ordering by spatial association* to describe the composition creation process, and the terminology *special places* to describe locations in space where objects are interpreted. Here, the user is given a construction kit containing items such as wooden building blocks and modelling clay. This construction kit is part of the video-card authoring system. Jacucci, Jacucci, Wagner, and Psik (2005) and Jacucci, Pain and Lee (2006) explain how the users craft their own tangible objects to represent video scenes. She then arranges the objects to form a pathway through the arrangement. This pathway determines the composition of the edited video. Various video clip combinations are possible by varying the pathway. The researchers identified that the term *architectural patterns* describes the arrangement well and borrowed terminology from the field of architecture to label these arrangements. Examples include *museum format*, *town format*, *geometric format*, and *abstract format*. The museum format describes an arrangement in which the space is divided into themes, with each space having a single point of entry and exit, respectively. The town format describes a layout that resembles streets while the geometric or abstract format describes layouts such as circles and radial arms.

Another example of a system that relies on an arrangement of physical blocks is `Task Blocks` (Terry 2001). These blocks are arranged to form a pipeline sequence. Certain blocks manipulate the data as it passes through while others set parameters that determine how data are to be manipulated.

Ullmer's (2002) `mediaBlocks` are physical objects of arbitrary shape and size. Embedded circuitry uniquely identifies each object when electrical contact is made with an interrogation circuit. Digital media is not stored inside `mediaBlocks`; instead, Ullmer, Ishii and Glas (1998) explain that digital media is associated with the block by means of a block's identity. `mediaBlocks` serves as a mechanism to exchange data and also provide a physical reference to the data. When combined with physical constraints, `mediaBlocks` may be used to capture, retrieve, and edit digital media sequences. One `mediaBlocks` implementation is called the `mediaBlocks sequencer`. The `mediaBlocks slots` is another implementation that supports data transportation and the `media browser` device affords the browsing of the data represented by a `mediaBlock`. Five physical constraints are used in conjunction with mediaBlocks to form physical controls. Ullmer et al. (1998) refer to the constraints as *racks*, *stacks*, *chutes*, and *pads*.

By combining tangible disks with a *query* rack, Ullmer's (2002) system of parameter wheels and bars allows a user to construct database queries. Rotating the disk changes the value that it represents on the query rack. Physical bars can also represent database query parameters, with the distance between the bars setting the logical interpretation of the query parameters. A visual display provides user feedback.

The `Media Cubes` (Blackwell & Hague 2001a) programming language incorporates tangible cubes that are dynamically associated with appliances when these are placed next to each other. The user can then control the appliance by pressing a button on the cube.

The tangible interfaces in Patten, Recht, and Ishii (2002) and Patten et al.'s (2006) `Audiopad` is a set of circular pucks and four-sided *selector* pucks. The circular pucks represent data while the selector pucks help the user manipulate data. Both the puck position and orientation are tracked to provide representation and manipulation functionality. A puck is associated with data when it is placed on top of a data projection. The user manipulates the associated data by rotating the puck.

Camarata et al.'s (2002) Navigational Blocks queries a database and navigates the results. A query is compiled when two block types are manipulated. The first represents a time span (for example, "1850's to 1890's"). The second represents people (for example "founding fathers"). A block can be rotated to expose sides with various representations. A query is compiled by placing one of each block type next to the other thereby composing a query with time and person variables. Embedded electromagnets are system controlled, making it possible for one block to attract or repel another. These electromagnets also provide tactile user feedback. A third block type acts as a "host" and serves as a mechanism with which the user can navigate the search results. Navigation is possible by sliding the host block across the interaction surface, similar to using a computer mouse.

As far as artefact-representation relationships are concerned, Price (2008) reports that data can be exposed when objects are moved in relation to each other. Mazalek et al.'s (2002) Tangible Viewpoints demonstrates this method when they calculate the distance between tangible objects to determine what data to expose to the user. In general, Tangible Viewpoints retrieves data associated with a tangible object when the object is placed on an interaction surface. Multiple objects may be used simultaneously to retrieve data associated with each object. However, when two objects are in close proximity to each other the retrieved data is the data common to both objects.

Triangles (Gorbet & Orth 1997b) is a set of two-dimensional equilateral triangles that can be configured as either two or three dimensional structures. The triangular shape supports complex structures including decision trees. A decision tree is implemented by providing input on one side of a triangle and detecting the result at the other two sides. What makes Triangles particularly interesting to my study is that its designers considered the meaning embedded within objects and the meaning that emerges when these objects approach each other.

Tangible objects and their relative positions

According to Gorbet and Orth (1997b) and Marco (2011), the board games of chess, Monopoly, backgammon, dominoes, and checkers are examples of systems that embed meaning in the relative positions of the game pieces. In addition to these well-known systems, some computer-based tangible systems also rely on the relative position of the pieces. Three examples are Gorbet and Orth's (1997b) Triangles and their Digital Veil, and Ullmer's (2002) Parameter wheels and bars.

`Triangles` (Gorbet & Orth 1997b) exploits the meaning attached to objects by distributing images across triangles for the purpose of storytelling. The system produces a result when matching triangles are joined. According to Gorbet, Orth and Ishii (1998), the system's behaviour is determined by the proximity of the `TriMediaManager` to the `display triangle` as well as the sequence in which the remaining triangles have been joined. In Chapter 6, I explore inter object spacing and their sequence in a programming context.

Gorbet et al. (1998) and Gorbet and Orth (1997a) explain that the `Digital Veil` system user attaches personally meaningful narratives to a single or group of interlinked triangles. The narratives are retrieved and played back when the original triangle (or group of triangles) are attached either directly or indirectly to the display triangle.

As Ullmer (2002) explains, the `parameter wheels and bars` tangible system is used to compose database queries. In this system, physical bars represent database queries and the spacing between the bars represents Boolean operations such as logical `AND` and logical `OR`. The `AND` operation is implied when the bars are close together while the `OR` operation is assumed otherwise.

3.2.5.3 Tangible objects as containers of data

The application of bottles as containers and controls for digital data was co-invented by Ishii (2004). As described by Gorbet (1998), metaphorical objects such as *information eyedroppers*, *information tweezers*, and *information funnels* are used to "place" data inside such containers. The following paragraphs discuss systems that incorporate glass bottles.

The `musicBottles` (Ishii et al. 2001) system is an example of glass bottles used to contain data. Mazalek (2001) and Mazalek, Wood and Ishii's (2001) `genieBottles` not only contains data but also controls it. Mazalek explains that a spirit is "released" when the container is opened and multiple spirits "interact" with each other. Replacing the stopper on a bottle confines the spirit to the bottle.

Ishii, Wisneski, Brave, Dahley, Gorbet, Ullmer and Yarin's (1998) `ambientROOM` is another example of a system in which glass bottles contain data. When the bottle is uncorked digital content such as internet traffic is rendered as motorcar sounds. They elaborate that when the cork is removed the data is "set free" to "fill" the room.

In Lee, Vargas, Tang and Ishii's (2012) `rainBottles` system a glass bottle "fills up" with a virtual liquid that represents network data. The tangible water bottle in Underkoffler's (1999a, 1999b) data container implementation “holds” and “transports” data. The data “inside” is visualised when the bottle interacts with a video projection system.

Finally, Blackwell and Hague (2001a) explains that, within the ontological paradigm of the `Media Cubes` programming language, an open box serves as a data container. They call the box an `Aggregate Cube`.

3.2.5.4 Tangible objects as pointers to data

Ljungstrand, Björk and Falk (1999) and Ljungstrand and Holmquist (1999) describe by means of examples how tangible objects can point to data. The first is a coffee mug with a barcode sticker that directs the user to a particular web site. When the barcode is detected, the owner’s web browser opens at the site. The second example is a physical dictionary that is “tied” to an online encyclopaedia. The encyclopaedia is accessed when the physical dictionary does not provide satisfactory information.

3.2.5.5 Tangible objects as representations of data

Bishop’s (Moggridge 2006; Shaer, Leland, Calvillo-Gamez & Jacob 2004) `Marble Answering Machine` associates data (in the form of voice messages) with marbles. Metaphorically, the marbles “contain” the data but in reality serve as proxies for data residing within the answering machine.

The method followed in the `StoryBeads` (Reitsma 2011; Reitsma, Smith & Hoven 2013) system is similar to Bishop’s approach and incorporates hand-crafted beads to “contain” data in the form of stories. Reitsma’s beads serve as proxies for the data and the stories are not contained inside the beads but instead in a device called the *bead reader*.

Pedersen and Hornbæk ‘s (2009) `mixiTUI` uses *loop tokens* to represent sound clips and *effect tokens* to manipulate parameters. Effect tokens are dynamically associated with a loop token by proximity when the user places these on an interaction surface. A computer program associates loop and effect tokens with digital media.

Data can be associated with positions in a garment using the `Spyn` (Rosner & Ryokai 2009, 2010) system that incorporates a specially prepared yarn. As knitting progresses, the crafter decides what data to associate with selected positions in the garment. Then, once the garment is completed, the user can retrieve the data “captured” within the garment.

Marco’s (2011) `NIKVision` tabletop and `ToyVision` (Marco, Cerezo, Baldasarri, Mazzone & Read 2009; Marco, Cerezo & Baldassarri 2012) systems allow the user to apply existing and custom crafted tangible objects as tokens to represent and manipulate data. User feedback is either by bottom-up projection onto the interaction surface or a separate computer display. These systems are based on Kaltenbrunner's (2009) `reacTIVision Toolkit` and can be programmed to respond when tangible objects are in close proximity to each other.

3.2.6 Tangible objects and digital models

Some systems arrange tangible objects to form shapes that are then interpreted as digital models. Examples of physical tools that rely on their spatial properties to capture digital models are `ActiveCube` (Jacoby et al. 2006; Kitamura et al. 2001; Watanabe, Itoh, Asai, Kitamura, Kishino & Kikuchi 2004a, 2004b; Watanabe, Itoh, Kawai, Kitamura, Kishino & Kikuchi 2004c), `Cognitive Cubes` (Sharlin, Itoh, Watson, Kitamura, Sutphen & Liu 2002), `Computational Building Blocks` (Anderson et al. 2000), `Universal Constructor` (Frazer 1995), and `MERL Block` (Aish, Frankel, Frazer & Patera 2001). What sets these systems apart from others already discussed is that the shapes are not limited to two dimensions. The ability to capture three-dimensional shapes is due to their mechanical interlocking design. The computer analyses the data path that results and from this constructs a corresponding virtual model. Although programs that take form as three-dimensional shapes are interesting, I limit my study to planar constructions and do not consider these systems further.

3.2.7 Conclusion to this section

In this section, I highlighted that Ishii's(2009) basic tangible interface model does not address the origins of the object. My research addresses the object origin by focussing on the personal meaning the object holds. This section thus focussed on the meaning of objects and how these interface to the computer. I identified three categories of meanings to add to those already published by Underkoffler and Ishii (1999) and these are an object as a quantity, an adjective, and a verb. I then considered the technologies that connect objects to the computer and classified these according to how data are exchanged between the system components and how the identity and position of the object are encoded. My classification scheme identified two prominent categories in tangible systems and these are what I call untethered passive tangible systems and untethered active tangible systems. I next addressed gesture modalities for manipulating data and identified that the most prevalent measurement methods are based on the proximity between objects or the distance between an object and a specific position on an interaction surface. Having now considered the meaning an object holds, I next investigate how these objects are created.

3.3 User-created tangible objects

Krippendorff (1989) discussed the relationship that exists between form and function. He considered the relevance of the slogan “*form follows function*” (an approach that abstracts the user out of the design process and disregards the meaning that a user attaches to the form) and argued that this slogan was, at the time, no longer the best approach to design. Instead, Krippendorff proposed “*form follows meaning*” as an alternative design process. Using this approach the user participates in the design process so that the meaning assigned by the user is captured in the final design.

He considered form to be an objective description of something that does not consider the user. In contrast to the objective description, meaning always considers the user. In addition, the user makes sense of the artefact within the context it is considered. As Krippendorff’s model in Figure 3-16 illustrates, an artefact has both form and meaning. According to this model, the designer primarily considers the form of the artefact whereas the user is concerned with its meaning. In my research, I consider the scenario where an individual is both the artefact designer and user.

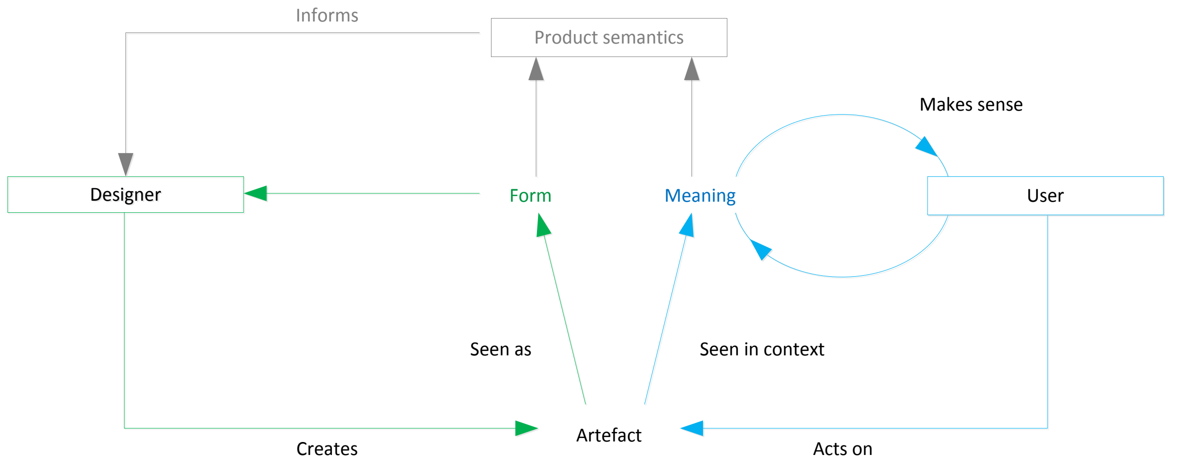


Figure 3-16 Krippendorff’s model of the user and the designer’s view of an artefact

(Krippendorff 1989)

I propose that when the artefact designer and user are the same person, then Krippendorff’s model can be adapted as shown in Figure 3-17.

In the adjusted model, the user views the artefact as a form and makes sense of the form. The result is a form that holds meaning for the user. Because the user can also create the artefact, he can alter the form by either changing the current form or by creating a new one. This ability to alter the form can be used to the advantage of the user because he can make minor adjustments to the form over time, thereby refining and adjusting it to better fit the meaning that he attaches to the artefact.

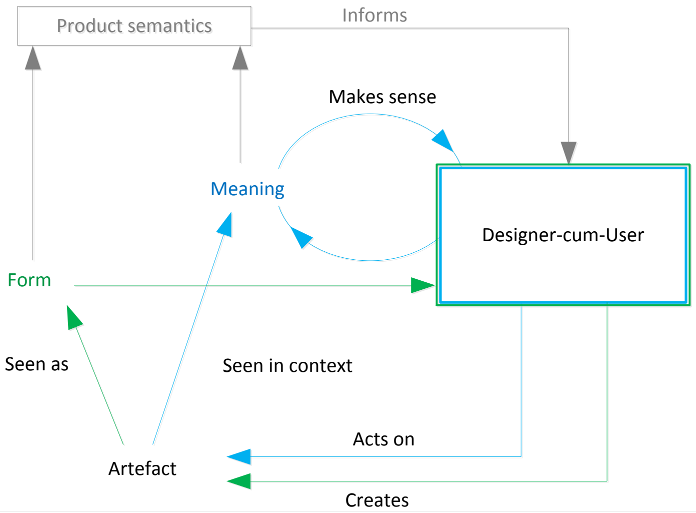


Figure 3-17 Krippendorff's adapted model reflects the case where and individual is both the designer and the user

Derived from Krippendorff (1989)

3.3.1 Motivation for user-created tangible objects

The meaning that individuals attach to objects may differ from one person to the next and the meaning may be influenced by, for example, the point in the object's life cycle at which the person first encountered the object (Boradkar 2010). According to Simmel (2004), the value that two individuals attach to the same object may also differ. I therefore argue that it is not only the object's own life cycle that influences the attached meaning but also the point in an individual's life at which the object is encountered. An example of this phenomenon is the meaning that an adult attaches to a banknote compared to the meaning an infant attaches to it. In this example, the adult attaches a meaning of prosperity to the banknote whereas the infant views it as a play object.

According to Fischer and Scharff (2000) and Nardi (1993) it is not possible for the designer to determine all the user's requirements in advance. The result is a design that lacks certain desired properties. Keeping in mind that the designer is often physically, cognitively, or emotionally distanced from the target user it can be argued that the meaning that the designer attaches to an object may not be the same as the meaning that the user attaches to it. Krippendorff (1989) offers as an example the design (and use) of a motorcar and highlights the varied meanings that the designer and the user assign to the same artefact: Whereas the designer assigns a meaning of transportation to the car the user may view the car as a status symbol. The result is that the vehicle has two meanings simultaneously attached to it. It is therefore interesting to consider systems in which the individual is not only the user of an object but also the designer and creator of the object.

As alluded to in the preceding paragraph, it is at times the responsibility of the user to define what the tangible interface to a system should look like and then create the tangible interface. When the interface design responsibility shifts from the designer to the user the result is an *improvisational* *interface* (Patten 2005). In creating an improvisational interface, the interface designer is no longer responsible for incorporating physical affordances and metaphors into the interface. Instead, the user is empowered to incorporate materials at hand and change the interface to improve it. An example of an interface designed to accommodate the user's preference is the hole in the common household key. With this design, the user is free to choose from a host of options when deciding what to attach to the key. As shown in Figure 3-18, options include a functional metal wire loop with no personal significance, personally meaningful memorabilia, and a motivational tag.

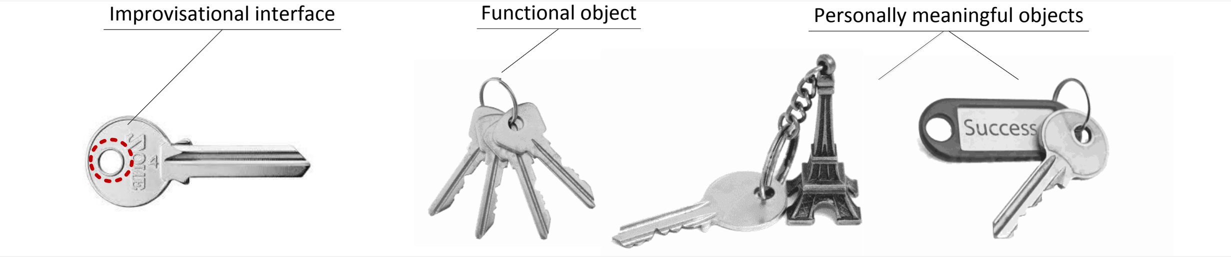


Figure 3-18 An improvisational interface

In Chapter 6, I discuss my T-logo programming environment in which I have defined a set of digital program elements but have left the physical representation to the user. I have also selected markers to represent each program element. The marker serves as an improvisational interface to which the user may attach a personally meaningful object to remind him of its function as a program element. Figure 3-19 illustrates six examples of what the user may construct to tangibly represent a program element that depicts the concept of speed. The user establishes a physical representation of the program element by attaching the assigned marker to the object of choice.

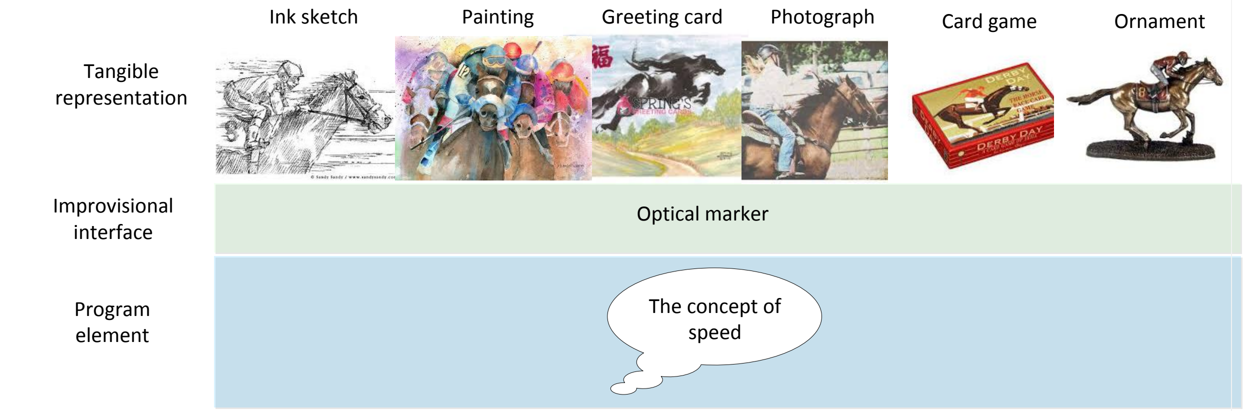


Figure 3-19 A marker as an improvisational program element interface

3.3.2 Craft materials for, and methods of, constructing tangible objects

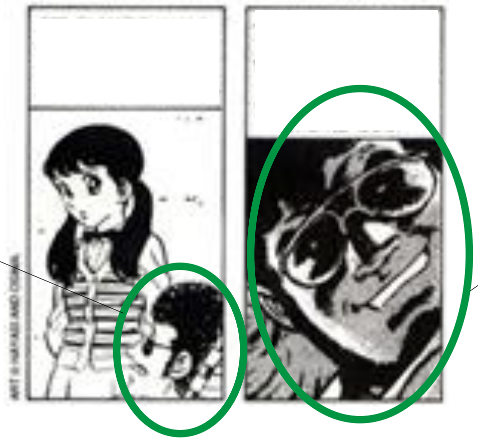
Ullmer (2002) states that a strong association exists between a tangible interface and the physical object that embodies it. Grounded in this statement, he advocates for artefact designs that are aesthetically plausible to the user. My research builds on Ullmer's assertion and gives the user an opportunity to construct the program elements of a tangible programming environment.

He investigated tools for creating tangible interfaces, including objects that are in everyday use (a key ring is an example), construction kits (such as LEGO (Griffin 2010)), handcrafting material like cardboard, traditional professional prototyping methods such as machining, and more recent prototyping methods that include laser cutting and 3D printing. According to Ullmer, the user faces two challenges when everyday objects are appropriated as tangible interfaces. The first is that the user may find it difficult to grasp that one object can have multiple meanings according to the context in which it is used. However, I am of the opinion that Ullmer's concern will dissipate if the user is given the opportunity to appropriate everyday objects as tangible interfaces.

The second challenge relates to the engineering effort required to integrate the everyday object with the system in which it will be used. As far as using construction kits as tangible interfaces is concerned, Ullmer concedes that these are useful for rapid prototyping yet have limited potential once the intended tangible interface concept has been demonstrated. He found the tangible interfaces crafted by hand to lack mechanical stability and precision if compared to traditional professional prototyping methods such as machining. Ullmer is of the opinion that these methods can be time consuming and do not support rapid design iterations. He however considers recent prototyping methods such as laser cutting and 3D printing to be useful in creating tangible interfaces due to the ease with which the design can be modified and the process repeated.

3.3.3 Environments for creating personally meaningful tangible objects

McCloud (1994) states that simple visual character designs are sometimes favoured over realistic designs. Figure 3-20 depicts two designs in adjacent panels that support the following discussion.



A "simple" character design supports observer self-identification well.

A "realistic" character design objectifies the character.

Figure 3-20 A simple character design supports observer self-identification better than a realistic character design

Based on McCloud (1994)

The absence of strong character-forming features in the simple design on the left allows the observer to identify with the character. This design can be compared to a blank canvas to which the observer applies imaginary markings so-as to reflect his personalised character. In contrast, a realistic character design on the right is rich in character-forming features that objectify the character; in other words, the character has its own identity that is separate from that of the observer.

In theatre, performers wear masks to express a particular character. A neutral mask (Figure 3-21) can be used for the opposite effect, that is, to depersonalise the performer by not forcing a particular character but rather let the character develop as the performance progresses (Jacucci 2007). From this, I deduce that if the system designer's objective were for the system user to create personalised artefacts, it would be appropriate to provide the user with materials that have little intrinsic "character". I am of the opinion that Sanders (2000) and Sherman et al. (2001) had this in mind when they designed paper and crayons into their toolkits. I will now discuss their research.

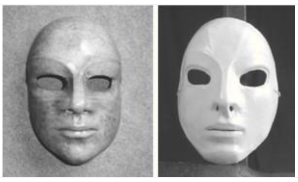


Figure 3-21 Masks with little intrinsic character

(Jacucci 2007)

In applying McCloud's view on character design to artefacts it is plausible that an individual will best identify with an artefact that has few features. Sanders (2000) aligned with McCloud's distinction between simple and realistic character design when he assembled his workshop toolkit. Using the toolkit, a participant can create an artefact to express an idea. According to Sanders, the toolkit was purposefully designed to include a range of simple and ambiguous parts so that the user can create an artefact that reflects his personal aspirations. He calls this the *projective quality* of the toolkit. It consists of tangible visual components such as stickers, photographs, sketches, coloured paper of various shapes, and other spatial forms. Hook-and-loop pieces on spatial forms support quick and simple artefact extensions. Markers and crayons are also included in this toolkit. Instructions to the workshop participants are simple with the only requirement being that the crafted artefact must hold personal meaning and that the artefact must express the way the participant feels about a given experience.

In contrast to Sanders' passive two dimensional artefact toolkit, `StoryKits` (Sherman et al. 2001) is a collection of high-tech and low-tech objects that can be combined to create tangible objects. High-tech items include sensors and actuators such as an electrical switch, radio frequency identification (RFID) technologies, a loudspeaker, and a light emitting diode (LED). Low-tech items include crayons, paper, boxes, and glue. Sherman et al. refer to the created objects as *physical icons* because these represent the task performed in the `StoryRoom` (Alborzi et al. 2000) programming environment. Figure 3-22 shows two objects used in the `StoryRoom` and these are a camera model constructed using low-tech materials and a high-tech RFID system item that can trigger the model. I elaborate on the StoryRoom in Chapter 4.

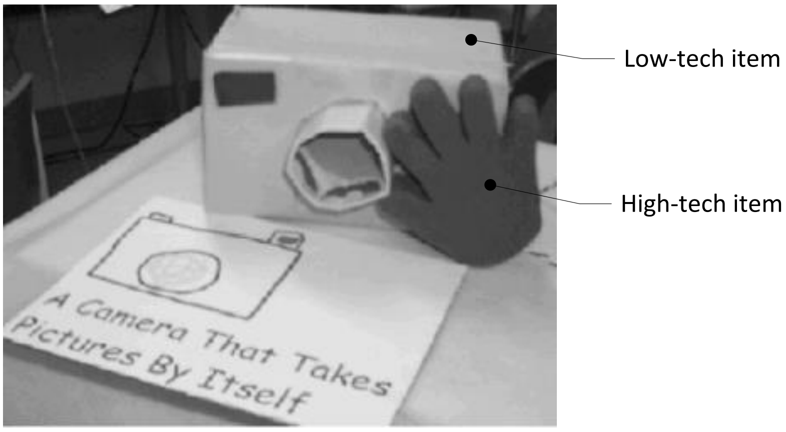


Figure 3-22 StoryKits integrates high-tech and low-tech items
Based on Montemayor, Druin, Farber, Simms, Churaman and D'Amour (2002) and Sherman et al. (2001)

Quilt Snaps (Buechley, Elumeze & Eisenberg 2006), Spyn (Rosner & Ryokai 2008, 2009, 2010), and RockBlocks (Smith 2009a, 2009e) are further examples of user-crafted artefacts made according to personal preference. These artefacts respectively incorporate the crafts of sewing, knitting, and sculpting. I chose construction materials in my T-logo programming environment based on McCloud's observation that the absence of strong features makes it easier for an individual to form personally meaningful objects. Chapter 6 discusses the materials used.

3.3.4 Conclusion to this section

This section considered the origins of tangible objects as applied to computer interaction. I proposed a modified Krippendorff model to reflect the scenario when an individual is both the designer and user. I also demonstrated that the improvisational interface is appropriate when personally meaningful objects are crafted. Based on McCloud's discussion regarding simple character designs in book drawings, I identified that researchers apply similar criteria when selecting construction materials for research on user-created objects.

3.4 Discussion

Ishii's (2009) basic TUI model describes how a user can apply a physical object to interact with data. However, Ishii does not include the origin of the object in his model and this is a problem since, according to Krippendorf (1989), the meaning that an object designer attaches to the object may not be the same as that of the user. Underkoffler and Ishii (1999) support Krippendorf's view. Krippendorf therefore advocates that meaning should be embedded within an object instead of designing the object only according to the function it will serve. However, since users do not always associate the same meaning to an object (Simmel 2004), a single object design will not suit multiple individuals. There is thus a need for individualised objects. Yet, it is not always possible for the object's designer to anticipate who the users will be (Fischer & Scharff 2000; Nardi 1993) and it is therefore worthwhile to design systems with improvisational interfaces (Patten 2005) that allow the user to customise the object.

Having now motivated for the design of systems that will allow the user to create personally meaningful objects, the next problem to consider is what materials to provide to the user. The problem lies in that these can already hold meaning. It is for this reason that McCloud (1994) and Jacucci (2007) explain that in order for the user to assign her own meaning to an object, it is best to use objects that do not already include strong meaning forming elements. This may explain why Jacucci et al. (2005, 2006), Alborzi et al. (2000), Buechley, Elumeze and Eisenberg (2006), Sanders (2000), Rosner and Ryokai (2008, 2009, 2010), and Sherman et al. (2001) supply to their users meaning-neutral resources including crayons, wood, clay, cloth, wool, paper, and glue .

3.5 Conclusion

This chapter focussed on tangible objects, how these are used, and where they originate. I adapted Krippendorff's object creation model to reflect the case where the designer and user is the same person. I then reported on literature that describes instances where the user either modifies or creates objects that hold personal meaning. The materials and mechanisms employed to this end were also discussed. Having now considered user-created tangible objects and how these are used for computer interaction, Chapter 4 will focus on tangible objects that serve as program elements and those programming environments that incorporate user-created objects.

CHAPTER 4
LITERATURE REVIEW: TANGIBLE PROGRAMMING

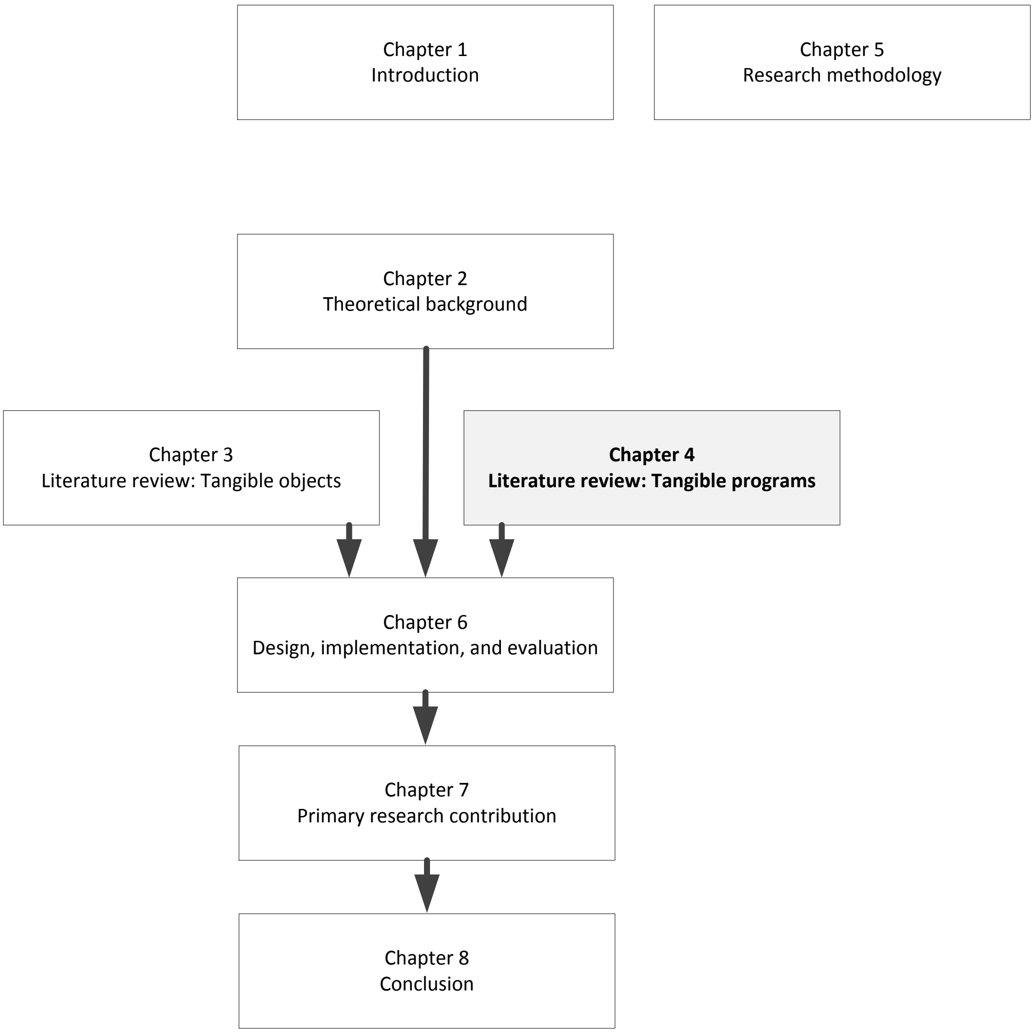

Chapter 1
Introduction
Chapter 5
Research methodology
Chapter 2
Theoretical background
Chapter 3
Literature review: Tangible objects
Chapter 4
Literature review: Tangible programs
Chapter 6
Design, implementation, and evaluation
Chapter 7
Primary research contribution
Chapter 8
Conclusion

Figure 4-1 Document structure

4.1 Introduction

Chapter 2 covered the theoretical background to this study by considering how programs are used to control the behaviour of computers thereby making the general-purpose computer a task-specific instrument. That chapter also discussed the meaning signs hold for humans (semiotics) and that programs serve as sign containers. Fundamental to semiotics is the way in which the brain perceptually organises stimuli. Signs will not be recognised and the stimuli will hold no value if the received stimuli is not properly organised. Gestaltists have identified principles that describe this organisation and I explored some of these.

In Chapter 3, I considered how tangible objects are used to interact with the computer. I also presented Ishii's (2009) model and identified that his model does not include the origin of physical objects. Also covered is how the object designer's view differs from the user's view of the same object. I proposed an adapted version of Krippendorff's (1989) model in which the designer and the user are the same person thereby eliminating possible misunderstanding between the two parties as far as the significance of the object is concerned.

Programming, as discussed in Chapter 2, is mostly based on text and graphics. Some researchers have developed physical signs that form programs while other researchers include the user in the design of the signs. However, the design of signs by explicitly considering Gestalt principles of visual organisation remains unexplored.

Having discussed in preceding chapters how computer programming incorporates signs, how the formation of signs can be expressed by certain Gestalt principles, and how physical objects can serve as signs in general, I now consider those environments in which the program consists of physical signs. Therefore, the objectives of this chapter are twofold. First, to identify literature that reports on tangible object programming environments. Second, to investigate literature where the user is the creator of the objects that comprise a tangible program.

Section 2 considers the difference between physical and tangible environments and I offer definitions for their respective programming environments. Section 3 debates tangible programming environments, making a distinction between environments in which the user is given objects and environments in which the user designs and constructs objects. The discourse also covers the Gestalt principles of perceptual grouping. In Section 4, I discuss Gestalt principles present in tangible programming systems and I suggest how personally meaningful objects can be included in systems not designed for this purpose. Section 5 concludes this chapter.

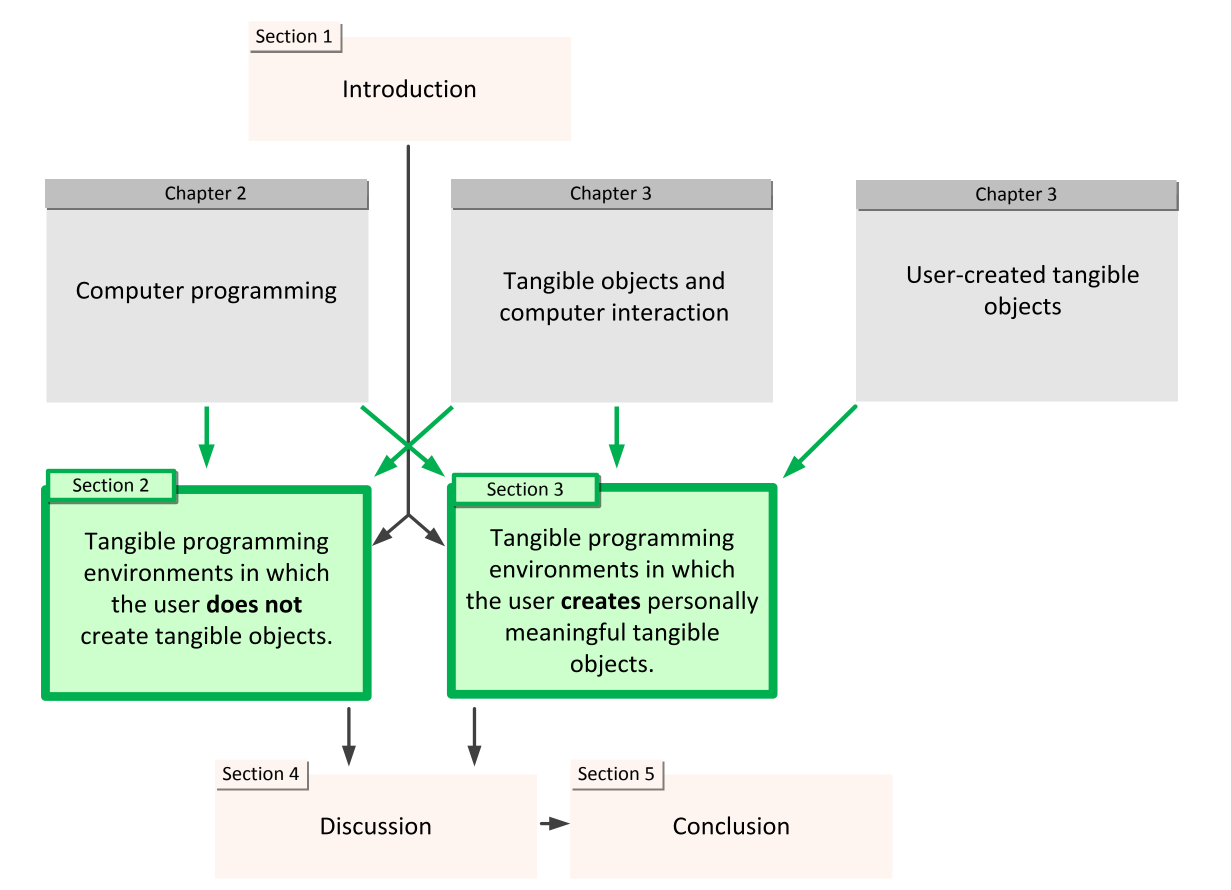

Section 1
Introduction
Chapter 2
Computer programming
Chapter 3
Tangible objects and computer interaction
Chapter 3
User-created tangible objects
Section 2
Tangible programming environments in which the user **does not** create tangible objects.
Section 3
Tangible programming environments in which the user **creates** personally meaningful tangible objects.
Section 4
Discussion
Section 5
Conclusion

Figure 4-2 Chapter outline

4.2 Distinguishing between physical and tangible programming

Tangible programming refers to the act of manipulating graspable objects with the purpose of creating a structure or an arrangement that will be interpreted as a program. Such computer programs take form as structures or arrangements of physical elements. Systems developed by Bers and Horn (2010) and McNerney (1999) are examples of structures and arrangements that serve as programs. The act of creating a program this way has also been referred to as physical programming (Mukherjee, Sharma & Prakash 2002).

Even though the terminology tangible programming and physical programming are used interchangeably, I view them as distinctly separate and different activities. I consider the term tangible programming most fitting to this study and therefore limit my discussion on physical programming to Section 4.2.1 and elaborate on tangible programming in Section 4.2.2.

4.2.1 Physical programming

Physical programming has been described as the act of defining the interactions between objects within a room (Sherman et al. 2001). In turn, Montemayor et al. (2002) defines physical programming as "*the creation of computer programs by physically manipulating computationally* *augmented (or aware) objects in a ubiquitous computing environment*". Montemayor et al.'s definition has been applied to a programming environment that comprised of a room in which objects were scattered and programming was done by demonstrating what actions must be taken when an event occurs (also called *programming-by-demonstration*). Based on Montemayor et al.'s definition of physical programming, I define a physical programming environment as follows:

> **Physical Programming Environment**
> A physical programming environment is an omnipresent computing environment in which computationally augmented or aware objects are manipulated to create a program.

4.2.2 Tangible programming

The term *tangible programming* was coined in 1994 to describe the AlgoBlock system (McNerney 2004). For the purpose of this study, I will use tangible programming to describe the activity of constructing a program consisting of physical objects that can be grasped by hand and manipulated using the fingers. Used in this way, tangible programming excludes virtual objects such as visual symbols rendered on a computer screen.

Suzuki and Kato (1995b) describe a *tangible programming language* as being a program language with physical presence. A tangible programming language also serves as a mechanism with which to instruct a computer to take action (Bers & Horn 2010) and is comprised of elements. Wyeth & Purchase (2002) use the term *tangible programming elements* to describe their stackable blocks that both form the computing structures and interact with the physical world.

According to McNerney (1999), the origins of tangible programming remains unresolved and it has been suggested that Perlman's (1976) TORTIS system's Slot Machine was the first implementation of a tangible programming language. In Chapter 6, I use the terminology tangible programming and tangible program elements to discuss my tangible programming environments.

4.2.2.1 The tangible programming environment

I base my distinction between physical and tangible programming environments on three concepts. The first is what McNerney (1999) calls the *graspable property* of a tangible programming environment. The second is the nature of the (at times immovable) objects in the physical programming environment, and Montemayor et al.'s (2002) *ubiquitous computing environment in the third*. Based on these concepts, I now define a tangible programming environment as follows:

> **Tangible Programming Environment**
> A tangible programming environment is a computing environment in which a static arrangement of graspable objects constitute the program.

4.2.2.2 Useful characteristics of tangible programs

Using tangible objects as a computer program presents interesting characteristics, with some already explored by Fernaeus and Tholander (2006). Two characteristics of particular relevance to my study are the persistence of physical code and how humans interpret it.

Some tangible programming systems reuse the same object to create a program, yet others such as AlgoBlock (Suzuki & Kato 1994) dedicate a particular object to a specific position in the program. In the case where an object is dedicated to a specific position within the program the result is a program composed of physical artefacts (Suzuki & Kato 1995b) and the composition is referred to as *graspable software* (Montemayor 2001). To the user, a tangible program takes form as a collection of physical objects (Fernaeus & Tholander 2006) and often on a flat surface such as a table top. In contrast to text and graphic programs, the user is able to view the complete program at a glance without relying on technology to interpret an intangible program and render it as symbols on a video monitor.

In this study, I am interested in applying personally meaningful tangible objects as program elements that remain in place once the program has been constructed. Chapter 6 explores this concept further where I discuss the evolution of my T-logo programming environment.

4.2.2.3 Tangible programming styles

McNerney (1999) discusses programming styles that have successfully been applied to tangible programming environments. These include the imperative, functional, rule-based, behaviour mixing (with priorities), and database query programming styles.

A program constructed in the imperative style is evaluated sequentially with only one instruction being executed at a time (McNerney 1999, 2004). The program also has states and side effects due to its modifiable variables, data structures, and objects. The application of this style results in an algorithm that provides systematic instructions (also known as *imperatives*) for execution.

In contrast, a program constructed in the functional style is not evaluated sequentially, nor has side effects (McNerney 1999). When applying this style, the user only states what is to be computed but not how it should be done. An electronic spreadsheet is an example of a system that applies this style (McNerney 2004).

A program constructed in the rule-based style consists of separate rules, each of which can be modified independently of the others. McNerney (McNerney 1999) suggests that the rule-based style is suitable to tangible programming environments. I explore this style further in Chapter 6.

The behaviour mixing (with priorities) tangible programming style combines various behaviours and these are prioritised as part of the program definition. An arguable example of a tangible programming system using this style is Schweikardt and Gross's (2006) roBlocks system as discussed in Section 4.3.1.3.

Finally, an example of a program constructed in the tangible database query style consists of stacks of physical query parameters: A single stack represents the logical AND statement in which all the conditions in the particular stack are evaluated as a group whereas multiple linearly aligned stacks represent the logical OR operation. Figure 4-3 illustrates this concept by means of a database query that considers eye colour, hair colour, and height.

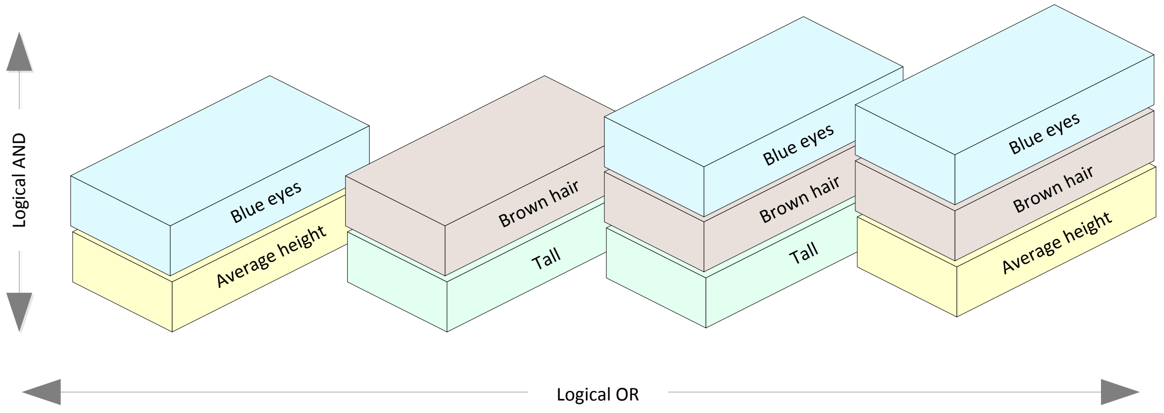


Figure 4-3 An example of a tangible database query

Based on McNerney (1999)

In Chapter 6, I continue to explore groups of tangible program elements by considering certain Gestalt principles of perception. For example, I propose that the stacks in Figure 4-3 need not lie along an imaginary line and that it is sufficient for the stacks to be perceptually grouped as Figure 4-4 illustrates.

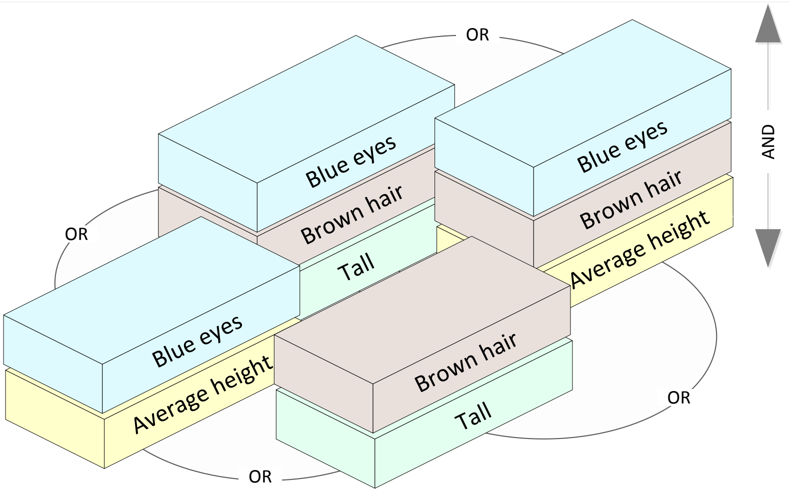


Figure 4-4 The Gestalt principle of perceptual grouping applied to the tangible database query style

4.2.2.4 Tangible programming modalities

Lund (2003, 2004) describes *programming-by-building* as the assembling of discreet tangible blocks into a single machine. In such systems, the three-dimensional topology of linked objects defines the program. These links are either mechanical or inferred. An example of a programming-by-building system with mechanical links is Ngo and Lund's (2004) I-BLOCKS.

The term tangible *programming-by-example* describes a programming modality by which the user manipulates tangible objects and instructs the system to store this interaction sequence. When using this modality, the user manipulates the objects whilst the system simultaneously monitors the manipulation sequence and object topology. This modality is also referred to as *programming by tangible demonstration* (Frei, Su, Mikhak & Ishii 2000), *programming-by-demonstration* (Knoll, Weis, Ulbrich & Brändle 2006), and *gestural programming-by-example*.

Some systems that incorporate the programming-by-example modality can animate user-generated manipulation and I call these *kinetic-kinetic* systems. Topobo (Raffle, Ishii & Yip 2007; Raffle, Parkes, Ishii & Lifton 2006; Raffle, Parkes & Hiroshi 2004; Raffle 2008) is an example and includes objects with embedded sensors and actuators. Using Topobo, the user creates a program by interconnecting the objects and physically manipulating them. This system then re-enacts the movements when instructed. Chung, Shilman, Merrill, and Ishii (2010) states that the kinetic-kinetic programming modality was pioneered by the Topobo project. However, this statement does not consider the CurlyBot project that predates Topobo. To program CurlyBot, a user gestures motions while simultaneously pressing the *record* button. Programmed motions are later re-enacted (Frei et al. 2000) when activated.

Certain programming-by-example systems do not animate physically and I refer to these as *kinetic-passive* systems. Chung et al.'s (2010) OnObject is an example of a kinetic-passive system with which the user associates auditory responses to his manipulations. The associated auditory responses are later replayed when the manipulations are repeated. Active Surfaces (Grönvall, Marti, Pollini & Rullo 2006) and StoryRooms (Alborzi et al. 2000; Montemayor, Druin, Chipman, Farber & Guha 2004; Montemayor et al. 2002) combined with StoryKit (Sherman et al. 2001) objects are two more examples of kinetic-passive systems.

4.2.3 Conclusion to this section

In this section, I made a distinction between physical and tangible programming by highlighting the differentiating aspects of each: First, physical programming requires a computationally ubiquitous environment whereas in a tangible programming environment this is not a requirement; instead, it is only the tangible object that requires computational abilities. Second, a physical program includes immovable objects whereas tangible programming objects can be grasped and manipulated by hand.

The following section focusses on existing tangible programming environments. In particular, it distinguishes between those environments in which programming objects are given to the user (Section 4.3.1) and those in which the user designs personally meaningful objects (Section 4.3.2).

4.3 Tangible programming environments

Having discussed the general concepts of programming and programmable systems in Chapter 2, I now re-examine these concepts by focussing on environments in which tangible objects define a program.

4.3.1 Environments that incorporate supplied objects

In this section, I discuss the group of tangible programming environments in which the user incorporates existing objects in the construction of a program. I categorise such environments according to the number of dimensions required to describe the program.

4.3.1.1 Objects utilised for their one dimensional property

All physical objects occupy space in three dimensions yet a single dimension sufficiently describes the properties of some objects. The dressmaker’s tape is an example of such an object with only the distance between the markings a being relevant. When in use, the tape does not have to lie along a straight line and may follow the contours of the object measured. Similarly, a single dimension sufficiently describes some tangible programs. What follows are descriptions of programming systems in which a single dimension is sufficient to describe the program.

- **Physical macros**

De Guzman and Hsieh’s (2003) physical macros system is a collection of tangible cardboard rectangles that are arranged to manipulate digital images. These transformations include rotating, horizontal and vertical flipping, resizing, and changing the colour depth of the image. To use this system, the user chains physical icons (also called phicons as already discussed in Chapter 2) together and optionally replaces this chain with a single phicon. The approach to replace a group of phicons with a single phicon (as is done in this system) is particularly appropriate to programming environments in which the elements that comprise the program (such as symbols on a computer display, or tangible objects on a horizontal surface) remain in place when the program is complete. This is because better use is made of the available programming area when a single representative phicon replaces multiple phicons. This approach is similar to the gather gesture applied in the Siftables (Merrill et al. 2007) interaction language already discussed in Chapter 3. Oh et al.’s (2013) instance block is another example of how a single phicon can be applied as a substitute for multiple objects. In Chapter 6, I further consider phicon grouping by highlighting the Gestalt principle of grouping by proximity.

Table 4-1 Summary table of physical

Programming style	Programming modality	Persistent program?	Gestalt principles	Number of physical program dimensions	Functionality
Imperative	Programming-by-building	Yes	Good continuation	1	Image editing

- **Physical strings**

Patten, Griffith, and Ishii's (2000) physical strings programming system is based on the way physical strings are configured to connect events with actions. Timers, light sensors, and pressure sensors generate events. Actions can pause program execution and control the behaviour of a robot such as forward and backward movements and turning. A program is constructed when the user maps events to actions by attaching one end to a board listing all expected events and the other end to a second board with all possible actions. Actions may be strung together so that one action flows into another. Program statements are executed concurrently when a single event is connected to multiple actions in parallel. Although not considered by the authors, it is conceivable for the user to customise the appearance of the string and thereby signify the logical relationship that exists between an event and the programmed action. An example is to attach a written note or diagram that depicts the cause and effect rule to the string.

Inspired by Patten et al.'s (2000) physical strings programming system, I present in Figure 4-5 a tangible programming concept in which the Gestalt principles of good continuation and perceptual grouping by proximity are explicitly applied. At the top in this figure, six artefacts represent actions and one artefact represents an event. The bottom portion of this figure shows the same objects but now arranged according to two Gestalt principles. The first principle perceived on the left is grouping by proximity and this is due to the close arrangement of the yellow, orange, and blue artefacts. The second Gestalt principle is that of good continuation. Two instances of good continuation are evident in this figure with an imaginary line seemingly stretching from the left to the top right and another from the left to the bottom right. For the observer it is as-if the two imaginary lines stem from the yellow "event" artefact. When considered in the context of a program, the artefacts that lie along these imaginary lines represent a sequence of program instructions. The overall effect is that an event simultaneously triggers instruction sequences A and B.

In Chapter 6, I extend this concept by providing the user with an opportunity to customise artefacts that represent events and actions. I also explore how the simultaneous application of the Gestalt principles of good continuation and grouping by proximity link events with actions.

Table 4-2 Summary table of physical strings

Programming style	Programming modality	Persistent program?	Gestalt principles	Number of physical program dimensions	Functionality
Imperative	Programming-by-building	Yes	Good continuation	1	Image transformations

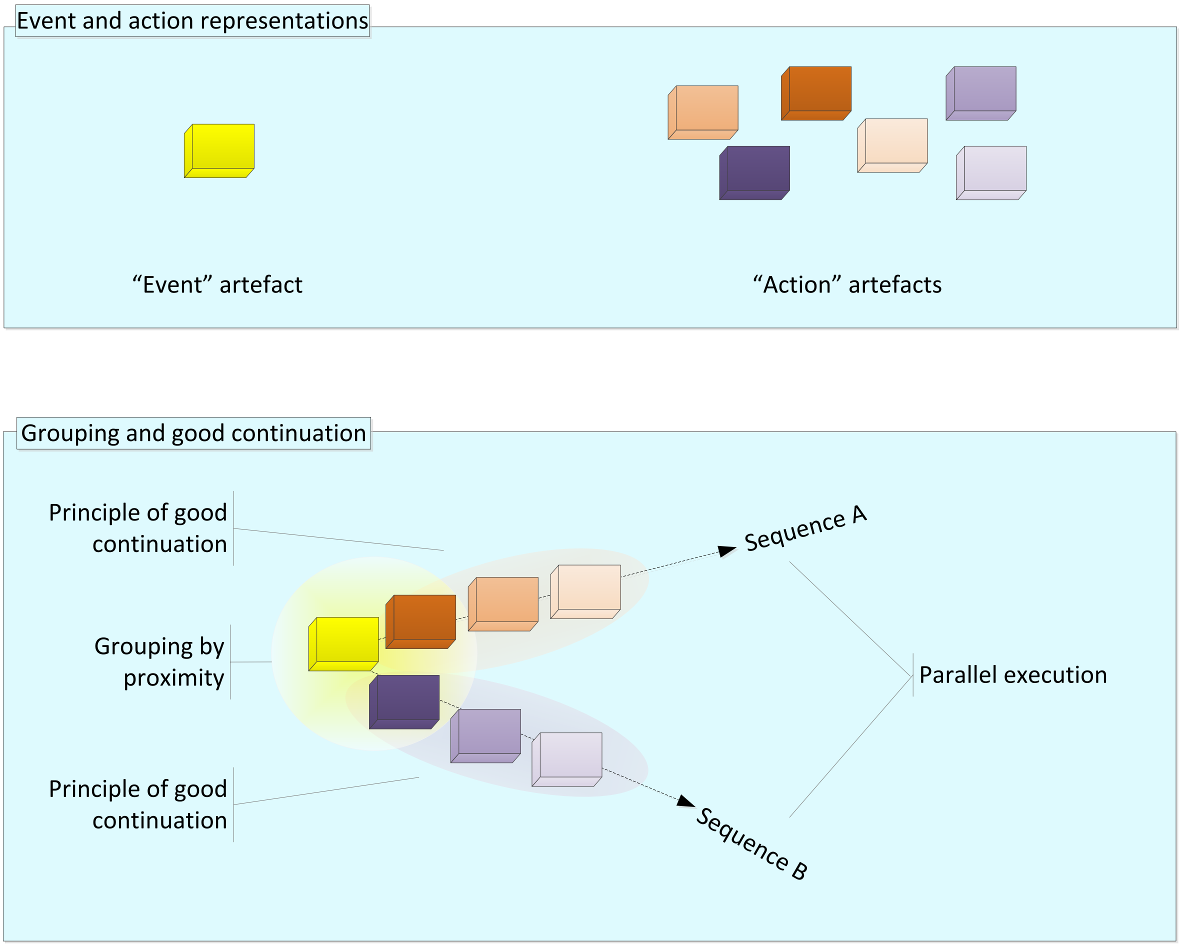

Event and action representations
"Event" artefact
"Action" artefacts
Grouping and good continuation
Principle of good continuation
Grouping by proximity
Principle of good continuation
Sequence A
Sequence B
Parallel execution

Figure 4-5 Gestalt principles applied in an arrangement

- **Electronic Blocks**

Electronic Blocks (Wyeth & Wyeth 2001) is a set of interlocking plastic blocks with which the user constructs a program by stacking the blocks. The user identifies a block's function by either memorising the function according to the block colour, by interpreting the sign on the block side, or by inferring the function based on the protrusions. Yellow sensing blocks incorporate icons that indicate their sensing modalities. For example, an eye icon represents a block that can "see" and the

＆ symbol represents the logic AND operation. An action block produces a physical output with light and sound being two examples of this. A movement block is equipped with four wheels and an electric motor. Logic blocks perform computational operations based on their inputs and include logical NOT, AND, signal inversion, and signal delay. Again, attached symbols indicate the function of a logic block; for example, the ＆ symbol represents the logic AND operation (Wyeth 2008).

In Chapter 6, I explore the design of an environment in which program element functionality is encoded in a personally meaningful manner. In such an environment personally meaningful program elements is the result of the user choosing the appearance and construction material.

Table 4-3 Summary table of Electronic Blocks

Programming style	Programming modality	Persistent program?	Gestalt principles	Number of physical program dimensions	Functionality
Imperative	Programming-by-building	Yes	Good continuation	1	Mechanical animation

- **TORTIS**

Although primary sources are limited, good secondary sources such as Kahn (1996), Morgado, Cruz and Kahn (2006), and Jetsu (2008) describe Perlman's (1974, 1976) TORTIS slot machine component design and operation. To create a program, the user places cards sequentially along a horizontal slot and the result is interpreted from left to right.

The user can also program subroutines by adding a directive card to the sequence. This card is of uniform colour and directs program execution to another slot where a matching card in the left-most position indicates the continuation point. Control returns to the first sequence when the second sequence ends. Although not explicitly stated by Perlman, the combination of mechanical constraints and the Gestalt principle of good continuation serve to guide the user in program construction. I incorporate the Gestalt principle of good continuation in all my design iterations as discussed in Chapter 6. My final design completely negates the need for mechanical constraints and instead relies only on the Gestalt principle of good continuation.

Table 4-4 Summary table of TORTIS

Programming style	Programming modality	Persistent program?	Gestalt principles	Number of physical program dimensions	Functionality
Imperative	Programming-by-building	Yes	Good continuation	1	Image transformations

- **Robo-Blocks**

Sipitakiat and Nusen’s (2012) Robo-Blocks controls a floor robot based on a program constructed from c*ommand blocks* and a *master block*. This is done by aligning command blocks with the master block. All blocks contain embedded electronic circuitry while magnets on the sides provide mechanical stability and electrical communication pathways. The master block interrogates the line of command blocks to determine their sequence. The robot can be programmed to turn left and right, move forward and back, prepare to draw, stop drawing, and make sounds. Rotation knobs can set program parameters.

In Chapter 6, I incorporate similar instructions in my T–Logo tangible programming environment. However, instead of setting a parameter value by means of a rotation knob, T-Logo derives the value from a set of grouped objects.

Table 4-5 Summary table of Robo-Blocks

Programming style	Programming modality	Persistent program?	Gestalt principles	Number of physical program dimensions	Functionality
Imperative	Programming-by-building	Yes	Good continuation	1	Image transformations

- **Turtan**

When constructing a program using Turtan (Gallardo et al. 2008), the user positions program objects along an imaginary line on top of the programming surface. A program can incorporate seven elements referred to as *move without painting*, *move and paint*, *rotate*, *scale*, *change colour*, *repeat*, and *start*. Elements that take numerical parameters derive these from the direction in which the object points.

Objects need not actually touch to indicate the sequence in which they should be interpreted; instead, implicit links are created based on the distance between objects. I use the terminology *implicit linking* to describe the mechanism by which such connections are made. Implicit linking may be either dynamic or static and Gallardo et al. (2008) experimented with both. In general, linking is determined by finding the geometrically shortest distance between a newly introduced object and existing program objects. If this object is later moved its closest object may no longer be the original to which it was linked. In the case of static links, the existing link is not severed when the object is moved. However, for dynamic linking, the link is severed and the object is then linked to one that is geometrically the closest. Although not stated explicitly, the dynamic linking mode incorporates the Gestalt principle of grouping by proximity.

Also inferred is that the `Turtan` design relies on the user's ability to place objects according to the Gestalt principle of good continuation. Figure 4-6 illustrates the Gestalt principle of good continuation by superimposing a dashed line onto a representation of a Turtan program. In Chapter 6, I not only incorporate the Gestalt principles of good continuation in the design of my T-logo programming environment but also add grouping by proximity.

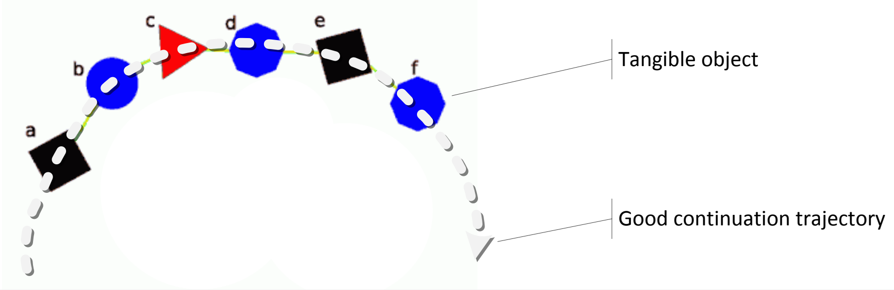


Figure 4-6 The Turtan system incorporates the Gestalt principle of good continuation

Adapted from Gallardo et al. (2008)

Table 4-6 Summary table of Turtan

Programming style	Programming modality	Persistent program?	Gestalt principles	Number of physical program dimensions	Functionality
Imperative	Programming-by-building	Yes	Good continuation	1	Image transformations

- **ReacTable**

Jorda et al.'s (2005) ReacTable is a dynamic tangible programming environment in which the objects are placed on a horizontal surface. The function of the program is determined by the relative positions of the objects and their pre-assigned functions. No physical links exist between the objects and associations are dynamically created and destroyed as the relative positions change. An object can take the form of a cube, thereby exposing one of six sides for interpretation. Manual rotation around the axis perpendicular to the table surface adjusts an associated parameter.

Of interest to my research is the mechanism by which objects are associated with each other. To make an association between objects, the user applies the Gestalt principle of perceptual grouping by proximity and places the relevant objects in close proximity to each other in such a way that they appear to form a group that is separate from any other objects close by. Once the ReacTable system has identified this grouping, the user is free to move the objects apart while still keeping the grouping intact. Figure 4-7 is a photographic depiction of the ReacTable programming surface.

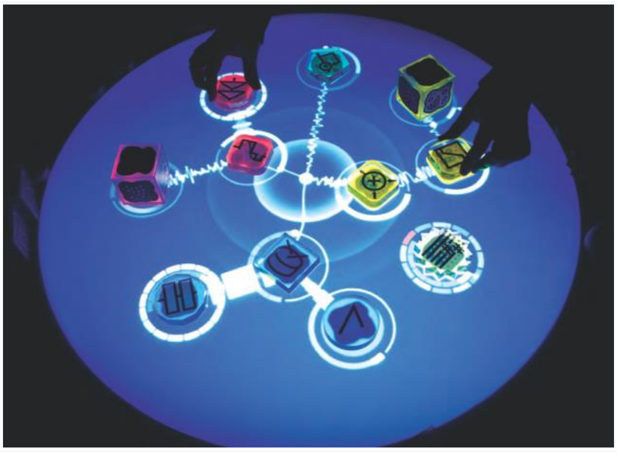


Figure 4-7 The ReacTable and the Gestalt principle of perceptual grouping by proximity

Jordà, Julià and Gallardo (2010)

Table 4-7 Summary table of ReacTable

Programming style	Programming modality	Persistent program?	Gestalt principles	Number of physical program dimensions	Functionality
Functional	Programming-by-building	Yes	Grouping by proximity	2	Waveform generation and transformation

- **SiteView**

SiteView (Beckmann & Dey 2003) is a system that combines interactors, a world-in-miniature (Stoakley, Conway & Pausch 1995), and a condition composer. Figure 4-8 is a photograph with descriptive annotations of a typical SiteView programming environment. A program is comprised of user-defined rules that include actions and conditions. Action interactors determine how the physical world should change while condition interactors represent real world states.

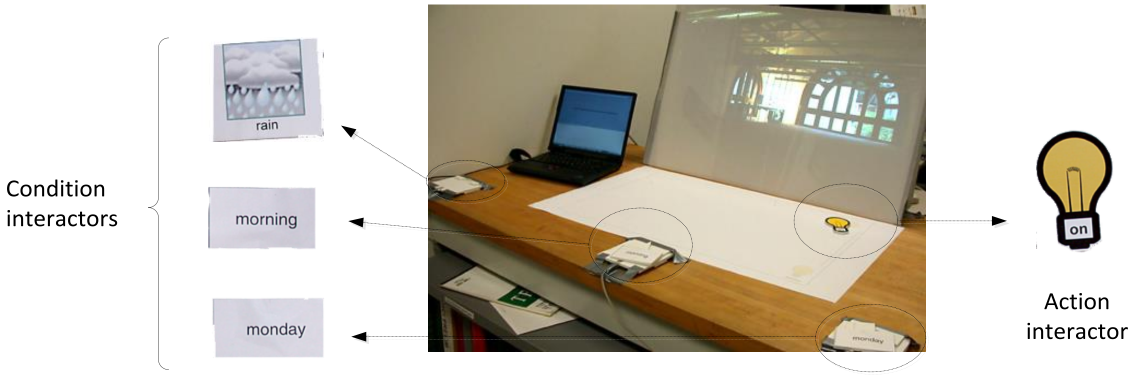


Figure 4-8 The SiteView tangible programming environment

Based on Beckmann and Dey (2003)

Program rules are recorded when the user demonstrates what actions follow when certain conditions hold. The user does this by placing action and condition interactors in their respective designated positions on the table. The programming interface design is such that the action and condition interactors remain spatially separated. Using as an example the control of a light, Figure

4-8 illustrates the SiteView programming environment that demarcates to the user the areas where to place the condition interactors. These areas are: on the left for the weather condition, in the centre for the time of day, and to the right for the day of week.

In Chapter 6, I consider an alternative approach to setting the program rules: Instead of assigning four spatially separated areas respectively for three condition interactors and one action interactor, I apply the Gestalt principle of perceptual grouping by proximity. Figure 4-9 demonstrates how the rule in Figure 4-8 can be implemented using my approach.

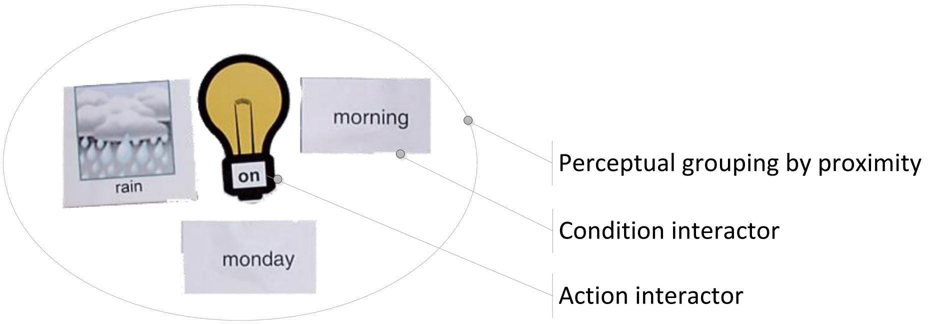


Figure 4-9 My application of the Gestalt principle of perceptual grouping by proximity to SiteView

Table 4-8 Summary table of SiteView

Programming style	Programming modality	Persistent program?	Gestalt principles	Number of physical program dimensions	Functionality
Rule-based	Programming-by-example	No	None	1	Environmental control

- **Media Cubes**

Blackwell and Hague’s (2001a) Media Cubes is a programming-by-demonstration system that consists of cubes. Images and text labels embellish the sides and these represent objects, states, and actions in the physical world. A program rule is comprised of a “when” part and a “do” statement (Hague, Robinson & Blackwell 2003). The “do” statement is executed if the “when” world state holds true.

To create a rule the programmer selects cubes with representative faces of the object, the world state, and an action. The user then holds together two cube faces representing “when” and the requisite world state, respectively. She also presses a button on each of the two cubes to indicate that this configuration should be recorded (Blackwell & Hague 2001b). This action sets the “when” part of the program rule. The “do” part is set when cube faces representing the object and desired action are pressed together.

The authors envisaged a version in which the controlled object is not only represented by a cube, but can be directly incorporated into the programming activity. For example, to set the "do" part of a program that turns a coffee maker machine on, the user holds the cube face representing a "power on" action directly to a suitably augmented coffee making machine and then presses a button on the cube and one on the machine to record the action.

Of particular interest to my research is the *aggregate cube*. An aggregate cube represents all the objects placed inside it. It can therefore represent a collection of objects, events, or actions. Conceptually, an aggregate cube is similar to Oh et al.'s (2013) Digital Dream Lab instance block as described elsewhere in Section 4.3.1.2. When viewed in context of the Gestalt principles, the aggregate cube can be regarded as a physical constraint that affords grouping by common region. In Chapter 6 I discuss the use of physical constraints as a means to group program elements.

Table 4-9 Summary table of Media Cubes

Programming style	Programming modality	Persistent program?	Gestalt principles	Number of physical program dimensions	Functionality
Rule-based	Programming-by-example	No	Grouping by proximity	1	Environmental control

4.3.1.2 Objects utilised for their two dimensional properties

A notation incorporating two dimensions is sufficient to describe certain tangible programs such as the Digital Dream Lab (Oh et al. 2013). This puzzle block system implements a token-and-constraint metaphor. Tokens include a character, animation type, colour, and size. These tokens are respectively referred to as *character*, *animation*, *color* [sic], and *size blocks*.

Blocks are shaped as puzzle pieces and constrain placement so-as to guide the user. The size block affords a mechanism with which the user can change the character's size. Jacucci (2007) refers to such an element as a *utensil*. This block is disk-shaped and fits in a circular cutout of the character block. The animation block determines the action the character performs and the colour block sets its appearance. A *component block* is another optional constraint that is used to hold tokens. *Instance blocks* are compact virtual copies of the component block and its tokens. A user may apply the instance block in the program as-if it is an exact copy of the associated component block. Multiple copies of the component block may therefore exist simultaneously in a program. The relevant attributes of all copies are affected when the user changes or adjusts the tokens of a component block. Figure 4-10 illustrates a component block with tokens and its associated instance block.

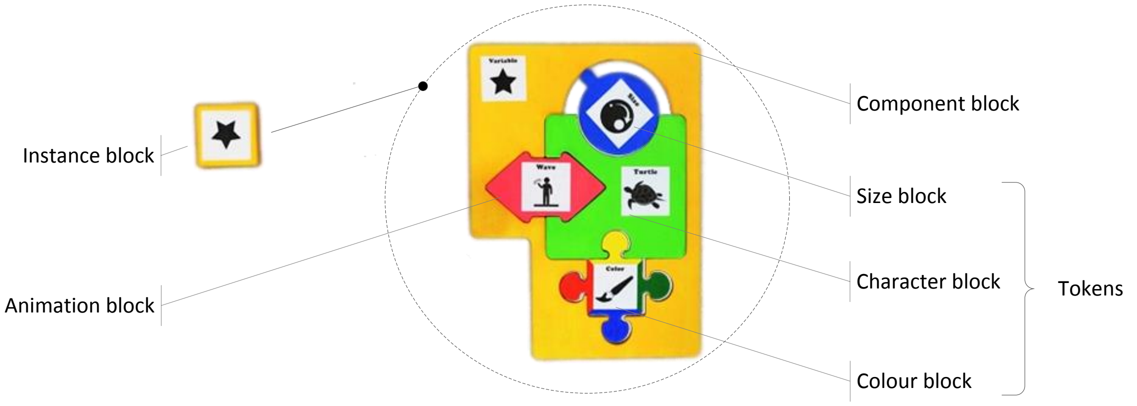


Figure 4-10 The instance block is functionally an exact copy of the component block

Based on Oh et al. (2013)

The Gestalt principle of grouping by common region is evident in a system that constrains multiple tokens to the same area. In Chapter 6, I explore the token-and-constraint metaphor using wooden hoops to define the constraining area. However, instead of restricting the number of tokens and their positions within the common region as Oh et al. (2013) do, the user is free to place multiple tokens of the same type in the same region. In addition, the user may adjust the relative orientations of the tokens according to personal preference.

Table 4-10 Summary table of Digital Dream Lab

Programming style	Programming modality	Persistent program?	Gestalt principles	Number of physical program dimensions	Functionality
Functional	Programming-by-building	yes	Grouping by common region	2	Animation

4.3.1.3 *Objects utilised for their three dimensional properties*

Some tangible programs can only be described using a system comprising of three dimensions. Examples include Schweikardt and Gross's (2006) roBlocks and Marti and Lund (2004) and Ngo and Lund (2004) and Nielsen and Lund (2008) and Nielsen's (2002, 2008) I-BLOCKS. These also happen to be physical programming systems in which the configuration of the physical artefact serves as both the program and the object being controlled. Both systems consist of discrete cubes that are physically and electronically interconnected. The cubes sense the physical world, perform calculations based on what has been sensed, and control actuators according to the calculation results. Data flow through these cubes according to their physical typology.

In the case of roBlocks, the flow of data does not simply originate at a cube and terminate at another; instead, the data propagate throughout the structure and reaches all cubes. The priority with which a cube deals with data varies according to the distance data have propagated through the structure. Priority decreases in direct proportion to the distance. Therefore, when multiple data streams reach a cube the data that has propagated along the short distance has the greatest effect.

Table 4-11 Summary table of roBlocks

Programming style	Programming modality	Persistent program?	Gestalt principles	Number of physical program dimensions	Functionality
Mixed behaviour with priorities	Programming-by-building	Yes	Good continuation	3	Mechanical animation

4.3.2 Environments that incorporate objects with personal meaning

In this section, I discuss the group of tangible programming environments in which the user creates or uses objects of choice when constructing a tangible program. The space allocated here to the StoryRoom environment reflects its contributions to this thesis. One contribution is to base rules on the Gestalt principle of perceptual grouping by proximity. Another is that the order of objects within the group is not significant.

4.3.2.1 StoryRoom

Brainstorming activities helped define Montemayor et al. (2002), Montemayor (2003), and Montemayor et al.'s (2004) StoryRoom physical programming environment in which the user combines low-tech and high-tech objects to create program elements. Brainstorming transpired at the University of Maryland's Human-Computer Interaction Lab where adults and elementary school children participated. Two visual language designs emerged from the brainstorming activities and these served as inputs to the final StoryRoom design. The third and final programming language involves spatial gestures and takes no physical form. All designs use grouping to define a program.

The following discussion first addresses the final programming language design and then the initial and second designs. This unusual approach allows the narrative to develop useful examples that are subsequently applied to describe important concepts that emerged from the initial and second design iterations.

- **The final programming language**

StoryRoom is a room-sized area that contains passive props and active physical icons (phicons). In this space, phicons sense the environment and act according to the user's program rules. To tell a story, the user combines props with phicons and creates program rules. The user designs and constructs props according to his personal preference. Placing a phicon and a prop close together establishes a mental bond between the two.

The final StoryRoom programming environment incorporates a wand, actuator phicons, and sensor phicons. Figure 4-11 depicts the wand, two light actuator phicons, and three pressure sensing phicons. The wand has two ends: the star shaped 'yes' end signifies a constructive action and the cross-shaped 'no' end corresponds to a destructive action. A program rule is defined when the user touches the wand to one or more phicons. Corresponding actions are added to the program when either end touches the phicon. For example, when the user touches a light phicon using the 'yes' wand end, an action to turn the light on is added to the current rule. Conversely, touching the same phicon with the 'no' wand end sets a rule to turn off the light. Like textual and graphic programs, rules do not manifest as tangible entities and exist only in computer memory.

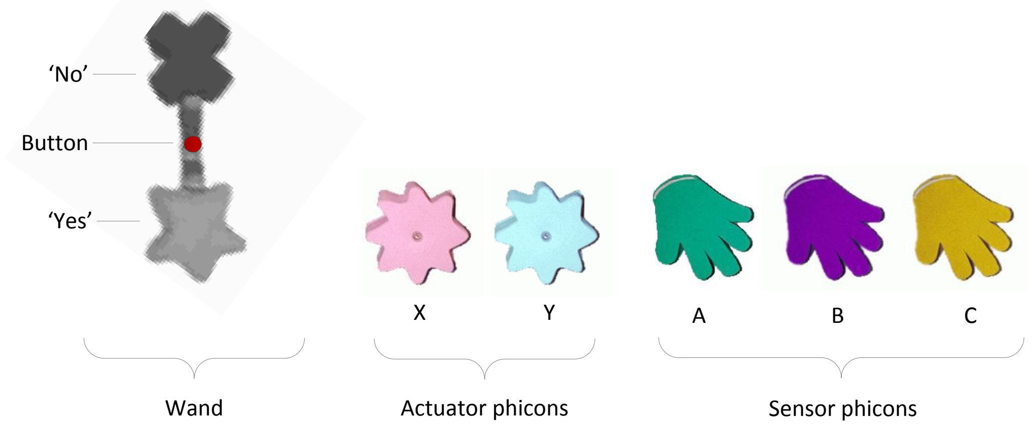


Figure 4-11 A selection of StoryRoom physical programming elements

Based on Montemayor (2003)

The user indicates the start of a new rule by briefly pressing the button on the wand. This button press is a temporal activity that serves to group phicon events into a single program rule and rules can include multiple phicon events. A rule is finalised when either the programming mode is exited or when the button is pressed again at the onset of a new rule.

Sensing phicons in a rule are interpreted as if they are combined with a logical AND operation. Rules function independently of each other much like the logical OR operation. To elaborate, when a rule contains multiple sensing phicons the rule only executes when all have been triggered simultaneously. Likewise, all actuating phicons in a rule execute together when it triggers.

The following two programs demonstrate how the wand, sensors, and actuators are used. The first demonstrates the creation of a rule that will result in the actuation of object X when objects A and B are triggered together. To construct the program, the user momentarily depresses the button and then touches objects A, B, and X using the 'yes' end of the wand. Now, the programed rule dictates that object X will be activated when both objects A and B are triggered. The rule can be expressed verbosely as in Expression 1 and by using a combination of icons and symbols as in Figure 4-12.

Rule 1: when object A and object B are triggered then actuate object X (1)

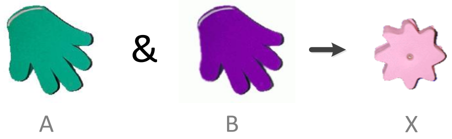


Figure 4-12 A combined iconic and symbolic depiction of the first program rule

Assuming that the & symbol signifies a logic AND operation, then the following is a concise symbolic representation of this rule:

Rule 1: A & B → X

The second example extends the first by adding the logic OR condition and this is done as follows: Having already used the wand to touch objects A, B, and X, the user again briefly depresses the button on the wand. This action indicates to the SmartRoom programming system that the current rule has been concluded and that a second rule is about to be defined. The user now manipulates the wand so that the 'yes' end touches objects C and X in sequence. By removing the wand from the programming area, the user indicates to the SmartRoom programming system that the programming activity has ended and that the program is ready to be executed. The second program can be expressed using icons and symbols as in Figure 4-13 and verbosely as follows:

Rule 1: when object A and object B are triggered then actuate object X

Rule 2: when object C is triggered, then actuate object X

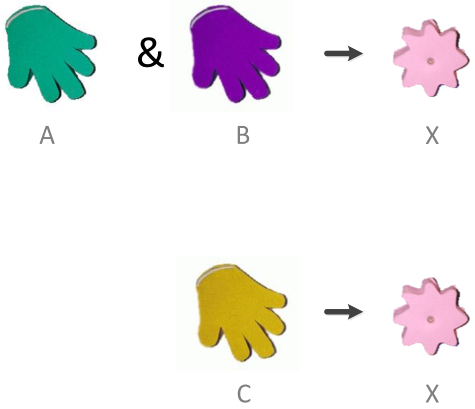


Figure 4-13 A combined iconic and symbolic expression of program Rule 1 and Rule 2

Object X is therefore actuated if objects A and B are triggered together, or if object C is triggered. Using the | symbol to signify the logic OR operation, the following is a concise expression of both rules:

(A & B) | C → X

For the programming activity, all actions that happen between the first and second button presses are grouped together. I call this *temporal grouping*. Both the StoryRoom programming activity and the symbolic/iconic expressions that represent this activity employ the Gestalt principle of perceptual grouping of a temporal or proximity variety. As far as the symbolic/iconic expressions that represent this activity are concerned, symbols and icons to the left of the → symbol are perceptually grouped together due to both convention and their close proximity. Figure 4-14 highlights the correspondence between temporal and proximity grouping for Rule 1.

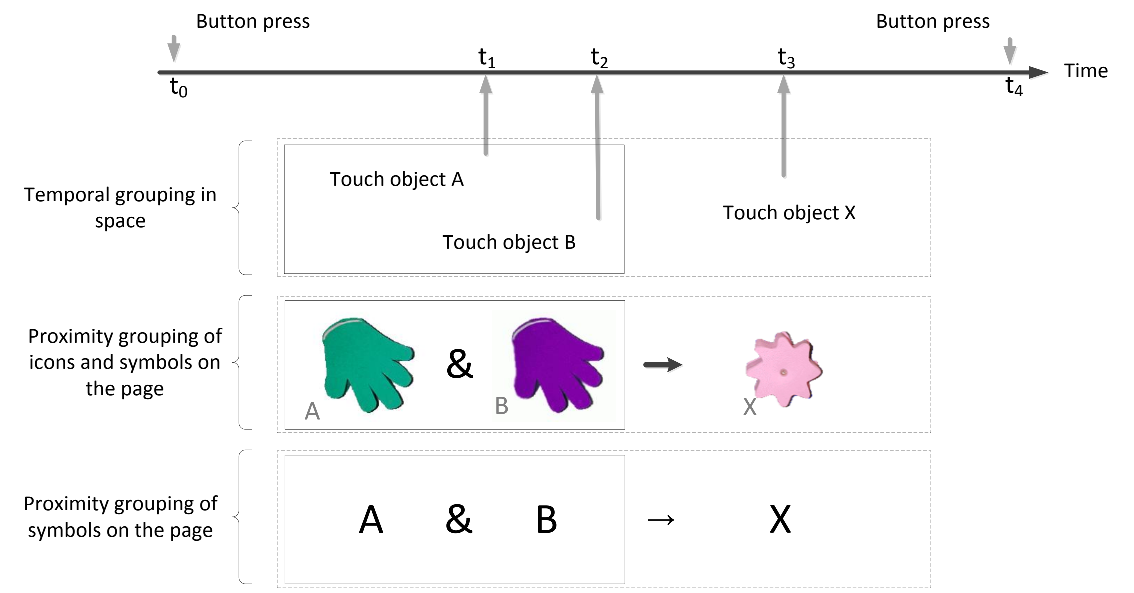
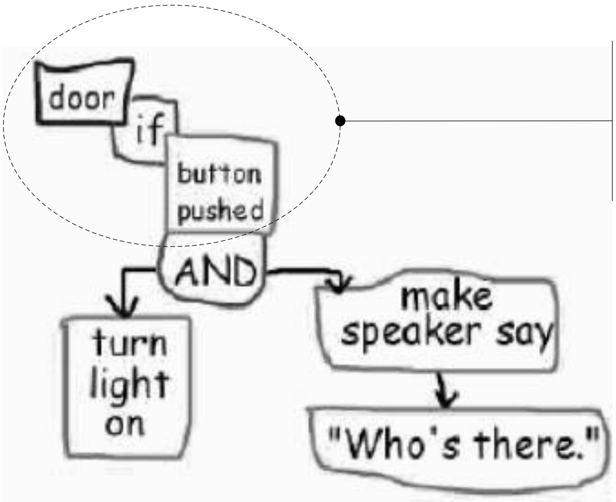

Button press
Button press
t_0
t_1
t_2
t_3
t_4
Time
Temporal grouping in space
Touch object A
Touch object B
Touch object X
Proximity grouping of icons and symbols on the page
&
A
B
X
Proximity grouping of symbols on the page
A & B → X

Figure 4-14 Temporal, iconic, and symbolic grouping identified in Rule 1

The above programming language emerged from a workshop in which two visual languages were initially considered. Although the final language incorporates some elements of the initial designs, some undeveloped concepts warrant further investigation and I discuss these in the following two subsections.

- **The first visual language**

The first visual language to emerge in the workshop incorporates words in boxes that represent objects, events, and branching decisions (the final StoryRoom design does not include branching). Overlapping boxes indicate that a relationship exists between the objects, yet this relationship is not explained. In other words, what matters is the proximity of the boxes to each other and not the order nor the direction in which they are stacked. Figure 4-15 is an example of a program constructed using this visual language. The group of encircled overlapping boxes (labelled door, if, and button pushed) represents the following program statement: “if the button on the door is pushed”.



Overlapping boxes indicate that a relationship exists between the boxes.

Figure 4-15 Overlap implies relationships between objects

Based on Montemayor (2003)

The discussion of the final StoryRoom language above follows the Western left-to-right reading convention. This convention dictates that logic expressions (such as those depicted in Figure 4-12 and expression (1)) are interpreted from left to right. Also, the → symbol distinguishes between the 'before' and the 'after' portions of an expression. However, the 'boxes' visual language of Figure 4-15 suggests a notation that is interpreted as a whole and at once and is not interpreted along any particular direction. I explore such a notation next.

Prompted by the first StoryRoom visual language design in which some expressions are orientation independent, I put it that the → symbol can be omitted in cases where there exists no risk of the before and after portions in an expression being interchanged. Consider for example a before portion that is comprised of camping commodities (matches and wood are examples) and an after portion representing the concept of fire. One of the possible logic relations that exist between these three nouns can be expressed as follows:

```
matches & wood → fire                                   (2)
```

However, I propose that even when the → and & symbols are omitted from this expression, the interpretation of this group of nouns remains the same. Therefore, the following three symbol sequences are valid representations of Expression 2:

```
matches,wood,fire     wood,fire,matches     fire,matches,wood
```

This notation can be applied to rule Rule 1 and therefore the following sequences represent this rule equally:

A, B, X   A, X, B   B, X, A   B, A, X   X, B, A   X, A, B.

The preceding shows that certain symbols are redundant in programming environments such as StoryRoom and that physical icons alone can sometimes adequately express program rules. Therefore, Figure 4-16 illustrates four alternative iconic representations of Rule 1 while Figure 4-17 illustrates four more representations of Rule 1.

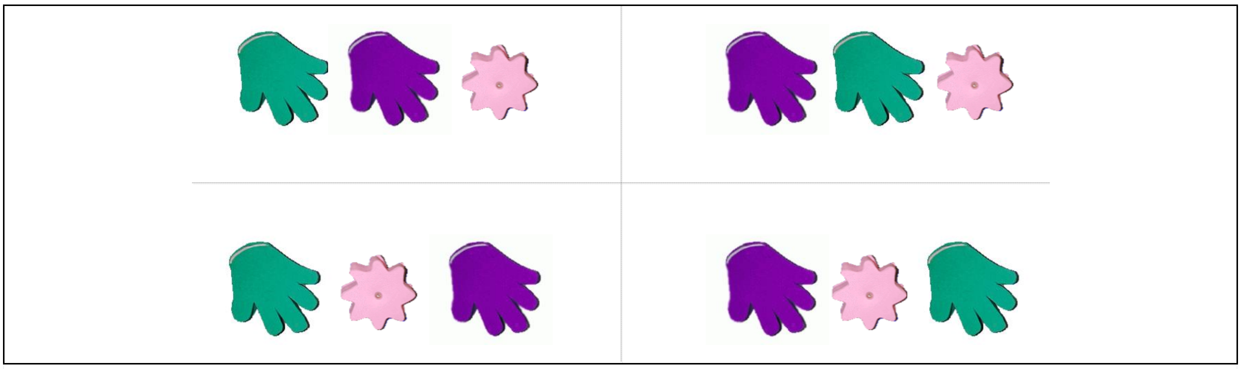


Figure 4-16 Alternative Rule 1 sequences

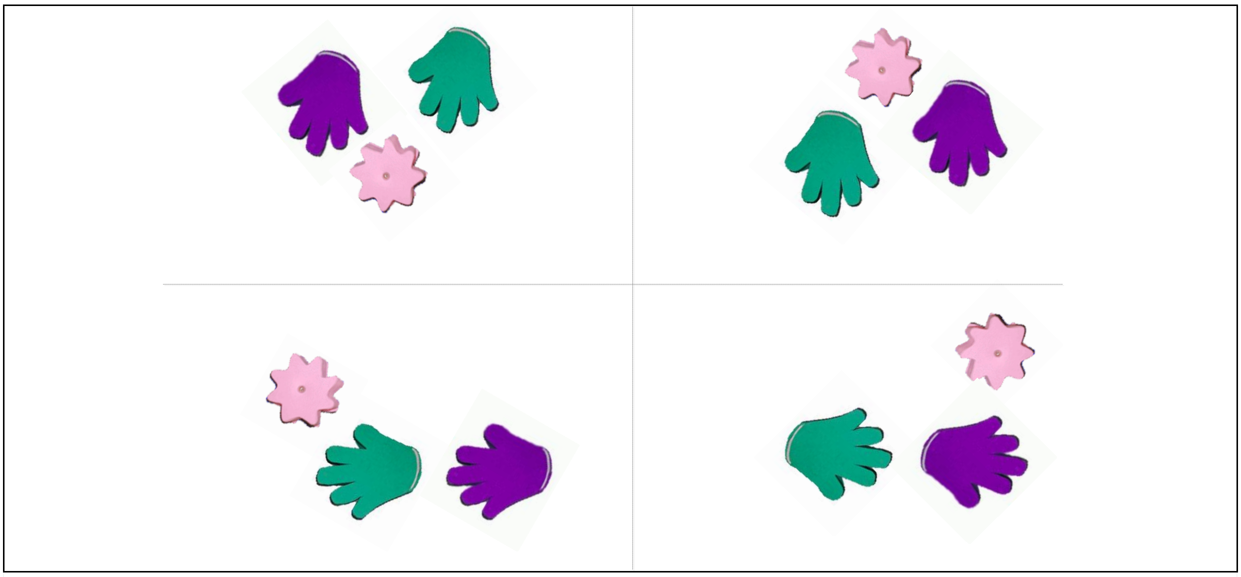


Figure 4-17 Alternative Rule 1 groupings

- **The second visual language**

The second visual language includes only iconic sentences and no words. Instead of words, workshop participants used icons of which the meanings had been agreed on. They designed these icons to hold personal significance. For example, the wavy lines that separate the camera and cup icons in Figure 4-18 represent close physical proximity between these objects.

Figure 4-18 depicts three program rules constructed using the second language. Of interest is the third sentence that represents the instruction "when the camera and the cup are near each other, the light comes on and the ear will listen." This sentence defines program Rule 3 which states that the light and ear artefacts are triggered when the camera is close to the cup.

Rule 3: when camera and cup are close together,
then actuate the ear and light

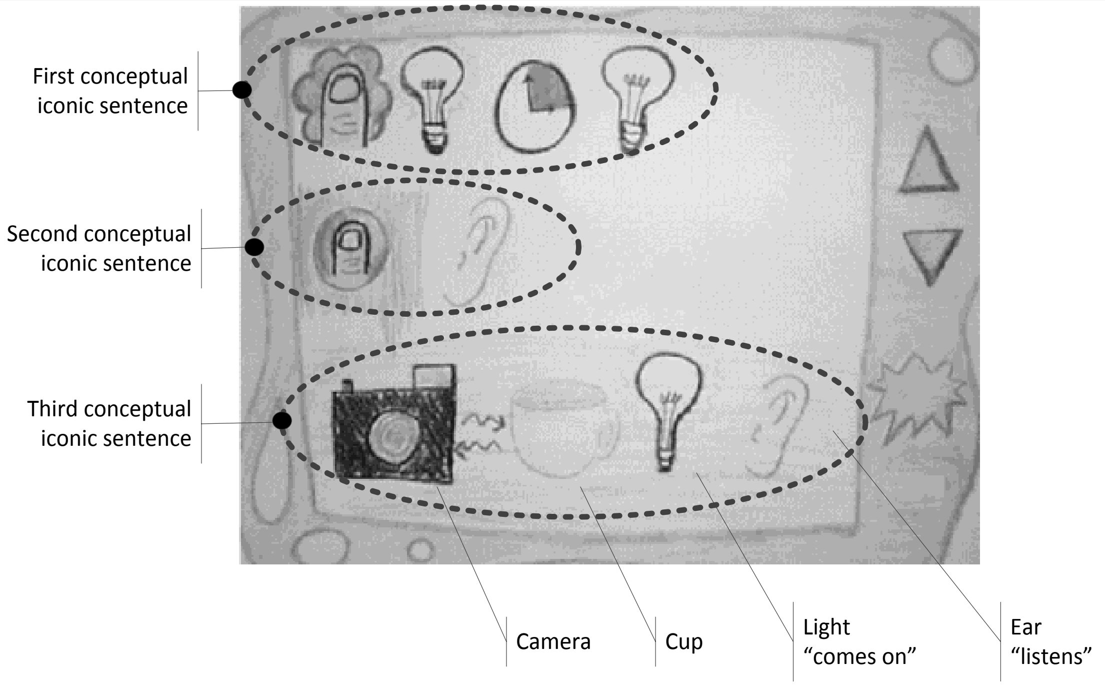

First conceptual iconic sentence
Second conceptual iconic sentence
Third conceptual iconic sentence
Camera
Cup
Light “comes on”
Ear “listens”

Figure 4-18 Three conceptual iconic sentences

Based on Montemayor (2003)

While applying my notation introduced in Section 4.3.1.1 and illustrated in Figure 4-5, Figure 4-19 illustrates the third conceptual iconic sentence shown in Figure 4-18. This figure highlights two Gestalt grouping types. The first is perceptual grouping by proximity and a yellow background highlights the associated components. The second perceptual Gestalt principle is grouping by good continuation. One grouping is highlighted using an orange background and the other grouping has a purple background. I use tangible objects in Figure 4-20 instead of the non-descript symbols of Figure 4-19.

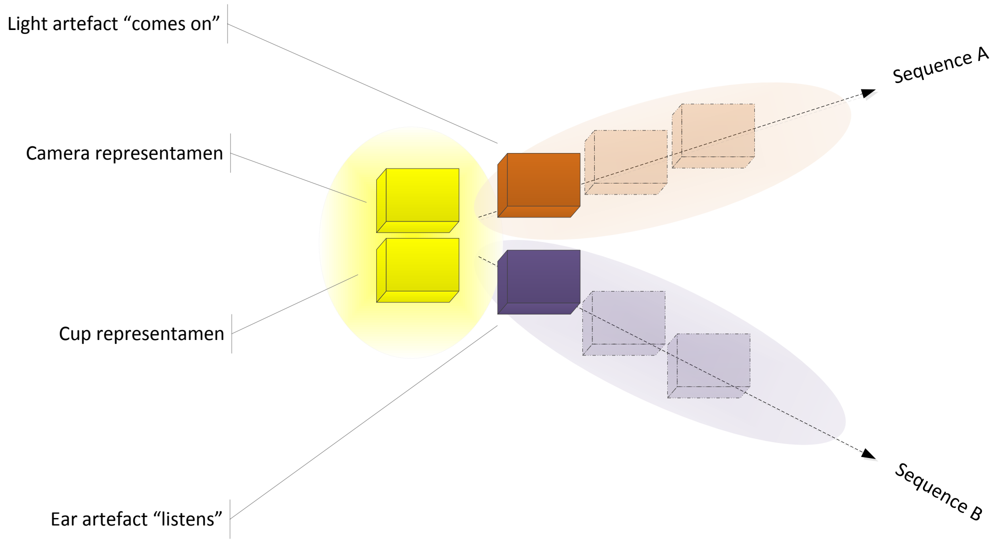


Figure 4-19 My depiction of Montemayor's conceptual iconic sentence using non-descript symbols

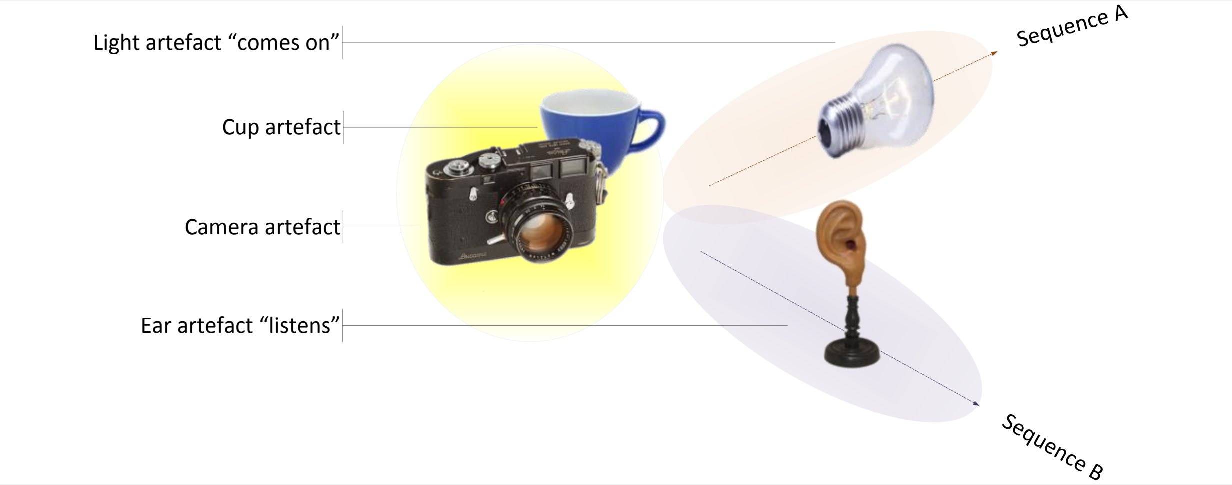


Figure 4-20 My depiction of Montemayor's conceptual iconic sentence with tangible objects

In Chapter 6, I incorporate the observations above in the design of a tangible programming environment. First, I use the Gestalt principle of grouping by proximity to determine which objects constitute a program rule. Second, the user can design and create personally meaningful physical icons and use these to construct a program.

Table 4-12 Summary table of StoryRoom

Programming style	Programming modality	Persistent program?	Gestalt principles	Number of physical program dimensions	Functionality
Rule-based	Programming-by-demonstration	Yes	Grouping by proximity.	1	Storytelling

4.3.2.2 Quilt Snaps

Quilt Snaps (Buechley et al. 2005, 2006) consists of a square quilt patch (Figure 4-21a) that contains electronic circuitry. The red dot in Figure 4-21b represents the electronic circuit and the program that it executes, while the three smaller dots represent available touch and light sensing modalities, and light output. The blue lines indicate the flow of information as it moves from the inputs, through the circuit, and finally to the outputs. Sensing strips on the sides detect touch whereas software controls circuitry that turns the embedded light on and off.

Inputs feed from adjacent quilts and outputs send signals to adjacent quilts. Press-studs serve as keyed connecters and indicate to the user which sides receive inputs and which ones generate outputs. A quilt can either behave independently or interact with others in a quilt network.

Each patch is already assembled and programmed by the time it reaches the user. The user then adds the embellishments according to personal preference and connects the patches to create a larger program. Figure 4-22 shows three larger programs created when individual patches are combined. Also shown is the flow of information through the programs.

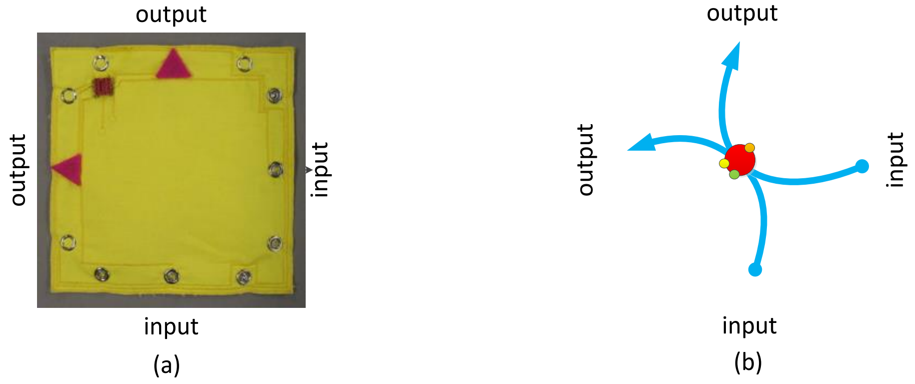


Figure 4-21 A Quilt Snaps patch with its associated information flow

(a) is based on Buechley et al. (2005)

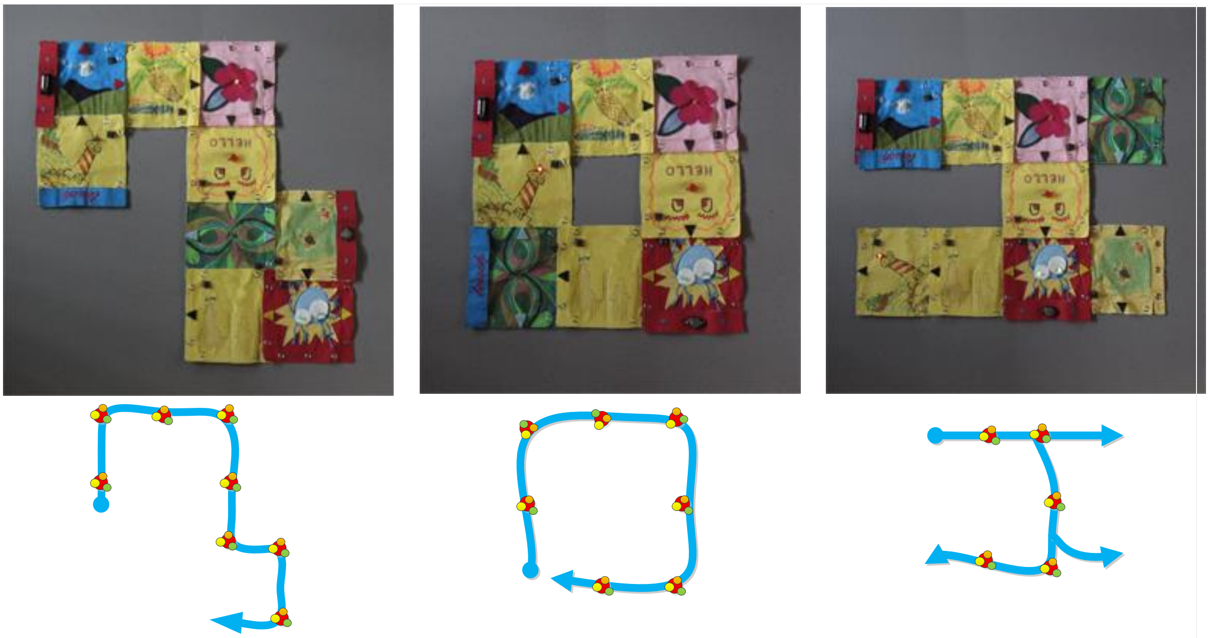


Figure 4-22 Three Quilt Snaps program examples and their information flow

Based on Buechley (n.d.)

The functionality of an individual quilt is predetermined by the circuit designer and cannot be altered by the user. However, the behaviour of a system that comprises multiple quilts is determined by the configuration in which the user has connected the quilts. Since each quilt is a stand-alone computational entity and since multiple quilts can be reconfigured to produce predictable results, the Quilt Snaps system can be classified as a programmable machine.

Table 4-13 Summary table of Quilt Snaps

Programming style	Programming modality	Persistent program?	Gestalt principles	Number of physical program dimensions	Functionality
Imperative	Programming-by-building	Yes	Good continuation	2	Decoration

4.3.2.3 Diorama Table

Diorama Table (Takahashi & Sasada 2005; Takahashi 2007c) is a system in which the user places everyday objects on a surface. The position and shape of the objects determine the nature of the animation that is projected onto the surface. Examples of objects used this way include shoestrings, knives, forks, and cups with saucers.

To demonstrate how the relative position of two objects influences system behaviour, consider the following four scenarios. First, when a single string is placed on the input surface the projected animation is centred on this. Second, when two (non-touching) strings are placed on the input surface, the animations are centred on each string and independent of the other. Third, when the two strings are made to touch each other the animation incorporates both pieces as if they were a single string. Finally, when a string forms a loop, an animated projection fills the enclosed area. Objects such as a knife and cup-with-saucer have yet other effects on the system. Figure 4-23 shows the Diorama Table in use.

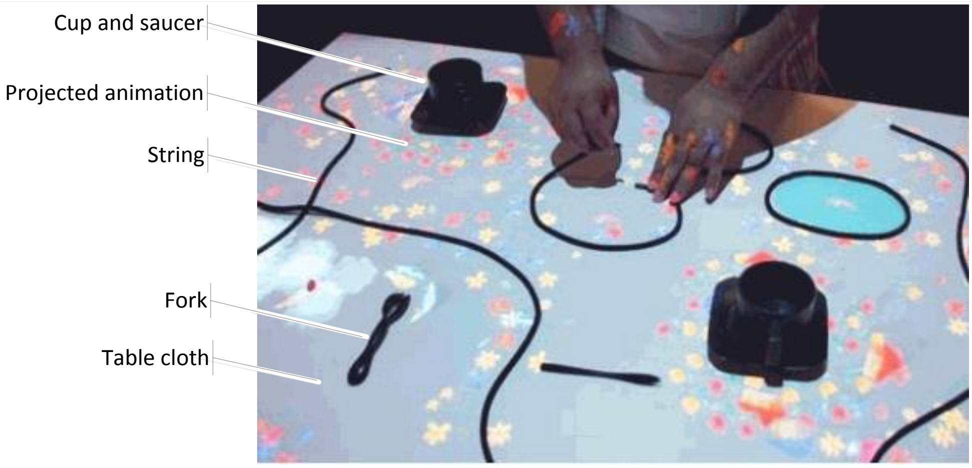


Figure 4-23 Diorama Table with everyday objects and projected animation

Based on Takahashi (2007b)

From this description, it is evident that the user can control system behaviour by placing everyday objects on the input surface. Diorama Table can therefore be considered to be a user programmable system in which a program is defined by the outlines and positions of the objects.

Table 4-14 Summary table of Diorama Table

Programming style	Programming modality	Persistent program?	Gestalt principles	Number of physical program dimensions	Functionality
Functional	Programming-by-building	Yes	Good continuation and grouping by proximity	2	Animation

4.3.3 Conclusion to this section

Of particular interest to this study are programs that can be viewed in their entirety at a single glance. Physical persistence is therefore relevant and visual Gestalt principles describe how the user experiences the completed program. Of the programming environments considered, the principles of visual perceptual grouping by proximity and visual perception of good continuation are the most prolific.

This section also investigated the origin of programming objects. I first considered tangible programming environments in which the user is given pre-made programming objects that are then arranged and combined to create a program. The second part is the most interesting and addressed systems that incorporate user-defined programming objects.

Story Room, Quilt Snaps, and Diorama Table let the user determine (within certain constraints) the appearance of objects incorporated into a program: StoryRoom combines previously made physical icons with user-crafted tangible objects to provide a programming environment with which the user can tell a story. The system designer determines the set of physical icons and their appearance. However, the user is free to combine the physical icons with objects of his own design and thereby create a personally meaningful programming environment. Although the user is constrained by Quilt Snaps' fixed form (a square patch), input and output locations, and embedded program, the user is free to embellish each patch according to personal preference. The Diorama Table constrains the user through the a-priori mapping of object outlines to predetermined animation types. However, the user can still create and decorate these objects.

On their own, the user-created objects in StoryRoom and the embellishments in Quilt Snaps are not sufficient to define a computer program. Instead, these objects are used in conjunction with items made by the system designer. The user-created objects of the Diorama Table also hold no inherent significance; rather, the value of these objects are determined by their outlines and the functions assigned by the system designer.

From the preceding it is evident that, on their own, user-created objects are insufficient to create a computer program. This is because the objects have not yet been mapped to their digital counterparts by means of a sensing system. However, when used in conjunction with a pre-existing programming system the result is a programming environment in which the user can incorporate personally meaningful objects.

4.4 Discussion

In the previous sections, I examined tangible programming systems to identify their underlying Gestalt properties. I also determined the mechanisms by which the user can incorporate personally meaningful objects into their programs. I found that designers do not explicitly incorporate Gestalt principles, yet certain Gestalt principles are evident in their designs. In addition, I identified that in general designers do not explicitly make provision for the incorporation of personally meaningful objects. In this section, I discuss the Gestalt principles I identified and I make suggestions regarding the inclusion of personally meaningful objects into systems not designed for this purpose.

4.4.1 The Gestalt principle of good continuation

Although not explicitly discussed in the literature and not previously identified as such, Electronic Blocks (Wyeth & Wyeth 2001), TORTIS (Perlman 1974, 1976), Turtan (Gallardo et al. 2008), and Diorama Table (Takahashi & Sasada 2005; Takahashi 2007c) incorporate the Gestalt principle of good continuation. I identified this shared Gestalt principle across these systems and I highlight its utility by making it explicit as a Gestalt construct in my model in Chapter 7. I next discuss the underlying Gestalt principle of good continuation that I identified in these systems as well as the mechanisms that support the principle. Of the three mechanisms covered, the vision-based option affords the user the most choices when constructing a program.

The Gestalt principle of good continuation is evident in Electronic Blocks where vertically stacked and interlocking plastic cubes form a sequence of program instructions. Instructions are interpreted from top to bottom along an imaginary line. Instead of a vertical configuration, the user places TORTIS, Robo-Blocks, and Turtan objects in a horizontal plane. The TORTIS design is based on a linear arrangement of program statements that the user places side-by-side along a straight track. The track implicitly incorporates the Gestalt principle of good continuation. Robo-Blocks (Sipitakiat & Nusen 2012) is also implicitly based on this principle and incorporates magnets to keep the objects in a straight line.

Turtan and the Diorama Table do away with need for mechanical and magnetic constraining mechanisms as used in Electronic Blocks, TORIS, and Robo-Blocks. Instead, they use an optical system to identify the programming objects and this affords the user an opportunity to adjust the object arrangement while retaining the original program logic. The user option to feely adjust relative object positions is not possible in programming environments that rely on physical constraints. In addition, the Turtan design implicitly assumes that the distances between the objects are an integral part of the program logic.

As far as Quilt Snaps (Buechley et al. 2005) is concerned, I propose that it is possible to visualise four imaginary data flow lines that connect a square's inputs to its outputs as illustrated in Figure 4-21. These lines merge at the embedded electronic circuitry where the data are interpreted and new data generated. Once the user has linked the squares to each other, I put it that it is possible to imagine routes along which data can flow across the resultant structure. I also posit that the imaginary data lines continue from one quilt edge to the next thereby forming in the user's mind a network of data paths. Based on this, I conclude that the Gestalt principle of good continuation describes the imaginary data paths that form across the quilt structure as shown in Figure 4-22.

4.4.2 The Gestalt principle of grouping by common region

Digital Dream Lab (Oh et al. 2013) applies physical constraints to limit the number of positions in which the user can place objects. The purpose of this design is to guide the inexperienced user during the program construction activity. A problem with this design is that it only supports one instance of any given object type in a program statement. Furthermore, the user is not free to adjust the relative positions of the objects according to personal preference.

I put it that systems similar to Digital Dream Lab can be improved by giving users an opportunity to include more than one object of a given type in a statement. For example, a statement can contain an object that will make the character wave its arm and another object to make it walk. The result would be a character that waves its arm while walking.

Another problem with token-and-constraint systems such as Digital dream Labs is that these do not allow the user to adjust the relative orientation of objects in a program. One solution to this problem is to substitute the token-and-constraint approach with another mechanism that supports changes to object orientations without altering their relative positions. To this end, I will explore In Chapter 6 the application of wooden hoops to define a common region. Using this approach, a programming environment assumes that all objects within the common region belong together and interprets the group as a program instruction. A user of a system based on this design is then free to place multiple tokens of the same type in the common region. In addition, the user may now also adjust the relative orientations of the tokens according to personal preference while preserving the program logic.

4.4.3 The Gestalt principle of grouping by proximity

Some programming environments like Oh et al.'s (2013) Digital Dream Lab dictate the relative positions in which objects have to be placed to constitute a proper program instruction. Other environments do not prescribe the relative positions of objects and instead leave that to the user's discretion. An example is Media Cubes. Although this system requires that the programming objects make physical contact with each other to be considered an instruction, the object order is not relevant. A benefit of Media Cubes' design over that of Digital Dream Lab's is therefore that it affords the user with an opportunity to arrange the objects according to her personal preference. For example, if one cube represents a desk lamp and a second cube represents the time to turn on the lamp, then the sequence in which the user arranges the cubes is irrelevant. This approach dictates that the relative positions of objects need not be prescriptive and can instead be left to the user's discretion. The consequence it that the user no longer needs to remember the sequence in which objects have to be placed and can instead position the objects according to personal preference.

Whereas Media Cubes only considers instructions comprising two objects, the Story Room (Alborzi et al. 2000) designs explore instruction compositions comprising multiple objects. To this end, its designers considered the meaning conveyed both by three overlapping squares (Figure 4-15) and closely spaced physical objects (Figure 4-18). I concluded that the Story Room designs implicitly depend on the Gestalt principle of grouping by proximity. However, Story Room does not fully explore object arrangements and I argued that a group of objects can maintain its meaning when the parts are rearranged. I supported my argument using eight arrangements comprised of Story Room sensor and actuator objects.

In addition to considering physical arrangements, Story Room follows an approach that I call grouping-by-temporal-common-region. I put it that when items are selected in a period demarcated by two previously well-defined events this action can be considered as grouping by common region, albeit temporal common region and not spatial common region. To elaborate, grouping by common region assumes a physical region demarcated using, for example, a coloured area or physical barrier. In contrast, I propose that two successive button presses can define a temporal common region group. I label in Figure 4-24 (a) the time interval between the presses as temporal-common-region. The first press in this example demarcates one temporal boundary while the second button press defines the end of the temporal area. The result is that all actions between these two events are grouped together. Likewise, my argument for temporal grouping can be extended to grouping by proximity and the result will then be a new grouping category that I call grouping-by-temporal-proximity (Figure 4-24b). Using the same scenario as for grouping-by-temporal-common-region, I base my argument on the time that transpires between events. That is, if the events are temporally close together then they are considered members of the same group. A challenge in using this grouping is in specifying the maximum time interval between two successive events to be viewed as belonging to a common group. A second problem is to determine the minimum interval between events to consider them as belonging to two distinct groups. I do not explore temporal grouping further in this thesis.

While Media Cubes and Story Room program instructions are not spatially confined, SiteView defines dedicated and spatially separate locations where the user must place objects. I propose that instead of dictating that the objects have to be kept separate, the system can be designed in a way that allows the user to place them in close proximity to each other. This will result in a system based on grouping by proximity. In Figure 4-9, I offered an example of what I propose a program can look like when arranged this way. In this example, the lamp will turn on if it rains on a Monday morning. I further posit that it is possible to include multiple condition interactors of the same type in a single program. For example, a logical OR condition is indicated when a Tuesday indicator is added. As directed by the new configuration, the light will be activated on both Monday and a Tuesday mornings but only when it rains on these days. Likewise, more action interactors may be added and they will be actuated when the conditional expression holds true. To illustrate by means of another an example, if a coffee maker is included in addition to the light actuator, the system will turn on both the coffee maker and the light on rainy Monday and Tuesday mornings.

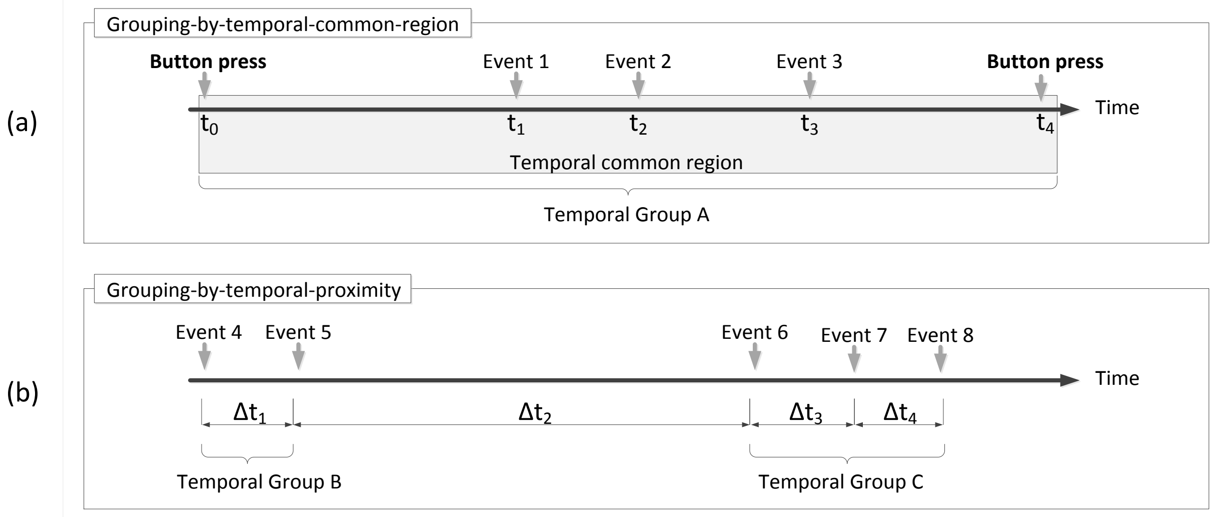


Figure 4-24 Grouping-by-temporal-common-region and grouping-by-temporal-proximity

4.4.4 Personally meaningful objects

There exist programming environments purposefully designed to encourage object personalisation. Story Room and Quilt Snaps are examples of such environments. Using these systems, users can create personally meaningful objects by, for example, embellishing the objects using cardboard and cloth.

In contrast to environments in which the user can incorporate personally meaningful objects in their programs, a group of system designers prescribe the visual appearance of their objects. Examples include Electronic Blocks and TORTIS. Whether the designer's decision to fix the appearance without the individual's input is intentional or not, I argue that all physical programming environments can be modified to include personally meaningful objects. For example, even though the colour of the blocks and icons imprinted on the sides in Electronic Blocks serve to remind the user of their function, the designer does not give the user an opportunity to choose her own colours or other signs. It is therefore the user's burden to memorise their meanings. Based on this argument, I put it that Electronic Blocks can be personalised using pictures and by writing on the plastic parts.

TORTIS is another system that does not give the user an opportunity to create personally meaningful program objects, yet can be adapted for this purpose. Since the program objects are made of cardboard I posit that it is possible to add personally meaningful signs by drawing on the cards or by gluing pictures onto the cards. A final example is Patten, Griffith, and Ishii's Physical Strings system that can be customised to better represent the purpose of each string. To illustrate, the user can attach to each string a written description or another sign to remind her of its purpose. This illustrates that physical strings can be adapted to include personally meaningful objects customised to the user.

4.5 Conclusion

This chapter focussed on programming environments in which a program is defined when the user manipulates physical objects not usually associated with programming. I made a distinction between physical and tangible programming with the former involving computationally ubiquitous environments and the latter defined by the grasp-ability of the objects.

Central to the discussion was the user's involvement in the creation of objects; that is, instead of using premade objects the user chooses personally meaningful objects with which to program. Of the literature studied, only three tangible programming systems explicitly include the user in the design and creation of the programming objects. These systems are StoryRoom, Quilt Snaps, and Diorama Table. Although some design methodologies do include end-user representation (Druin 1999), the design outcome is usually a compromise and often does not afford the user the opportunity to design and create a personally meaningful interface. Of the systems identified in this study that do explicitly include the user in the design and creation of the objects, none explicitly incorporates Gestalt principles in their design approach. In Chapter 6 I apply the design science research methodology to develop a tangible programming environment in which the user is both the designer and creator of programming objects and in which certain Gestalt principles are explicitly incorporated in the use of the environment.

CHAPTER 5

RESEARCH METHODOLOGY

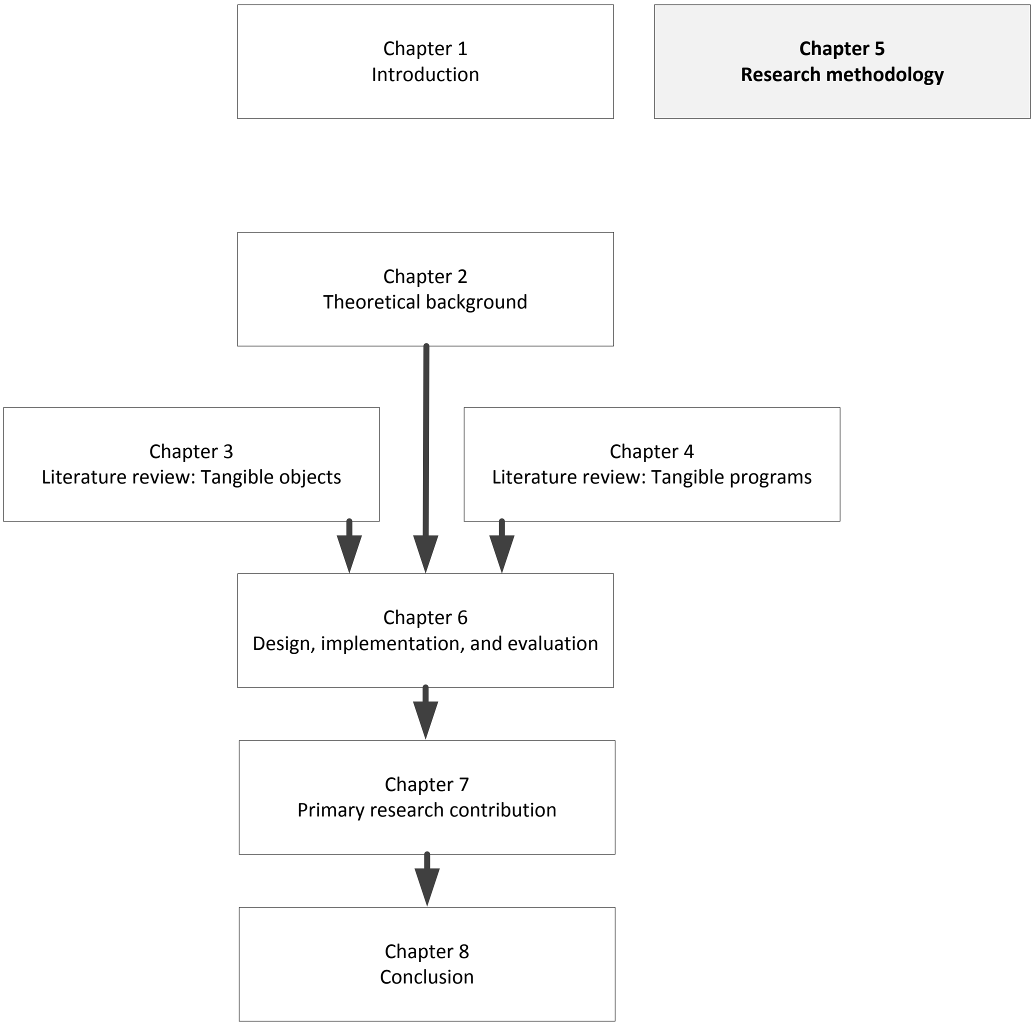

Chapter 1
Introduction
Chapter 5
Research methodology
Chapter 2
Theoretical background
Chapter 3
Literature review: Tangible objects
Chapter 4
Literature review: Tangible programs
Chapter 6
Design, implementation, and evaluation
Chapter 7
Primary research contribution
Chapter 8
Conclusion

Figure 5-1 Document structure

5.1 Introduction

A paradigm implies the assumptions and philosophical views on ontology, epistemology, and human nature that the researcher makes and these influence his chosen methodology and methods (Burrell & Morgan 2005; DeVilliers 2012; Schwandt 2007). I state my views on ontology, epistemology, and human nature in Section 5.2.

Research is guided by the chosen methodology (DeVilliers 2012) while research methods describe the tools for collecting research data. One or more research methods may be applied simultaneously to a project (Dawson 2009). In order to answer the research questions tabled in Chapter 1 (and repeated in Section 5.3.4.4), I applied Vaishnavi and Kuechler's (2008) general Design Science Research methodology and used the methods of laboratory work and direct observation. The methodology and methods are described in Section 5.3.

5.2 Philosophical stance

Research is a human activity and is hence not value-neutral; nor is research conducted without preconceptions that can influence the findings (UNISA 2007). Researchers have individualised views regarding ontology, epistemology, human nature, and methodology (Burrell & Morgan 2005). These should be considered when the results are interpreted because differing views can influence the outcome. I therefore present in the following subsections the ontological and epistemological stances and assumptions about human nature made in the design and execution of this study.

5.2.1 Ontological nature of reality

The ontological debate (Burrell & Morgan 2005; Roode 1993; Schwandt 2007) regarding reality relates to the separation between the individual and reality and is centred on the *nominalist* and the *realist* views. The nominalist view argues that the individual creates structures by defining concepts that are dynamic and recreated as the individual experiences the unstructured reality; that is, the individual adjusts the structures in an attempt to make sense of the unstructured reality. In contrast, the realist view argues that the individual and reality are separate in that the structures exist independently of the individual and that he "discovers" these structures through study.

A nominalist view therefore recognises that individuals may differ in their interpretations. The result is that a system designed with a nominalist view accommodates users with differing views, such as is the case for my T-Logo system described in Chapter 6. In contrast, a system designed with realist views is prescriptive to the user.

In this study, I considered the computer program to be a structure created by an individual that reflects his understanding of reality; therefore, the program does not exist independently of its creator. My research explored this link by focussing on the individual and considered what a programming environment could look like in which he incorporates personally meaningful elements. This study therefore assumed a nominalist view of reality.

5.2.2 Epistemological assumption

The word *epistemology* is derived from the Greek word *epistenie* that means knowledge (Blackburn 1996) and describes the branch of philosophy dealing with the theory of scientific cognition (Novikov & Novikov 2013). In particular, epistemology deals with the nature of knowledge; that is, whether knowledge can be separated from the individual investigating a phenomenon.

Irrespective of the standpoint taken regarding the nature of knowledge, scientific research remains an activity undertaken by one or more individuals and is therefore a subjective process (Novikov & Novikov 2013). According to Novikov and Novikov, the knowledge generated contributes only partially to a view of reality with additional sections of reality created by other researchers, each following their own subjective process.

Positivism and interpretivism are two research approaches in which the positivist epistemological standpoint holds that the observer can be separated from the object being studied and that the study results will be objective. In contrast, the interpretivistic epistemological standpoint states that reality is relativistic; that is, reality is studied from the individual’s point of view and can therefore not be objective (Roode 1993) as it is due to the observer’s subjective point of view that the world is understood (Niehaves 2007; Purao 2013). Whereas the positivist paradigm produces unbiased results that other scientists can reproduce, interpretivism is mediated by the researcher and produces findings that may not be reproducible (DeVilliers 2012).

Both the positivist and interpretivist epistemological standpoints contribute to the body of knowledge by studying (what is assumed to be) an existing and unchanging world. A third and complementing perspective is Design Science Research (Vaishnavi & Kuechler 2013). This perspective contributes to the body of knowledge through its continuous influence and subsequent change of reality as the research project evolves (Purao 2013). The artefact creation process is a specific contribution integral to the Design Science Research process. I apply both the interpretivistic and Design Science Research perspectives to this study.

5.2.3 Human nature

Since my research considered how the user interprets and manipulates tangible objects, it was worthwhile to look at the assumptions one could make regarding human nature as it relates to the link between humans and the environment in which they find themselves. The deterministic and the voluntaristic views are two extreme stances of this relationship (Burrell & Morgan 2005).

The deterministic view of human nature considers human beings and their activities to be determined by the situation and environment within which they are located. In contrast, the voluntaristic view considers humans beings as autonomous and free-willed creators of their environment (Roode 1993). In this study, I assumed that the individual controls certain aspects of the world by creating and choosing personally meaningful artefacts and that the Gestalt principles of perception describe the way he perceives the objects.

5.3 Methodology and methods

It is not always possible to anticipate the outcome of an experiment by only applying theory. In such cases the only way to understand, develop, and improve a system is to construct and observe the system's behaviour (Simon 1996). The iterative and incremental nature of Design Science Research paradigm (Hevner et al. 2004) supports this approach. Multiple descriptions of the Design Science Research methodology exist including the Information Systems Research Framework (Hevner et al. 2004), March and Smith's (1995) framework, and Vaishnavi and Kuechler's (2008) general methodology.

This study iteratively explored what a tangible programming environment could look like. According to the American heritage dictionary of the English language (Patwell 1992), knowledge includes all that is perceived, learned, and discovered and this includes principles and theories. I embarked on this study without knowing in advance what knowledge was required to achieve my research objective. What sets Vaishnavi and Kuechler's (2008) methodology apart from Hevner et al.'s (2004) and March and Smith's (1995) frameworks and which also made it particularly appropriate to the current study is that it highlights the fact that not all knowledge that could support an acceptable solution is known at the onset of the project. As my research progressed, I identified that the research objective can be attained using Gestalt principles and Semiotic Theory. These theories and principles (discussed in Chapter 2) were therefore part of the missing knowledge. I combined knowledge acquired from the first four iterations with these theories and principles to inform the fifth and final iteration.

Therefore, guided by Vaishnavi and Kuechler's (2008) general Design Science Research methodology, I applied methods of laboratory work, observation, and direct interaction with participants. Data collected at the end of each iteration was subjected to analysis and interpretation. I applied deductive logic based on the assumptions that a) personally meaningful objects can be used to construct programs, and b) the principles of Gestalt are relevant to computer programs. Interpretation outcomes informed additional literature surveys and subsequent iterations.

5.3.1 The nature of Design Science Research outputs

In general, the output of Design Science Research is descriptive knowledge and can take form as artefacts, recommendations, constructs, models, methods, instantiations, and improved theories (March & Smith 1995; Purao 2002; Rossi & Sein 2003; Sonnenberg & Brocke 2012; Vaishnavi & Kuechler 2008). According to March and Smith (1995), an instantiation is the creation and deployment of an artefact in the environment for which it was designed. They add that instantiation, by applying intuition and experience, is possible even before the supporting constructs, models, and methods have been finalised. They put it that supporting constructs, models, and methods are then finalised by studying the artefact in its environment and this is also the approach I followed.

5.3.2 The basic activities of Design Science Research

The basic activities of Design Science Research are to build and evaluate one or more artefact versions to demonstrate its feasibility (March & Smith 1995). The artefact thus constructed then becomes the object of study and the artefact is evaluated in order to determine how well it performs. Venable, Pries-Heje, and Baskerville (2016) elaborate on the reasons for evaluating by stating (amongst others) that it should give evidence that the theory underpins the developed artefact that solves a given problem. In this study, the artefact is a programming environment and my primary research question states the problem being addressed.

The Design Science Research evaluation process prescribes that metrics defining the aim of the research be developed against which the artefact will be assessed. Research efforts cannot be successfully judged without assessing the artefact against metrics (March & Smith 1995). Section 5.3.4.4 discusses the evaluation metrics applied in this study.

5.3.3 Vaishnavi and Kuechler's process model overview

Figure 5-2 illustrates Vaishnavi and Kuechler's (2013) Design Science Research process model that I followed. The two key components in this model are *Knowledge Flow* and *Process steps*. Also shown are the cognitive processes involved in each activity and the nature of the outputs.

The Process steps comprise five sequential activities of which the first four (*Awareness of problem*, *Suggestion*, *Development*, and *Evaluation*) are repeated in a research project. I refer to these as an *iteration*. The fifth and final activity (*Conclusion*) is executed at the end of a project.

The following are the cognitive processes associated and their related activities: Once the problem has been identified (the Awareness of problem activity), a process of *abduction* leads to a design (the Suggestion activity). The design is then realised by applying a process of *deduction* (the Development activity). Following this, the artefact is evaluated against predetermined metrics (the Evaluation activity). The Conclusion activity is the final step and it reflects on all the preceding iterations by extracting operational principles and design theories.

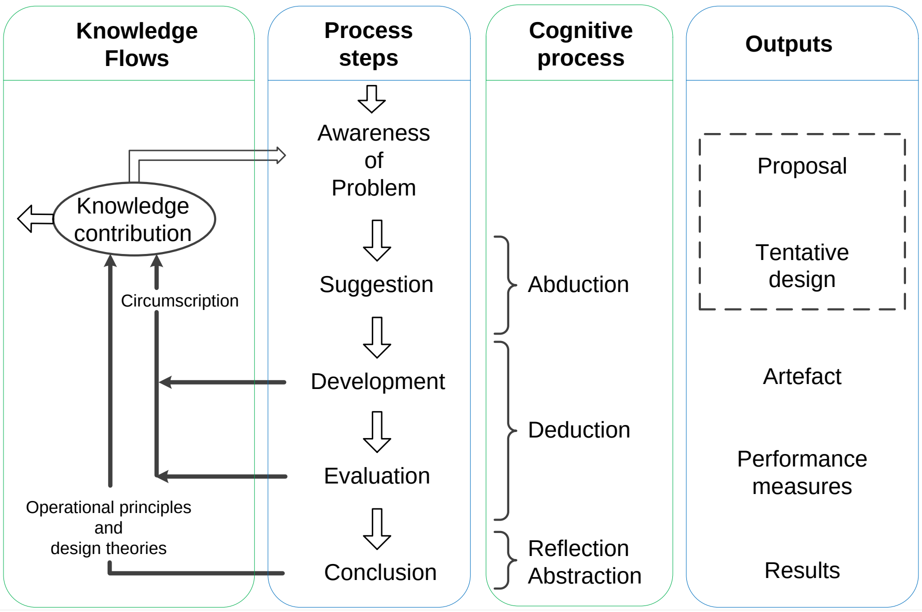


Figure 5-2 The Design Science Research process model with associated cognition

Based on Vaishnavi and Kuechler (2013)

No single knowledge base exists; instead, each profession (the Design, Architecture, and Aeronautical professions are examples) has its own base yet none of these is complete (Vaishnavi & Kuechler 2015). Knowledge bases are also specialised and well bounded (Schon 1983). In addition, a particular knowledge base may not sufficiently support a project and the researcher must incorporate methods such as experience, intuition, and trial-and-error (Hevner et al. 2004) to solve the research problem. I ultimately found theoretical support for my research problem by accessing the knowledge base used in the Psychology field.

Since knowledge bases are characterised as just discussed, I understood that my initial assumptions may later be proven incorrect and therefore some of the conclusions drawn from the completed tasks will have to be retracted. Vaishnavi and Kuechler's (2013) model explicitly includes circumscription as the mechanism available to the researcher to make reasonable assumptions regarding missing details about the state of the world. Circumscription is possible both when the analysis and interpretation of evaluation data are completed and when an iteration is terminated. I applied circumscription to the outcome of all iterations and the result informed subsequent iterations and literature surveys.

A description of the dashed line in Figure 5-2 concludes this overview of Vaishnavi and Kuechler’s (2013) process model. This line encloses the *Proposal* and *Tentative Design* outputs and illustrates the requirement of some research funding sources that a tentative design should be included with the research proposal submission (Vaishnavi & Kuechler 2013). The following section details Vaishnavi and Kuechler’s (2013) Knowledge Flow and Process steps as I applied them.

5.3.4 Vaishnavi and Kuechler’s Knowledge Flows and Process steps

I applied my intuition, experience, and the results of a preliminary literature study on tangible programming environments in the design and construction of an initial programming system. This approach is aligned with March and Smith’s (1995) observation in Section 5.3.1 regarding the origins of an instantiation. Studies of how children used the first and subsequent instantiations at science fairs informed the design of further artefacts.

Each iteration incorporated three stages. First, one or more Gestalt principles and tangible programming environment design heuristics were selected based on an awareness of the problem and on the results of earlier iterations. A system that applies these principles and heuristics was then built and evaluated. Finally, the evaluation data was analysed and interpreted and the result informed both future iterations and, in applying the principle of circumscription, informed additional literature surveys.

Figure 5-3 illustrates the generic design iteration process as applied to the current research. In this figure, I draw attention to the fact that the researcher’s knowledge base is 1) not as comprehensive as that of the research community and 2) contains knowledge not yet disseminated. This figure also highlights that new knowledge ascribed to circumscription guides ongoing literature surveys. I therefore refined my literature search between iterations when the interpreted results highlighted the need for additional information. After the fourth iteration and as part of my ongoing search for theories to underpin my evaluation results, I discovered that Semiotic theories and Gestalt principles fit my research problem well. These theories and principles are contained in the knowledge base used by psychologists.

I disseminated the new knowledge that resulted from each iteration by means of oral presentations, a working exhibition at a conference, and publications in peer-reviewed conference proceedings. The front matter of this thesis includes a list of published peer-reviewed papers. The following subsections elaborate on the methods I applied when executing Vaishnavi and Kuechler’s Process steps.

5.3.4.1 *Awareness of Problem*

In general, Vaishnavi and Kuechler's (2008) Awareness of Problem step are prompted by either a problem that surfaced or by an opportunity to apply principles from other domains to the one in which the researcher is active. At the onset of the current project, an initial literature survey revealed that no tangible programming environment model existed in which the relative positions of personally meaningful objects define a program. This problem was formulated as a research thesis statement and from this were derived a research aim, the research objectives, and the research questions. Iterations were guided by the research questions as well as new knowledge, problems, and opportunities that emerged.

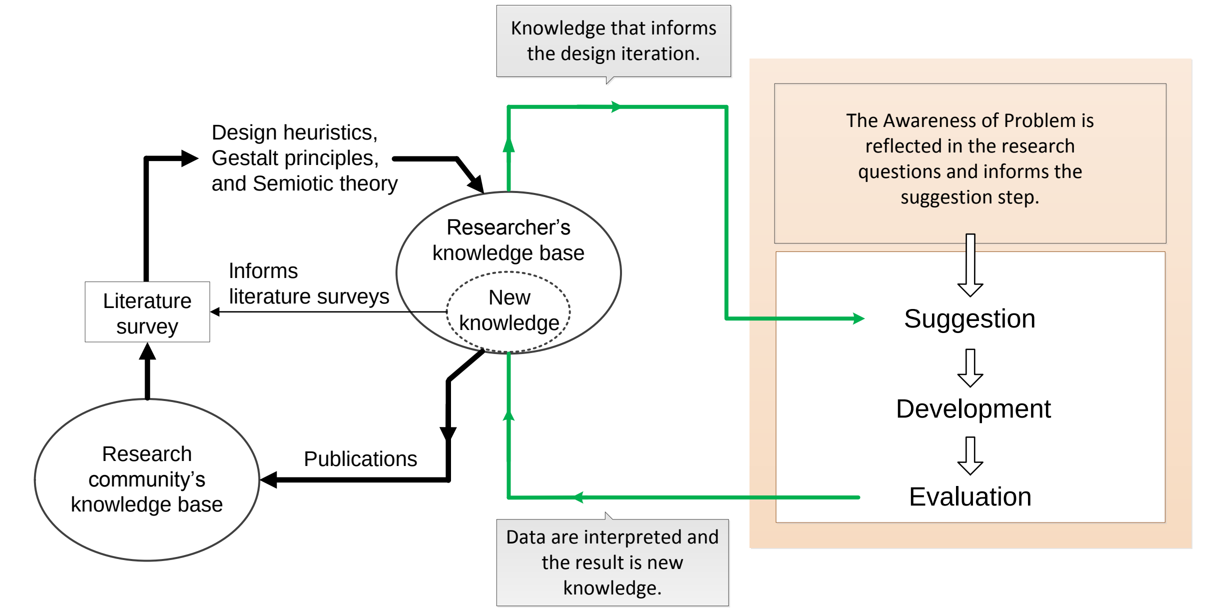

Knowledge that informs the design iteration.
Design heuristics, Gestalt principles, and Semiotic theory
Researcher's knowledge base
New knowledge
Informs literature surveys
Literature survey
Research community's knowledge base
Publications
The Awareness of Problem is reflected in the research questions and informs the suggestion step.
Suggestion
Development
Evaluation
Data are interpreted and the result is new knowledge.

Figure 5-3 The generic design iteration process as applied in the current research

Based on Vaishnavi and Kuechler (2013)

5.3.4.2 *Suggestion step*

Directed by Vaishnavi and Kuechler's (2008) Suggestion step, I studied existing knowledge to propose a solution to the identified problem. From this study emerged design heuristics for tangible objects and tangible programming environments. It also emerged that the Gestalt principles of visual perception and the theories on semiotics are relevant to the current research. These design heuristics, principles, and theories seeded the design iterations. Chapter 2 discussed the Gestalt principles and theories on semiotics while the literature on tangible objects and tangible programming environments were discussed in Chapter 3 and Chapter 4, respectively.

Drawing on Gestalt principles, this research reflected on the way in which individuals perceive objects that lie along an imaginary line and are closely spaced. The research domain of semiotics contributed to this study an understanding of the meanings an individual attaches to an object due to its shape and markings. Design heuristics extracted from literature describing tangible objects and tangible programming environments guided the selection of materials incorporated into the artefact instantiation. Finally, the sequential interpretation and execution of instructions is rooted in the research domain of Computer Science.

5.3.4.3 Development step

The Development step included the design and implementation of the artefact and was guided by the output of the preceding Suggestion activity. I alone designed and implemented all the artefacts with the exception of the first iteration. For the first iteration, I designed the artefacts and implemented these with the assistance of an undergraduate university student and a mechanical engineer. Based on the results of the Evaluation activity, the resultant artefact of an iteration was superseded by another in subsequent iterations. As Hevner et al. (2004) put it, all my artefacts embody my knowledge of the problem and my solution.

5.3.4.4 Evaluation step

Venable et al. (2016) and Hevner et al. (2004) emphasise the importance of evaluation in Design Science Research and this subsection is accordingly more detailed than the others in this section. In Design Science Research, evaluation metrics are used to measure the performance of an artefact against previously established criteria thereby providing a means to judge the progress of the research efforts (March & Smith 1995). The metrics for the final artefact have been set in Chapter 1 as a primary research question along with secondary research questions that elaborate on the former. The primary and secondary research questions are copied here from Chapter 1:

Primary research question

In the context of existing tangible programming environments and considering how tangible objects are currently used when interacting with data, what are the constructs to incorporate into a model for creating tangible programming environments in which the relative positions of personally meaningful objects define the program, and how do these constructs interact and relate to one another?

Secondary research questions

a. What program elements are suitable for a tangible programming environment in which the programmer can incorporate personally meaningful tangible objects?
b. How can a user associate personally meaningful tangible objects with program elements?
c. How can the arrangement of these personally meaningful objects be interpreted as program statements?

As Vaishnavi and Kuechler (2013) state, the initial assumptions made at the onset of a project may be incorrect. Therefore, as part of the Evaluation step I tentatively explained the reasons why the observations deviated from my initial assumptions. Additional information gained in the Evaluation step was added to my knowledge base and guided further literature surveys. Finally, the additional knowledge together with the results from incremental literature surveys informed the Suggestion activity of the subsequent iteration.

Evaluating the resultant artefact is an essential step when applying the Design Science Research methodology (Hevner et al. 2004) yet there is no agreement on how this evaluation is to be executed (Peffers, Rothenberger, Tuunanen & Vaezi 2012). Even though the Design Science Research community has not yet agreed on an evaluation method, evaluation criteria nonetheless do exist. According to Hevner et al. (2004) these criteria include the functionality of the artefact, the usability of the artefact, and the artefact's fit within the organisation. For the purpose of this research, the organisations under consideration were science fairs together with their young participants. The reasons for choosing children as evaluation participants are given later in this section. Except for the final one, I evaluated all artefacts in this context according to their functionality, usability, and fit. The artefact that emerged from the final iteration was evaluated in the laboratory against a prominent tangible programming environment and using the secondary research questions as metrics.

How an artefact gets evaluated, when the evaluation takes place, and the reason for the evaluation is project specific and is influenced by the resources available (Venable et al. 2016). Separate and independent entities hosted two science fairs primarily aimed at young people that suitably addressed these evaluation dimensions. These locations are respectively the city of Pretoria and the town of Grahamstown and are geographically separated by approximately 860km. Conducting the evaluations at geographically separated fairs was ideal since they provided access to participants from diverse backgrounds and different participants at each evaluation session. My decision to use fairs as evaluation opportunities exposed two challenges: Since I could not select the participants, no longitudinal study with a particular participant was possible and neither could I conduct inter-artefact evaluations with the same participants.

Children were ideal research participants for three reasons. First, I had easy access to a few hundred children through science fairs. Second, literature (Horn and Jacob (2006), Druin (2009), and Tarkan, Sazawal, Druin, Foss, Golub, Hatley, Khatri, Massey, Walsh, and Torres (2009) are examples) on the evaluation of tangible objects and tangible programming environments include child participants. Finally, Lucas, Bridgers, Griffiths and Gopnik (2014) and Gopnik, Griffiths and Lucas (2015) report that more knowledge may actually be a disadvantage when something new must be learned. They state that children are more tolerant to learning new concepts because they know less than adults do and are less biased by their existing knowledge.

Druin (2002) explains the four main roles that young children play in the design of new technology. These roles are as *users*, *testers*, *informants*, and *design partners*. Children assisted in the current research project as users, testers, and informants as highlighted in Figure 5-4.

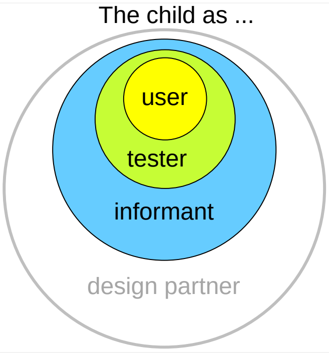


Figure 5-4 The role of children in this study

Based on Druin (2002)

Figure 5-5 illustrates the evaluation path followed in the context of Venable et al.'s (2016) evaluation strategies. Of the four evaluation strategies they describe, the current research most closely resembles their Human Risk & Effectiveness strategy. When considering their Framework for Evaluation in Design Science (FEDS), my evaluations were distributed across the axis that describes the functional purpose of the evaluation. Evaluations of the first, second and third iterations can be plotted in Figure 5-5 towards the naturalistic end on the evaluation study paradigm axis. The evaluation of the fifth artefact was conducted in my laboratory and can be plotted towards the artificial end. The fourth artefact was evaluated as a working exhibition at a conference reporting on the design of children's technology. I place this evaluation on the graph below the first three and above the fifth data point.

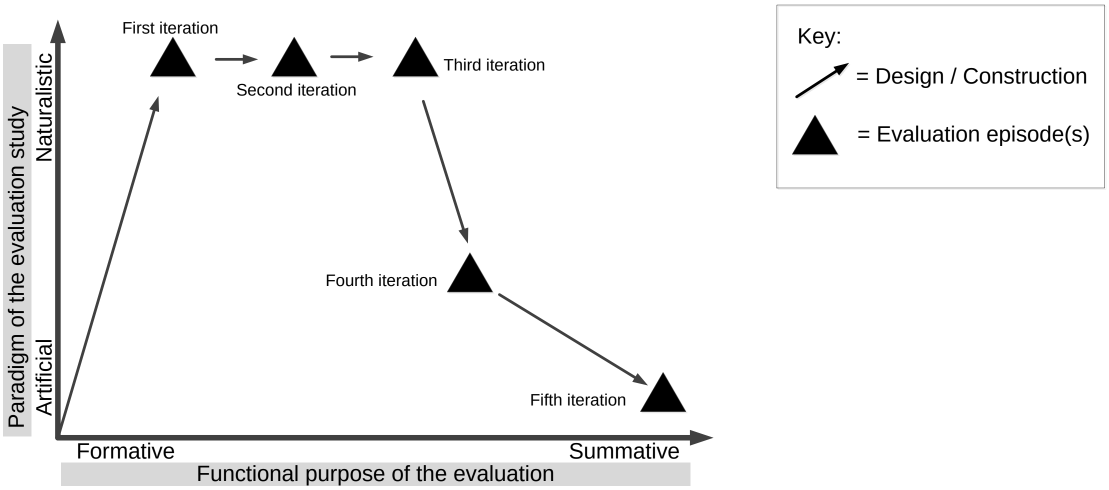


Figure 5-5 My evaluation strategy in context of the Framework for Evaluation in Design Science (FEDS)

Based on Venable et al. (2016)

5.3.4.5 Conclusion step

The Conclusion activity is the final contribution in the form of a model that captures operational principles and design theories. Iteration outcomes are included in this thesis and were published in peer-reviewed conference proceedings. The papers describe the artefact that resulted from each completed iteration and the accompanying evaluation outcomes.

5.3.5 Data collection and analysis

I collected data for this qualitative study over a period of three consecutive years during which time I conducted evaluations at two national but independent annual science fairs, two kindergartens, a technology club, a conference, and my laboratory. The collaboration with the SciFest Africa (Scifest n.d.) and ScienceUnlimited (ScienceUnlimited n.d.) event organisers was mutually beneficial: For the organisers, our evaluation workshops added to the list of attractions while my research benefited by having access to a large number of children. For my research, the fairs provided access to children of mixed ethnicity and gender and from varying and undetermined social and financial standing. During these years, I was concurrently participating in the TekkiKids (Marais, Smith & Duveskog 2007) technology club research project that also had children as participants.

Evaluation sessions conducted at the fairs were documented using both video and photographs. Sessions at the club, kindergartens, conference, and my laboratory were documented using photographs only. I archived all photographs and video recordings and later used these to compile conference papers.

For the second and third iteration, I interacted with participants and directed their activities in person. This put me in immediate physical proximity with each child. I adjusted my interaction approach based on my instantaneous observations. For example, I would offer assistance when the user was uncertain on how to proceed with an activity. At other times when the participant was clearly confident in her actions, I would retreat so as not to influence her thinking process. I could thus immediately identify design problems and confirm which design aspects worked well.

5.3.5.1 Science fairs

Thousands of children attended the two fairs each year. Both hosted a series of lectures and workshops with the number of participants at my Grahamstown workshops being approximately double that of Pretoria. Teachers, parents, and children made group and individual bookings. Two workshops per day was the norm. Participants at the science fairs were mostly school children on a school outing. Some participants were home-schooled and only a few participants were adults.

The event organisers required that workshop attendees book their places but I did not enforce this rule. Instead, I encouraged everybody to participate irrespective if they had adhered to the policy or not. Latecomers were not turned away either. A limit was set on the number of participants that could attend a lecture or a workshop. The number was based on either the size of the venue or set by the presenter. For my workshops, I set an official limit of 20 participants but I did not enforce this rule either.

Some sessions were oversubscribed yet at other times, as few as two children attended. I adapted the activities in all cases to accommodate the number of participants at a session. When only a few children attended the workshop, I adapted the activities to allow each more time to interact with the artefact. However, when a workshop was oversubscribed not all children had an opportunity to engage with the artefact or complete the questionnaires.

On average, four children per workshop session directly engaged with the artefact. With eight sessions per event per year, the average number of direct interactions was thus 64 children for each of the first, second, and third designs respectively. As many as 20 adults per year interacted with the programming environments outside of the formal sessions.

I set a time limit of one hour per workshop session. I considered this time to be a reasonable balance between mental and physical exhaustion for both the participants and researchers, and the time needed to 1) obtain written participant consent, 2) introduce the artefact, 3) for up to seven children to interact with the programming environment, and 4) for them to complete written questionnaires.

Children attended the workshops either as individuals or in groups. Some were home schooled, others attended private schools, and yet others attended public schools. Often all the participants would be from the same school while at other times the attendee composition would be a mix. I had no control over the number of participants that attended the workshops nor their age, school category, gender, or ethnicity. The ages of child participants ranged from three to 18 years.

I initially evaluated the first artefact at the Pretoria science fair. Based on the learning that emerged in Pretoria I adjusted the workshop format in anticipation of the workshops to be held in Grahamstown later that year. An assistant introduced the first programming environment to the participants and helped the children construct programs.

I evaluated the second artefact in both Pretoria and Grahamstown. For this evaluation, participants were not only users and testers but they were now also informants. As informants and before introducing them to the programming environment, I requested the participants to sketch signs of their own design that represent car motions (the first iteration had identified a need for this activity). Approximately 50 designs were legible and I classified and grouped those by hand according to similarities. Some designs were not processed as they included written text. The balance of the collected informant data are not usable for reasons that include unfinished drawings and incomprehensible markings.

For both the second and third artefacts, I interacted directly with the participants by introducing the programming environment, directing the evaluation activities, and assisting the participants as they interacted with the system. This put me in immediate physical proximity with a participant. I also interacted with him and adjusted my approach based on my instantaneous observations. Not only was I able to immediately identify design problems using this approach but also confirm which ones were good. I thus had little need to access the videos and photos taken by research assistants.

The evaluation conducted in the first iteration highlighted certain problems and my initial goal with the second analysis was to determine how well I had addressed those problems. The second artefact was therefore analysed according to three initial themes: First, I was interested in the way the user interpreted the signs. I also wanted to observe how the user manipulated the objects. The third theme was to gauge how successful the users were in constructing programs. I based my analysis of the third artefact on these themes. At the same time that I collected data on all the artefacts, I compared it with those already allocated to a theme and I created new themes when data did not fit an existing one.

5.3.5.2 TekkiKids

TekkiKids was a joint research project funded by the CSIR Meraka Institute, the University of Pretoria, and the University of Joensuu in Finland. It was modelled after the University of Joensuu’s Kids’ Club and focussed on primary school children aged nine to 12 years. The aim of this three-year project was to encourage children to follow a career in science, engineering, and technology. To this end, three groups of 12 children each were established on a voluntary basis. Six children from two schools respectively comprised one group. Participants were pre-selected by their respective schools and we requested that both well performing children and those who were not top performers be selected. Thirty-six children thus participated in the project activities. The fortnightly technology club sessions lasted two hours. In contrast to the science fairs, the children that participated in the club were selected based on their school’s recommendations.

In addition to the science fairs, the second artefact was also evaluated at the club. The second artefact evaluations and design solicitations at the club followed the same format as for the science fairs. One noteworthy difference is that the same children attended the workshops over many weeks and were thus at ease with the environment, each other, and me as session coordinator. Interactions were therefore informal and relaxed.

5.3.5.3 Kindergartens

In addition to the science fair evaluations, the third artefact was also evaluated with the help of approximately 40 children at two kindergartens. Two researchers accompanied me as we conducted evaluations in Pretoria and the Johannesburg area. The evaluation design differed from those at the science fairs in that only one evaluation was conducted per kindergarten and much more time was spent explaining the artefact to the children.

5.3.5.4 Interactive exhibition

The fourth artefact was evaluated using data collected during an interactive exhibition (Smith 2009c) at an interdisciplinary conference in Italy. I observed approximately 20 children and adults as they used the programming system.

I did my analysis along four themes. The first considered how well the magnet-based positioning mechanism functioned. I considered the design to be a success if the user placed the disk on the surface and it remained centred. A second requirement was for the disk to return to its original orientation when slightly turned. I also wanted to determine if the repurposed materials remained functional: Had any artefact part come dislodged during use the event would have been recorded as a failure. Finally, I needed to confirm that the tangible program could produce a result on the computer screen.

5.3.5.5 Laboratory with children

Another opportunity to observe how children interacted with the third artefact presented itself when a group of about 100 children of ages 16 to 19 visited our institution. Of this group, approximately 10 directly participated in an informal evaluation session while the other children looked on. Since I was the only researcher at this event, I requested the children to take photographs using my still camera. This freed me from taking pictures and allowed me to direct the proceedings and interact with a participant as he constructed a program. My close interaction gave me an opportunity to detect design problems and identify which design aspects worked well.

5.3.5.6 Laboratory without children

In the laboratory, I compared the fifth and final artefact with a prominent tangible programming environment. I did this by identifying between the two environments language structures that coincided and those that differed. I then constructed program segments using both languages.

I analysed the data according to three criteria. First, I wanted to determine the user effort required when converting a personally meaningful object into a program element. To answer this question, my initial objects were constructed using cardboard pieces cut from household product packaging. I then printed markers and attached these. The software was then put in Mapping mode and the object mapped to either an action or a numerical value. I next experimented with small wooden cubes and dowels procured from an arts-and-crafts store. The cubes and dowels are from a child's toy and fit together to form interesting structures. I again applied optical markers to the cubes. I also attached personally meaningful paper "flags" to the dowels. This and other tangible assemblies were then mapped to either functions or parameters. Finally, I constructed programs using the objects.

The second criterion was to strike a balance between the object size and the number of objects that could fit the construction surface. A balance is important for two reasons: The marker must be large enough to be identifiable using the camera and yet the objects must be small so that many would fit. I found the balance by first fixing the size of the construction surface assembly and I then determined through inspection the smallest marker that could work with the image recognition software.

Finally, I set out to determine if my artefact could be used to implement program structures of the prominent tangible programming environment. To this end, I chose instruction set elements from that environment and discovered that some constructs mapped directly to my own language and yet others had to be manipulated into a compatible structure.

5.3.6 Ethical procedures

The way in which experimental data is collected and the results reported can negatively impact humans, animals, organisations, and the physical world (Hofstee 2006; Olivier 2004; Welman & Kruger 2001). Guidelines to minimise the potentially negative impact of research are available and I list some of them here. First, research should be conducted in an honest manner (Olivier 2004). For example, research results are not to be fabricated, falsified or modified to fit the expected outcomes (Hofstee 2006). Second, human participants should not be subjected to physical harm, be embarrassed, or lose their privacy (Cooper & Schindler 2006). Cooper and Schindler (2006) also suggests that the potential benefits that the study holds and the rights of the participants should be explained to them and informed consent obtained. As shown in Appendix A, I completed a research ethics training course to familiarise myself with research ethics concepts and principles.

Cooper and Schindler (2006) define ethics as the norms that direct the researcher's moral choices. Yet, individuals may have differing and conflicting views on acceptable norms (Cooper & Schindler 2006). Also, Olivier (2004) puts it that the researcher may conduct his work with the best of intentions but at the same time not be aware of all the consequences that may follow. A researcher has an opportunity to identify and correct ethical issues by presenting the project objectives and methodology to experienced researchers. It is for these reasons (among others) that ethics committees are established at universities and research institutions (Olivier 2004).

Participants were introduced to the researcher and research assistants at the onset of each science fair workshop. They were then briefed on the research objectives and the technology used, explained the activities to follow, and what their role was. Adults who assumed responsibility for the children then gave signed consent. I also ensured that the children assented to the activities. The responsible adults were invited to watch the workshop proceedings. I ensured that a teacher was always present during activities involving kindergarten children. Appendix B contains an ethics checklist with declaration, and the ethics committee's approval to conduct the research. An example of the participant instructions and consent form is given in Appendix C.

Participants remained with the TekkiKids study for two years. The research objectives were therefore explained once with this being at the initial introductions. On this occasion, the participants also assented to the research and informed consent was obtained. The objective of a particular session would occasionally deviate from the original plan. When this happened, the changes were explained to the children at the onset of a session. The TekkiKids project manager affirms in Appendix D that the children had assented to the research. He also states in it that the parents had given their consent. Appendix E shows the invitation to participate given to the children as well as the consent form completed by the child's legal guardian.

I endeavoured to maintain participant privacy by storing data separate from personally identifying information. Participant identifying information was captured on paper and filed in my locked office. I stored data on secure servers where my login credentials and password denied access to unauthorised persons. Participant information will be destroyed simultaneously with the publication of this thesis. Images and data were anonymised before being published in research literature.

I was therefore careful to treat the participants with sensitivity and not isolate them physically from their peers. I also considered the psychological impact and the ethical implications of my actions. Finally, I kept participant identifying data confidential and strived to ensure their anonymity.

5.4 Conclusion

In this chapter, I presented the philosophical stance that underpinned my research. My ontological view was biased towards nominalism, my epistemological assumption that was biased towards interpretivism, and my assumptions regarding human nature was a balance between the deterministic and voluntaristic views. I also discussed the Design Science Research methodology and my motivation for selecting Vaishnavi and Kuechler's (2008) general methodology. I explained the methods applied to each process step in this methodology. I then distinguished between the evaluation metrics applied to the interim and the final artefacts. The inclusion of children as evaluation participants was motivated and I explained their roles as users, testers, and informants. I discussed the mechanisms by which I collected and analysed data. Finally, I motivated why ethical guidelines are necessary and how I addressed them.

CHAPTER 6

DESIGN, IMPLEMENTATION, AND EVALUATION

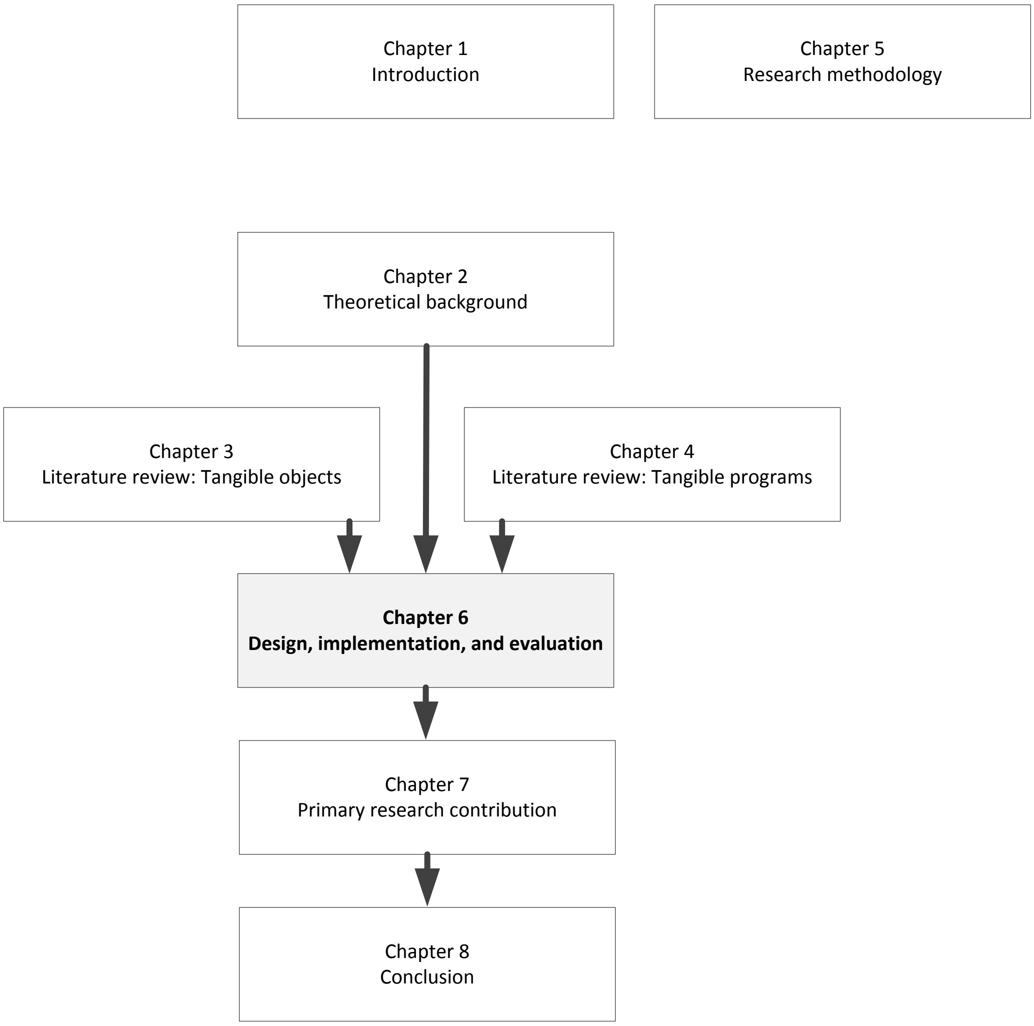

Chapter 1
Introduction
Chapter 5
Research methodology
Chapter 2
Theoretical background
Chapter 3
Literature review: Tangible objects
Chapter 4
Literature review: Tangible programs
Chapter 6
Design, implementation, and evaluation
Chapter 7
Primary research contribution
Chapter 8
Conclusion

Figure 6-1 Document structure

6.1 Introduction

In this chapter, I discuss the design, implementation, and evaluation of a series of tangible programming environments that were developed with the aim of answering the secondary research questions stated in Chapter 1. I reiterate them here:

a. What program elements are suitable for a tangible programming environment in which the programmer can incorporate personally meaningful tangible objects?
b. How can a user associate personally meaningful tangible objects with program elements?
c. How can the arrangement of these personally meaningful objects be interpreted as program statements?

I will apply the new knowledge gained here to answer in Chapter 7 the primary research question, namely:

In the context of existing tangible programming environments and considering how tangible objects are currently used when interacting with data, what are the constructs to incorporate into a model for creating tangible programming environments in which the relative positions of personally meaningful objects define the program, and how do these constructs interact and relate to one another?

The research reported here is my own. Portions of this chapter were presented at conferences and published in peer-reviewed proceedings. Page iii lists my publications since 2010. The papers published during the course of this research each addresses a particular problem. All papers were peer-reviewed and are included in conference proceedings. Smith (2008c) is an exception since it was presented at a conference but no proceedings were published. The relevant publications for the first iteration are Smith (2006, 2007a, 2007b, 2008c, 2010b) and Smith, Kotzé and Gelderblom (2011a). Results of the second iteration are reported in Smith (2008a, 2008c, 2009c, 2009d, 2010b), Smith, Foko and Van Deventer (2008), and Smith et al. (2011a). Knowledge gained in the third iteration are reported in Smith (2008c, 2009d, 2009e, 2010b, 2014b), Smith et al. (2008), Smith et al. (2011a), and Smith and Gelderblom (2013a, 2013b). Smith (2009b) gives an overview of the first four iterations. For the fourth iteration, the relevant publication is Smith (2010a). Smith (2014a) reports on the fifth iteration while my initial model for tangible programming environments is described in Smith and Gelderblom (2016).

I now describe two related projects to which I made intellectual contributions. These are the StoryBeads (Reitsma 2011; Reitsma et al. 2013; Smith, Reitsma, Hoven, Kotzé & Coetzee 2011b) research project and the Tactuslogic (Smith, Springhorn, Mulligan, Weber & Norris 2011c) tangible programming environment. The StoryBeads research project is relevant since it deals with personal and community-wide meaning that an object holds. The project outcome both confirmed and influenced my decision to use personally meaningful objects in a tangible program. Finally, the

Tactuslogic project served as a test bed for a tabletop tangible program and confirmed the artefact design approach I followed in the fifth iteration.

The design of a tangible programming environment was refined through five iterations. Figure 6-2 summarises this design progression. The first three iterations (these being GameBlocks I, GameBlocks II, and RockBlocks) interpret tangible programs and produce results that affect the physical domain. Iterations four and five (Dialando and T-Logo) produce results in the digital domain.

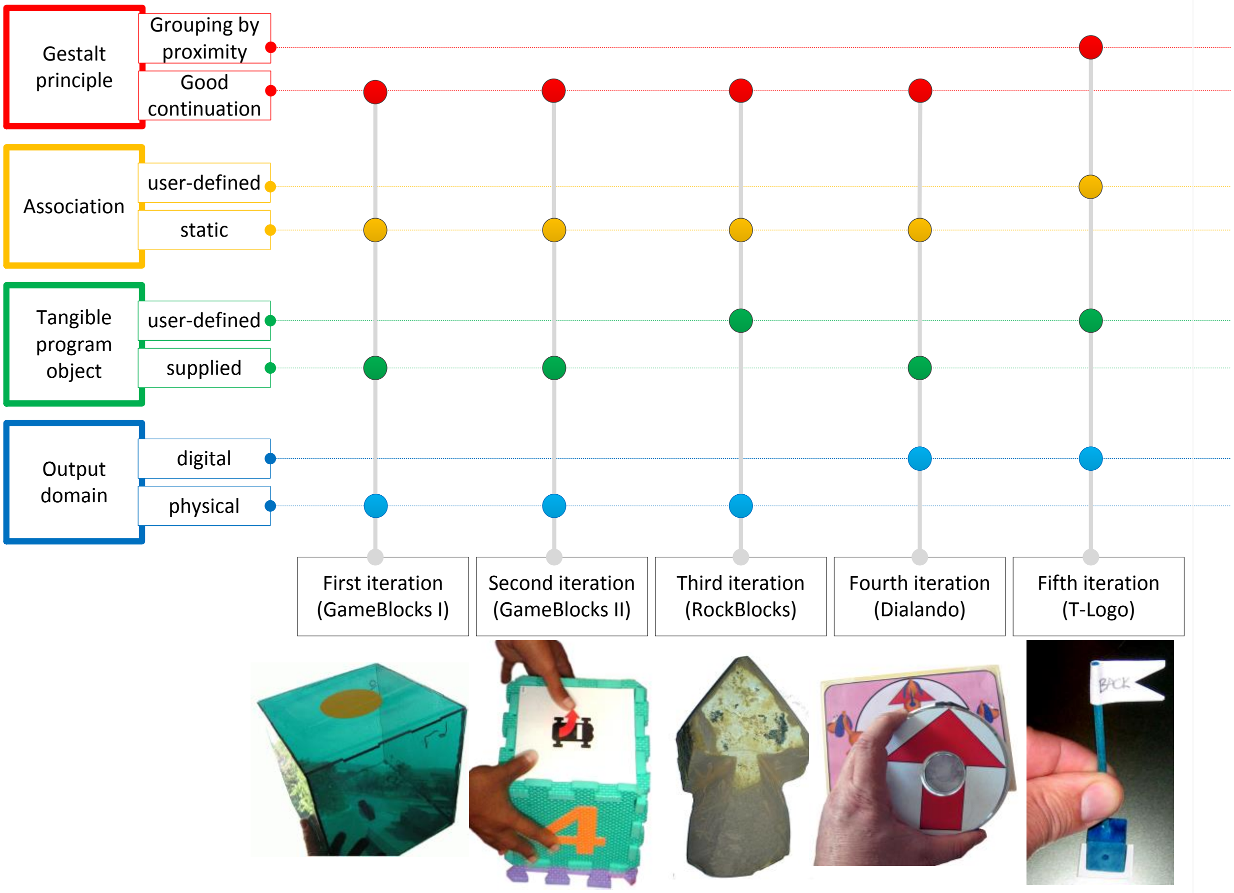


Figure 6-2 The five design iterations and their properties

Figure 6-3 graphically relates the five iterations to the section numbers in which they are discussed. This figure is based on the Design Science Research process model (Vaishnavi & Kuechler 2013) and makes explicit for each iteration the design knowledge that supported the Suggestion step and the new knowledge that emerged. I will use portions of this graphic to introduce each iteration and insert iteration-specific text in the Design Knowledge and New Knowledge blocks.

I based my artefacts on three Gestalt principles as summarised in Figure 6-3. The Gestalt principle of good continuation was applied to the first, second, third, and fourth iterations. The Gestalt principle of grouping by common region was considered but discarded in the fifth iteration. For the fifth iteration I designed and successfully implemented a tangible programming environment based on the Gestalt principle of grouping by proximity.

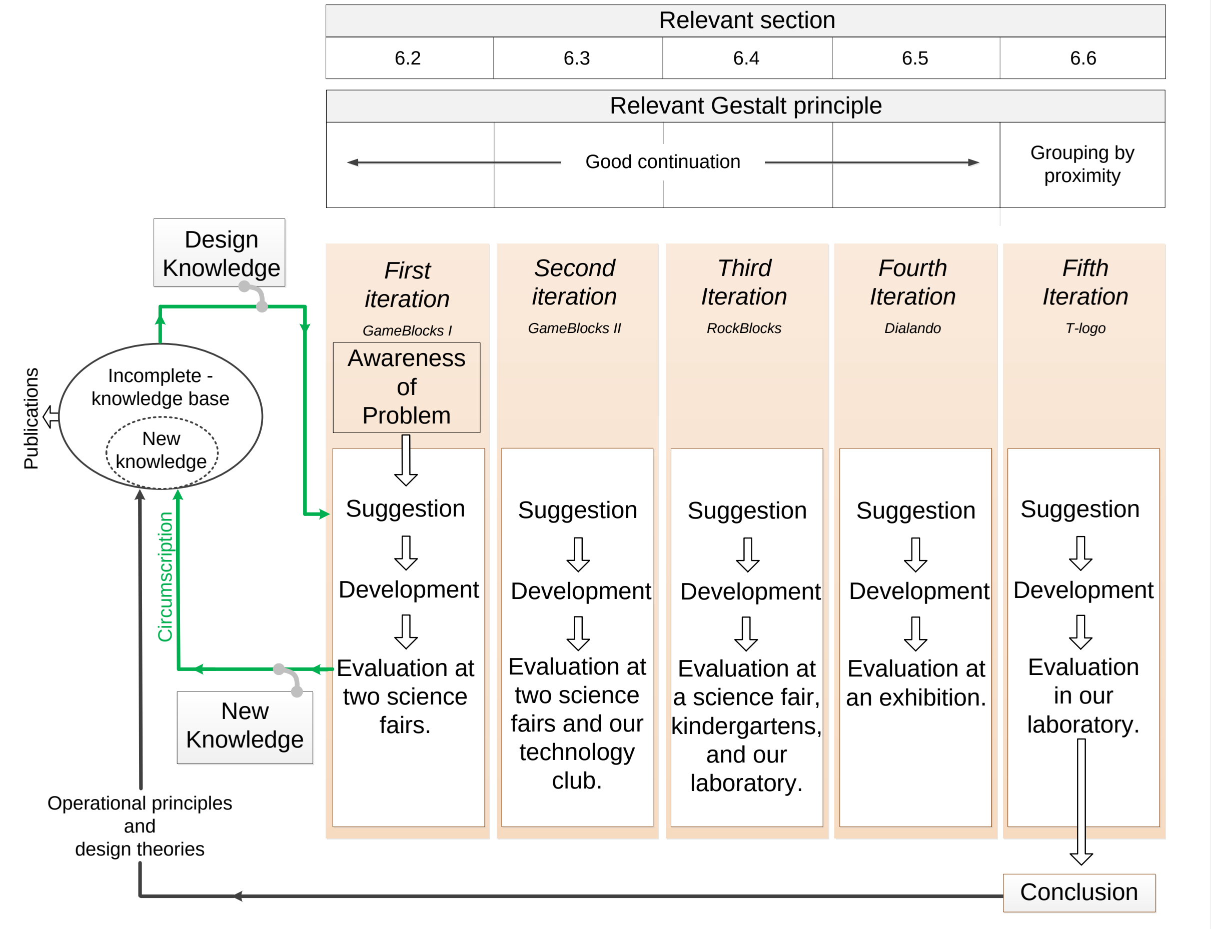


Figure 6-3 The five iterations in context of the Design Science Research methodology

Based on Vaishnavi and Kuechler (2013)

The initial design was informed by knowledge that the test participants would be children with abundant energy. I therefore designed the programming objects to be as large as possible and encourage whole body movement when the user constructed a program (Smith 2006). The result was that children had to use both hands to manipulate the object and they also had to constantly shift position around the programming area on the floor. However, evaluation results revealed that the children considered the objects as being too large. The object size was progressively reduced in consecutive iterations to the point where many objects can be held in the palm of the hand. Across the iterations, there was thus a progression from the floor-based design using large objects to a table-top version and smaller objects.

This chapter is structured as follows: The sections each describe a single iteration where Section 2 discusses the first iteration that I call *GameBlocks I* and Section 3 covers the second iteration called *GameBlocks II*. This is followed by Section 4 in which the *RockBlocks* (the third iteration) is explained.

The fourth iteration (*Dialando*) is covered in Section 5. The fifth and last iteration, called *T-Logo*, is discussed in Section 6. Section 7 concludes this chapter.

6.2 Iteration one: GameBlocks I

This was the first of five iterations aimed at answering the primary research question. The question is shown in Figure 6-4 and under the heading "Awareness of Problem". The design knowledge on which this iteration is based had emerged from the literature study and I formulated this knowledge as follows: "An arrangement of physical objects can define a program."

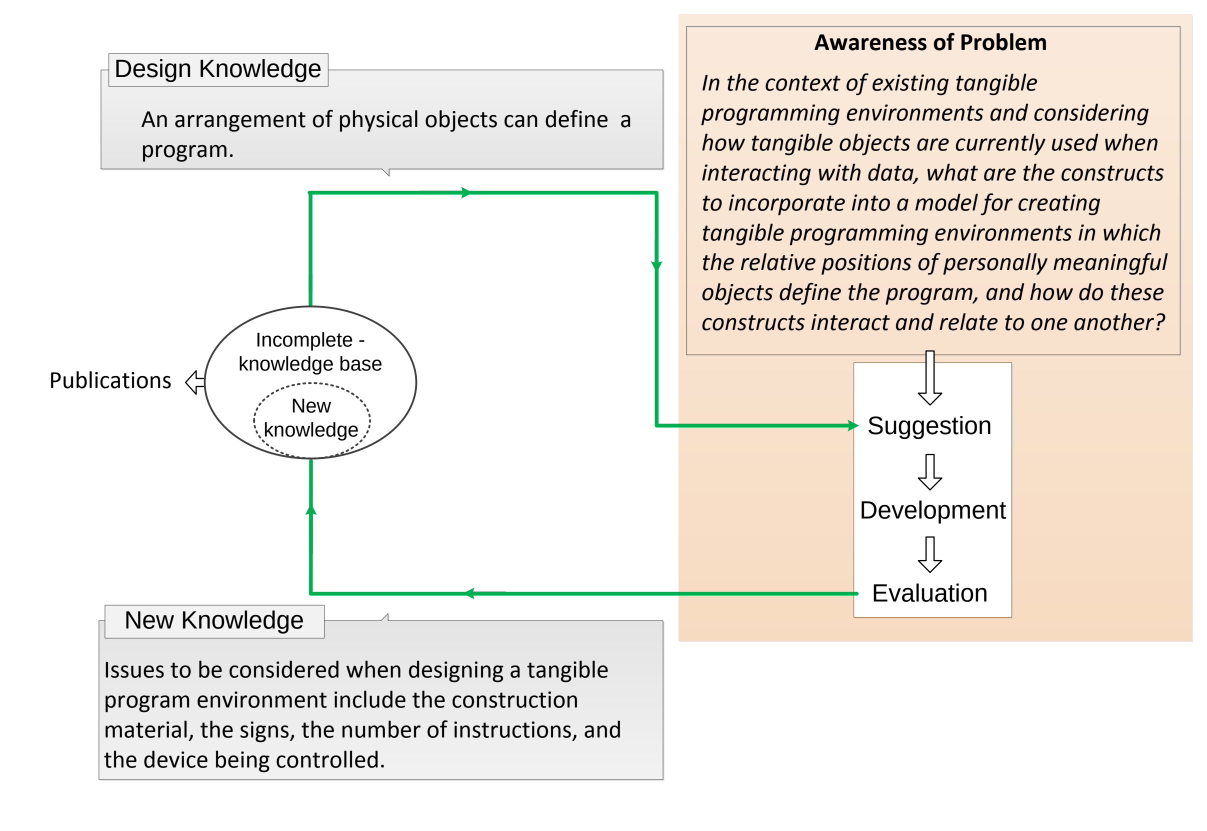
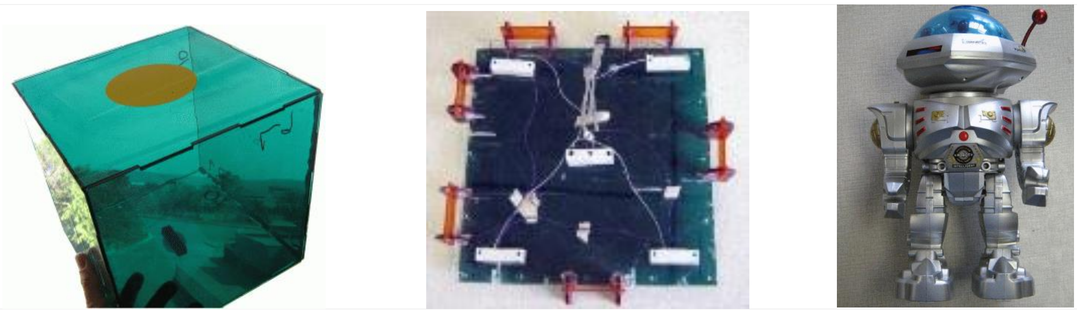

Design Knowledge
An arrangement of physical objects can define a program.
Awareness of Problem
In the context of existing tangible programming environments and considering how tangible objects are currently used when interacting with data, what are the constructs to incorporate into a model for creating tangible programming environments in which the relative positions of personally meaningful objects define the program, and how do these constructs interact and relate to one another?
Publications
Incomplete - knowledge base
New knowledge
Suggestion
Development
Evaluation
New Knowledge
Issues to be considered when designing a tangible program environment include the construction material, the signs, the number of instructions, and the device being controlled.

Figure 6-4 The knowledge that informed the first iteration and the results

To answer the primary research question I retrieved from the scientific community's knowledge base the fact that an arrangement of physical objects can define a computer program. Horn's (2009) Tern is an example of a programming environment in which an arrangement of physical objects define a program. I then proceeded to design an initial tangible programming environment that incorporates colourful acrylic cubes and a toy robot. A team of research assistants evaluated the environment under my direction and with the help of children. Evaluations were done at two science festivals and also in our laboratory.

6.2.1 System design

This artefact consists of coloured acrylic cubes, trays, a control circuit, and a humanoid robot (Figure 6-5). A program is constructed by placing cubes on the 24 pre-arranged trays in the order in which the instructions should be executed. Five magnetic sensors in each tray detect magnets in the cube. When activated, the control circuit identifies the cubes and their sequence on the trays and sends corresponding instructions to the toy robot. The colour of a cube and the sign on top of the cube represent the cube's function while a combination of magnets at the base encodes the function.



Cube Tray Robot

Figure 6-5 Cube, programming tray, and toy robot

(Smith 2007b)

The robot performs motions that include turning its head right and left, moving forward and backwards, and turning its body left and right. Figure 6-6 illustrates the range of signs (in yellow) on the cubes according to their functionality. It also shows the virtual grid that defines the location of magnets, the program function represented by a particular cube, and the colour of the cube that corresponds to that function. Red squares in this figure represent the position of the magnets in the grid. Programs with up to 24 steps are possible by combining 20 cubes and vacant spots. A tray with no cube on it delays program execution for a few seconds. Figure 6-7 is an illustration of a program that controls the toy robot. Shown below each sign is the effect the instruction has on the robot.

6.2.2 Tangibles design

The design allows for flat-pack transportation and easy assembly and disassembly using parts cut from sheets of acrylic material. An arrangement of up to five magnets inside each cube encodes the function (Figure 6-6). The placement of the magnets in Figure 6-6 was done based on the assumption that the cubes had only one valid orientation and therefore no provision was made to rotate the cubes and magnet placement symmetry is irrelevant.

All trays each have five magnetic switches that sense the magnet configuration of the cube on top. The control circuit identifies the cubes and their sequence by sensing the switches in turn. A mechanical alignment mechanism ensures that a cube is placed securely onto the tray.

6.2.3 Evaluation

This iteration was informally evaluated at the ScienceUnlimited (n.d.) fair in Pretoria (Figure 6-8a) while formal evaluations were conducted at the Grahamstown SciFest Africa (n.d.) fair (Figure 6-8b).

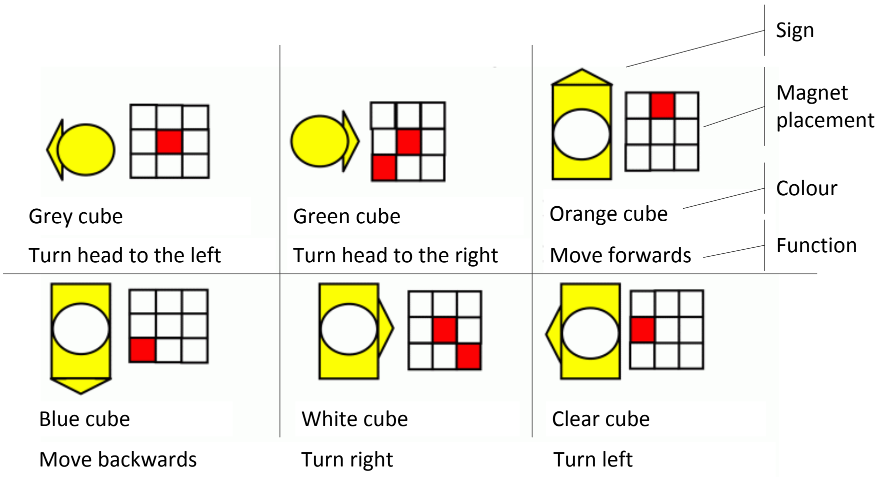


Figure 6-6 The sign, colour, function, and magnet placement for each cube type

(Smith 2007b)

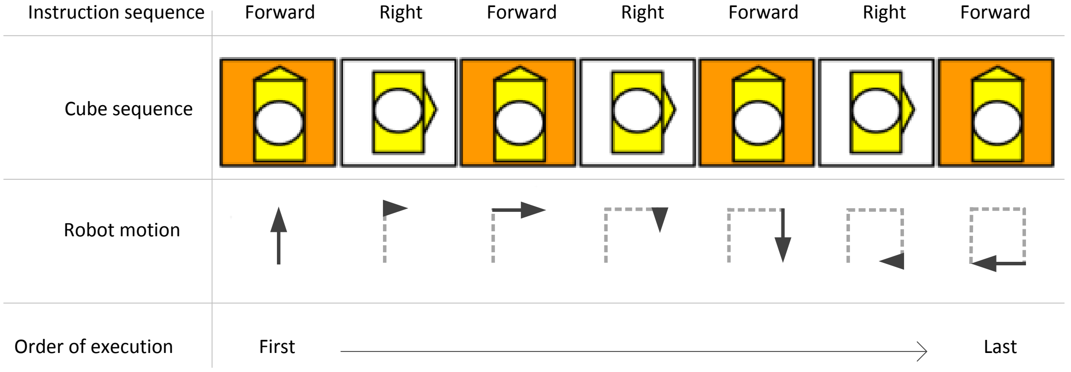
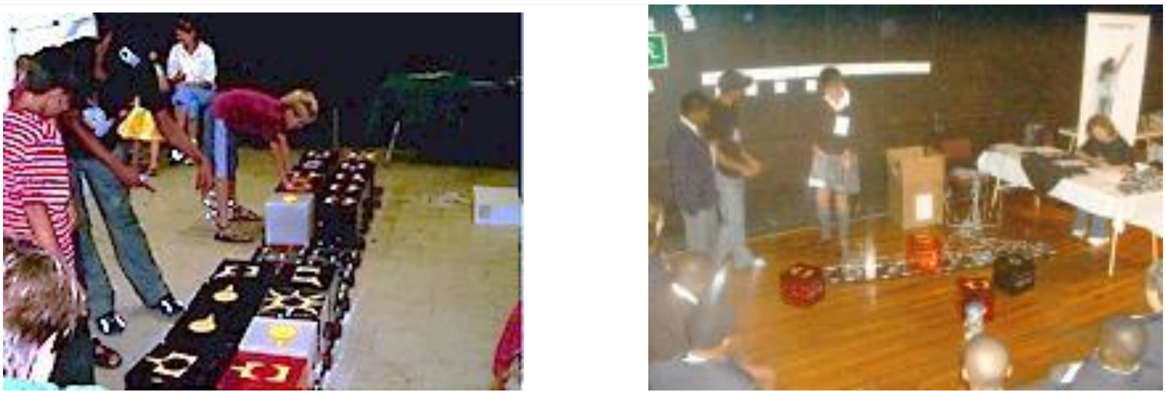


Figure 6-7 A seven-part program and the associated result of each instruction

Based on Smith (2007b)



(a) (Smith 2007b)

(b)

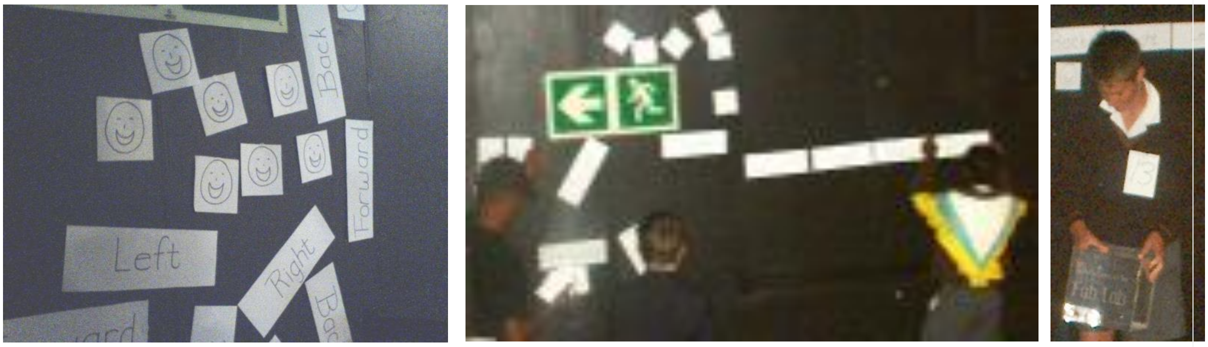
Figure 6-8 Evaluations in Pretoria and Grahamstown

6.2.3.1 *Evaluation design*

About 50 children participated in the informal evaluation sessions held in Pretoria. Then followed formal evaluation sessions in Grahamstown that involved approximately 30 children. Both types of evaluation sessions lasted approximately 45 minutes and participants were between 10 and 12 years of age. Parents or guardians (as applicable) gave written consent for all participants. Assistants used video and still cameras to capture all the evaluation sessions. Participants were asked to achieve a predetermined task by designing and constructing a program that controls the robot.

For both formal and informal evaluations, participants were given an overview of the research project at the onset of each evaluation session. The overview included explaining the correlation between the graphic signs on the cubes and robot movements. Research assistants also demonstrated the programming process by placing cubes onto the trays and activating the system to illustrate how the robot responded to the program.

Informal evaluation included observing how the participants interacted with the system and discussing the system design and operation with the children. Professional usability testers (Figure 6-8b) assisted with the formal evaluations and documented their findings and recommendations in a report (Bekker & Kruger 2006). Preparations for the formal evaluations included attaching written program instructions (Figure 6-9a) to the venue wall using sticky putty and in no particular order. These instructions were *Forward*, *Back*, *Left*, and *Right*. Facial expression icons were also randomly stuck to the wall. Finally, a sheet of paper indicating the age of a participant was pinned to his clothing (Figure 6-9c).



(a) (b) (c)

Figure 6-9 Evaluation aids and activity

Individual participants proceeded as follows: First, he arranged the sheets on the wall to imitate the desired robot movement sequence (Figure 6-9b). Copying this sequence, the participant then placed the corresponding cubes onto the trays (Figure 6-9c). He initiated program execution when he was satisfied that the cube sequence reflected the sequence on the wall. The robot then responded to the instruction represented by each cube while the participant followed the sequence against the wall and placed a facial icon below the current instruction if the robot moved as programmed.

6.2.3.2 Evaluation results

My design decision to incorporate icons but no text on the cubes proved to be problematic. Both research assistants and participants found it hard to recall what these icons represented. The colour of a cube did not help a user recall the cube's function. Instead, only the sign on the cube was of value in recalling the function. Another observation was that the physical properties of the cubes were not ideal. First, the smooth acrylic surfaces are slippery and difficult to grasp. Second, the material is fragile and breaks when dropped. Also, the corners of the cube are sharp and hard and it can cause injury if the cube falls onto an exposed foot. Finally, magnets came dislodged inside cubes. The result was that the system did not identify those cubes as expected.

The Pretoria-based evaluations were informal and the participants provided verbal feedback. Participants at this venue commented on the following: Robot sounds were not loud enough to be heard over the ambient noise. Second, the cubes were too large. The final comment relates to the cubes on the trays and the direction in which the icon on the cube points: Participants expected the robot movements to follow world co-ordinates and not the robot's own coordinate system. Put differently, participants expected the robot to move in the direction of the icon on the cube yet actual motion centred on the robot's own co-ordinate system.

6.2.4 Discussion

The following discussion first covers the informal and then the formal evaluation. I observed that 24 programming trays are too many for a novice user. The trays were configured as two parallel rows (Figure 6-8a) with 12 trays each. The program was interpreted one row at a time and from left to right. This interpretation sequence was not intuitive for either the users or me. An alternative execution sequence that I considered but did not implement is to interpret the first row from left to right and the second row from right to left. Subsequent to the informal evaluations I realised that a configuration consisting of two adjacent rows is needlessly challenging to the user and should be avoided in future iterations.

In preparation of the formal evaluation in Grahamstown and informed by the Pretoria observations, I reduced the number of trays from 24 to eight. I also simplified the configuration from two rows to a single one while keeping the interpretation sequence from left to right. Although I only became aware of the Gestalt principle of good continuation after the completion of the fifth iteration, this principle predicts that users will not have trouble in grasping the left-to-right linear interpretation of a single row of objects. The second, third, and fourth iterations adopted configurations based on this principle and the evaluation results confirm that the Gestalt principle of good continuation is applicable to tangible programs.

The physical properties of the acrylic cubes make the cubes unsuitable for public use. Also, the signs are too abstract and this problem can be addressed using a set of four descriptive signs (Figure 6-10a through d) as Bekker and Kruger (2006) suggest. In this figure, a sign consists of an arrow indicating the direction of intended motion and an outline of the toy robot. Observations also confirmed that eight programming steps are appropriate for the novice user. Some participants commented that the cubes are too large. This observation relates to the comment that the cubes are slippery. Finally, the evaluators (Bekker & Kruger 2006) suggest that a physical hook-and-loop linking mechanism (Figure 6-10e) be added as highlighted. They put it that this mechanism would make the interpretation sequence explicit.

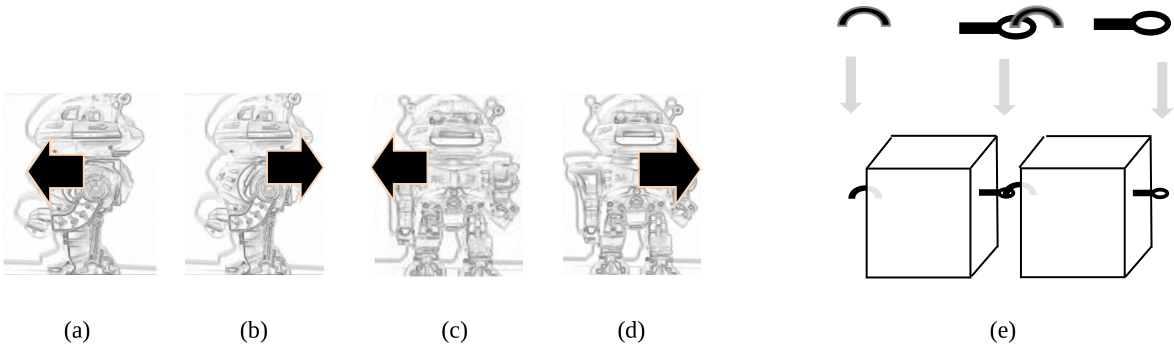


Figure 6-10 Suggested design improvements

Based on Bekker and Kruger (2006)

Based on the results of this iteration I added to my knowledge base the following factors to be considered when designing tangible program environments: The sequence in which objects are interpreted should be unambiguous and the construction material should be chosen with care. Also, the design of signs should consider the user and the number of instructions should match the time available to the user to become familiar with them. Finally, the device under control should execute clearly recognisable actions.

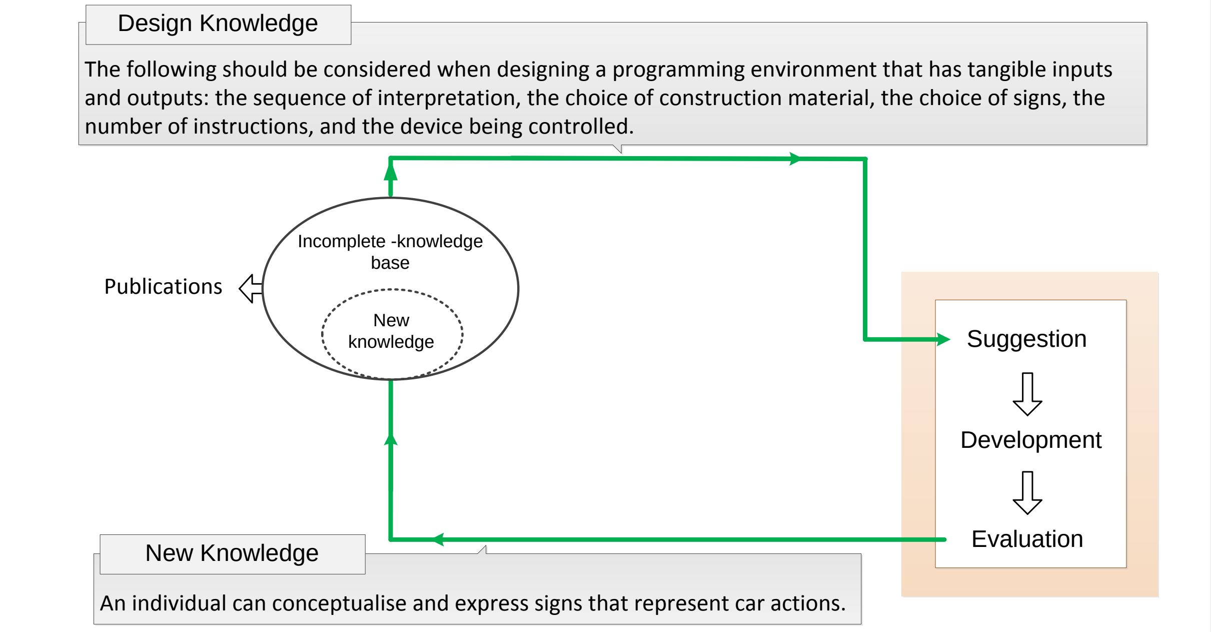
The second design iteration incorporates a new sign set and smaller cubes. Compared to the acrylic cubes, these cubes are easy to grasp, lighter, smaller, and made using soft foam. They are also safe for public use since they will not shatter when dropped. Finally, I replaced the toy robot with a toy car that has exaggerated movements.

6.3 Second iteration: GameBlocks II

Evaluation results from the first iteration indicated that the choice of construction material, signs, number of instructions, and the device being controlled should all be considered when designing a programming environment that has tangible inputs and outputs. Figure 6-11 captures these indicators in the block marked Design Knowledge and I used them to design the second artefact.

6.3.1 System design

This design incorporates the new knowledge that emerged from the first iteration: First, the signs are now more descriptive. Also, the cubes are smaller, easier to manipulate by hand and safe for use by children. Finally, the instruction set is smaller and confusing instructions have been removed.

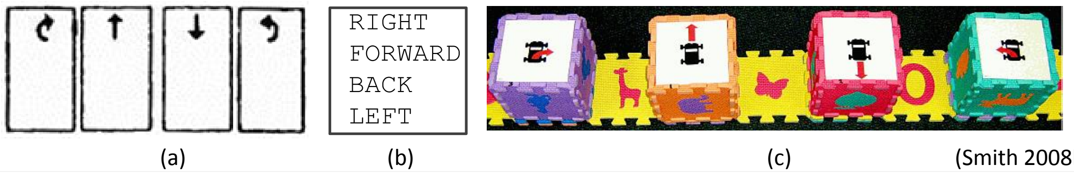


The following should be considered when designing a programming environment that has tangible inputs and outputs: the sequence of interpretation, the choice of construction material, the choice of signs, the number of instructions, and the device being controlled.

An individual can conceptualise and express signs that represent car actions.

Figure 6-11 The knowledge that informed the second iteration and the results

The choice of signs and overall design of the second iteration were guided by signs and designs in the literature: The Tortis (Perlman 1976) action card arrows and the TURTLE TALK (Papert 1980) text in Figure 6-12 (a) and Figure 6-12 (b) are two programs with comparable outcomes. Shown in Figure 6-12 (c) is a sequence of my cubes with signs that are based on the TORTIS arrows and the Turtle Talk text. The cube sequence represents a program similar to those in Figure 6-12 (a) and Figure 6-12 (b).



(a) (b) (c) (Smith 2008b)

Figure 6-12 Programs constructed using Tortis, Turtle Talk, and GameBlocks II

The system incorporates two “mats”. The *programming mat* serves as the surface on which the program is constructed and it is composed of eight sensing squares that are interspaced by one or more passive foam squares. Together these form a linear programming surface that can be adjusted in length. A second, “execution” mat has an embedded pathway and the user creates a program that directs the toy car along the path. All tangibles and mats are constructed using modular foam squares and the user can easily change the configuration. For example, the user may interchange squares to produce cubes of varying colour and height.

This iteration defined six language elements and these are *move forward*, *move backward*, *turn to the left*, *turn to the right*, *play the first tune*, and *play the second tune* (Figure 6-13 a through f). The toy car moves a distance of one square for each forward or reverse instruction. A turn instruction causes the car to rotate 90 degrees on the spot and then stop.

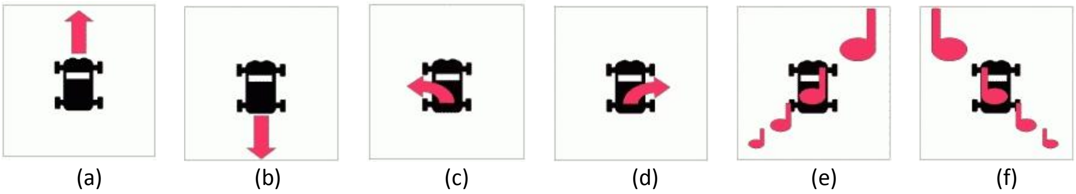
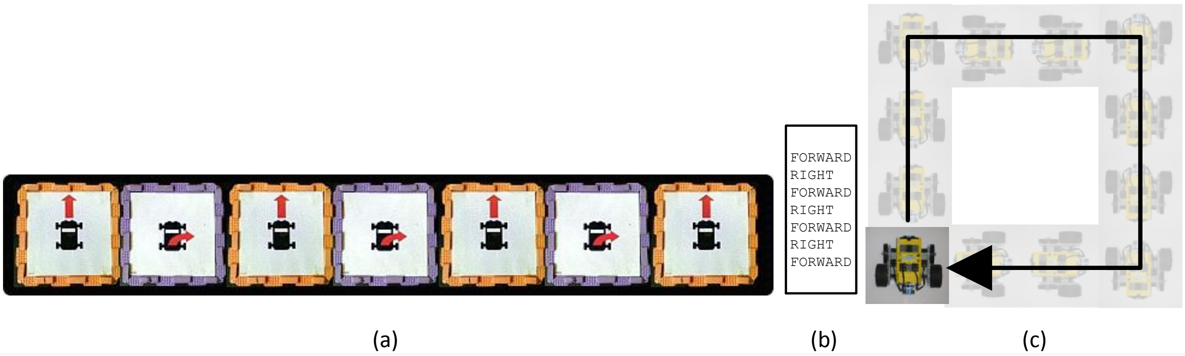


Figure 6-13 The sign set that defines the language elements

Programming involves placing cubes on the programming mat in a linear sequence and this row is then interpreted from left to right. Figure 6-12 (c) is a photograph of a program that instructs the car to turn to the right, then move forward, move backward, and finally turn to the left. Figure 6-14 illustrates a program (a) with its TURTLE TALK equivalent (b) and the result (c) when the program is executed.



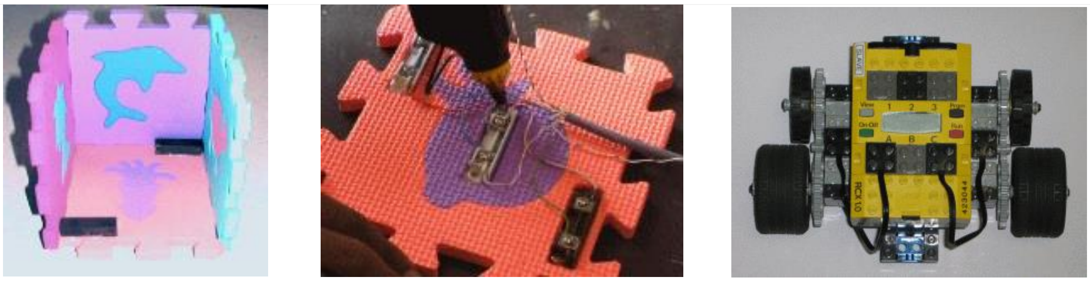
```
FORWARD
RIGHT
FORWARD
RIGHT
FORWARD
RIGHT
FORWARD
```

Figure 6-14 A program, its TURTLE TALK equivalent, and execution results

Based on Smith (2009c)

6.3.2 Technical description

Cubes have soft foam sides and each contains between one and three magnets that are fixed along a diagonal line at the base (Figure 6-15a). The magnet configuration uniquely identifies the cube's function to the electronic circuit and the sign (Figure 6-13) on the top serves to identify the function to the user.



(a) (b) (c)

Figure 6-15 Foam cube, active square under construction, and toy car

The programming mat is a combination of *active* and *passive squares* where the passive squares serve as aesthetic and structural elements. Active squares (Figure 6-15b) each contain three magnetic switches that are sandwiched between two foam squares. Electronic circuitry can identify the cube based on the magnet arrangement inside the cube. This circuitry is also connected to a Lego RCX "brick" (Knudsen 1999).

The system interprets the line of squares from left to right and transmits one of six signals to the car (Figure 6-15c) corresponding to the cubes on the squares. The six signals represent each of the six language elements. Software executing in the car receives the signal and activates the electric motors and loudspeaker in response. The car then either adjusts its position, orientation, or plays a musical tune as is appropriate for the message received.

6.3.3 Evaluation

Formal evaluations provided insight to how well the artefact served as a tangible programming environment. The evaluations were integrated with workshops held at ScienceUnlimited (n.d.), SciFest Africa (n.d.), and TekkiKids (Marais et al. 2007): ScienceUnlimited in Pretoria and the Grahamstown based SciFest Africa are regional science fairs targeting the youth. TekkiKids was a multiyear research project that studied children's interaction with technology through workshops in our laboratory and at a local school. Appendix F shows the workshop invitation extended to children at the SciFest Africa workshops in 2008.

6.3.3.1 Evaluation design

Workshop participants were school-going children, of mixed gender, mixed ethnicity, and undetermined social class. The workshops held at our laboratory and the school were part of a separate 24-month long research project conducted in collaboration with 10 to 13 year old participants. This research involved 36 children with six participants from each of four schools and 12 children from a fifth school. The participants from the four schools visited our laboratories twice a month, being two schools per visit. Interaction with the participants of the fifth school was on the school premises. Workshops at the science events were attended by groups ranging in size of between two and 20 children and aged four to 17 years.

At the start of each science festivals workshop the research assistants solicited written consent from the parents and guardians who were legally responsible for the children. The same was done for the participants who were involved in our 24-month research project. The solicitation served to inform the parents, guardians, and participants of their rights. Appended E is an example of the consent form. The collected data included completed questionnaires, video recordings, and photographs. The video recordings and photographs informed our evaluation analyses.

I introduced the participants to the programming environment and explained how the artefacts should be selected and positioned on the programming mats. The artefact components are the toy car, cubes, programming mat, and the execution mat. The functionality of each cube type was explained and demonstrated using the car.

User responses were elicited during evaluation sessions using two questionnaires. The questions are included in this thesis as Appendix G and Appendix H. The first questionnaire (Appendix G) is associated with the workshops conducted in our laboratory. The second questionnaire (Appendix H) supported the workshops conducted at the two science festivals. At the onset of a workshop, the participants completed Appendix G (Section A) and Part 1 of Appendix H. Section C of Appendix G and the third part of Appendix H solicit design inputs on sign language element representation. Appendix G (Section B) and the second part of Appendix H were completed at the end of each workshop. Approximately 15 minutes were allocated to this activity.

Two primary activities formed the basis of evaluation workshops. First, the participants had to solve two challenges using the artefact. For the second activity, the children suggested alternative language element representations. The following paragraphs elaborate on these activities.

The two challenges required the participants to design and construct programs to guide the car to two rewards along a fixed route. The rewards are marked “Target object #1” and “Target object #2” in Figure 6-16 (a) and Figure 6-16 (b), respectively. These targets were toys for the participant to keep. The toy served as both a concrete programming objective and a token of our appreciation for participating. The two challenges differed in the way that the car moved. For the first challenge, the only requirement was for the car to reach both targets. For the second challenge, the car had to reach Target object #2 by reversing. Figure 6-16 (c) and Figure 6-16 (d) illustrate two solutions to these challenges. Participants were encouraged to design solutions using printed copies of cube signs called *programming aids*. Programming aids are printed copies of the sign set (Figure 6-13) and Figure 6-16 (a) shows them in use.

The programming activity went as follows: First, a participant was given a set of programming aids while at the same time I placed all the available programing cubes close to the programming mat and in no particular order. The child then placed the aids on the floor in the sequence (Figure 6-17a) that she envisaged would solve the challenge. I then helped her mentally execute each instruction in the compilation to validate the program against the set objective. When the participant was satisfied with the program design, she copied the design onto the programming mat by placing (Figure 6-17b) appropriate cubes onto corresponding active squares. The system was then activated and the car's motions closely observed.

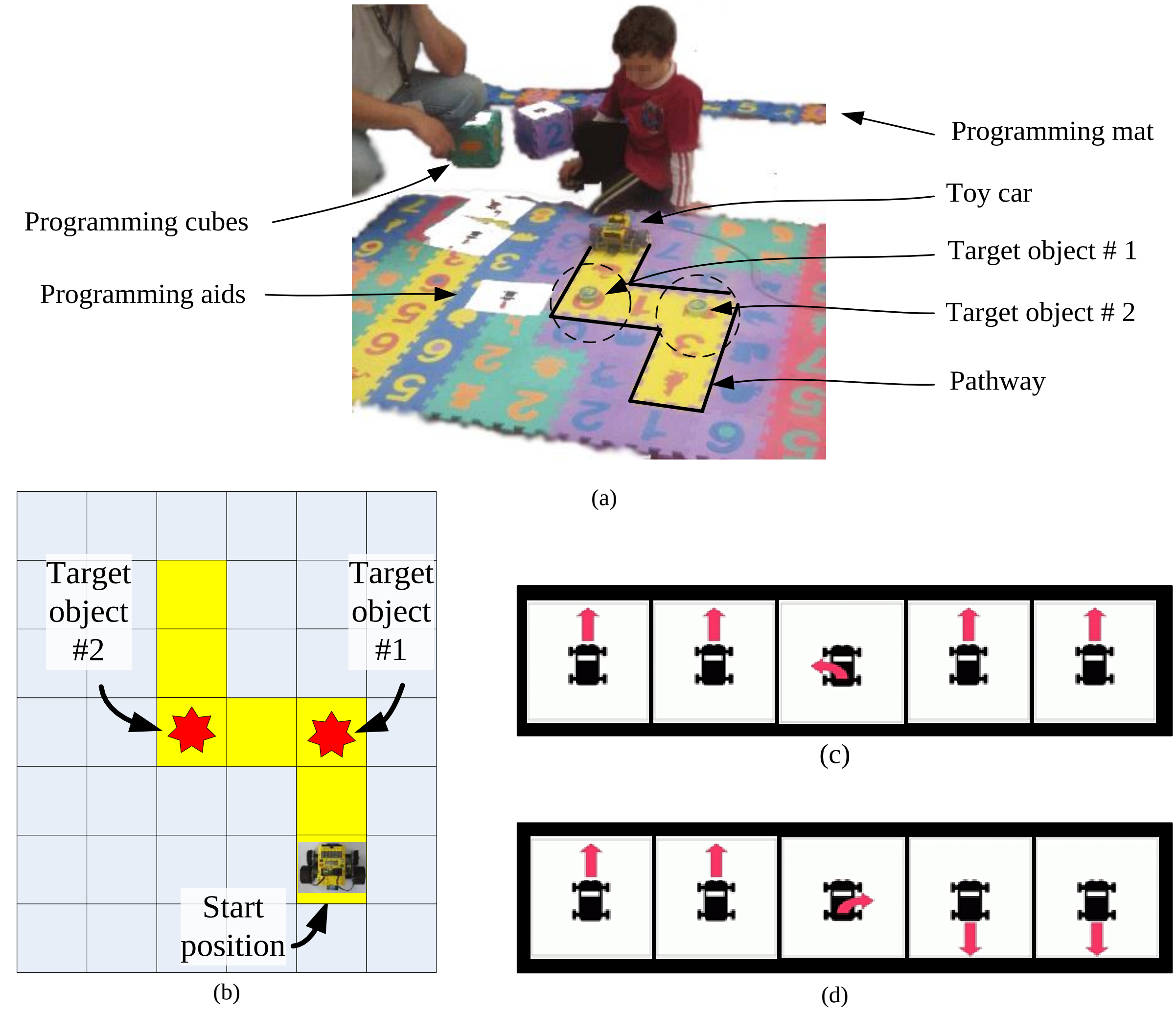


Figure 6-16 The physical configuration, the challenge, and two solutions

Based on Smith (2009c)

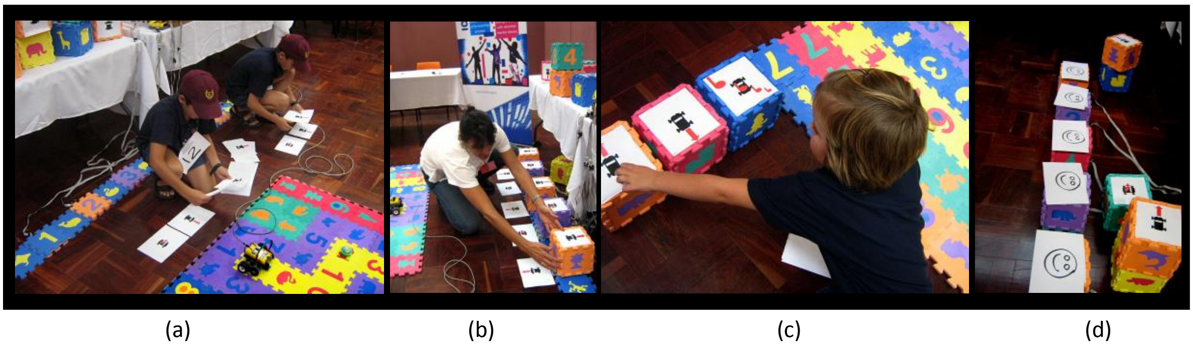

(a) (b) (c) (d)

Figure 6-17 Program design, construction, and debugging activities

Program debugging was done while the program ran. As the program executed, the user was encouraged to point (Figure 6-17c) at the cube being interpreted and simultaneously check if the car's movement corresponded to the cube sign. A research assistant placed a smiling face icon on top of the cube (Figure 6-17d) when the car behaved as expected. When the execution did not correspond to the user's intentions the discrepancy was resolved by inspecting the cube sequence and comparing the sequence to the car movements. The program was then modified, the car repositioned to the start of the route and the system reactivated. The debugging process was repeated until the car behaved as envisaged.

Once all the children have had an opportunity to construct and execute their programs they were asked to suggest alternative signs. They were requested to draw pictures (without text) to illustrate the following car motions: "Move forwards and keep on going", "move forward and stop", "turn right and stop", and "turn right and keep on going". Copies of the activity worksheets are in Appendix G (Section C) and Part 3 of Appendix H.

6.3.3.2 Evaluation results

Figure 6-18 and Figure 6-19 show some of the participants' suggestions. The diversity in the suggestions supported my emerging thinking that users can design their own signs. Three design themes emerged. First, abstract signs such as a dot or vertical bar are at times used to indicate a stopping action (Figure 6-19c and d). Other abstract signs include dotted lines and multiple parallel lines to indicate that the car continues its current motion (Figure 6-18b and c). Second, concrete signs as found on the road side or tarmac, and hand signals indicate the intended action (Figure 6-18e and Figure 6-19a,b,e,f and g). Finally, a combination of abstract concepts and concrete objects were suggested and this is exemplified in the sketch (Figure 6-18a) as a car that changes shape as it passes through an intersection.

6.3.4 Discussion

Workshop observations established that individuals are able to conceptualise and express signs that represent car actions. A significant number of the suggested signs are similar to those already incorporated into my artefact. The following may help explain this phenomenon: First, the participants were not motivated to generate original suggestions. Second, the participants did not understand the task and opted to imitate my signs.

The program element signs in this iteration are an improvement over those used in the first iteration. Substituting the acrylic programming mat and cubes with foam material was another refinement that yielded positive results.

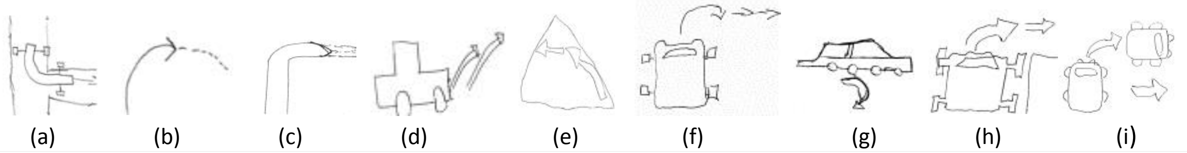


Figure 6-18 "Turn-and-go" signs

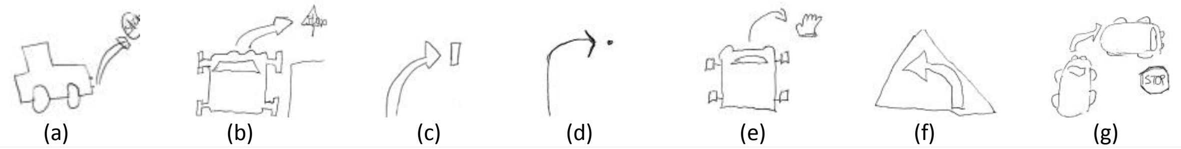


Figure 6-19 "Turn-and-stop" signs

6.4 Design iteration three: RockBlocks

Experience gained using abstract signs in the first and second iterations together with visual perception tests conducted with the help of young children (Smith et al. 2008; Smith 2009a, 2009e) led me to conclude that a person may not always interpret signs as the sign designer had intended. Insights from the second iteration indicate that an individual can conceptualise and express signs to represent car actions. The problem of sign interpretation combined with the individual's ability to conceptualise and construct signs (as determined in the second iteration) prompted me to investigate the use of personally meaningful artefacts as programming elements. I therefore explored if, and how, user-created artefacts can be appropriated as programming objects. My findings (Smith & Kotzé 2010) were encouraging and based on these I extended my research to include hand crafted objects as programming elements.

Concurrent to this iteration, I also advised on a related study (Reitsma et al. 2013; Smith et al. 2011b) that had an objective to determine the extent to which a rural African community would embrace objects that integrate digital technology with the community's handcrafted artefacts. What emerged from that study and also of particular interest to my own is the fact that rural communities construct artefacts that hold community-specific meaning. Examples (Smith & Kotzé 2010) of artefacts that hold personalised meaning include the Zulu fertility doll and beaded apparel (Figure 6-20).

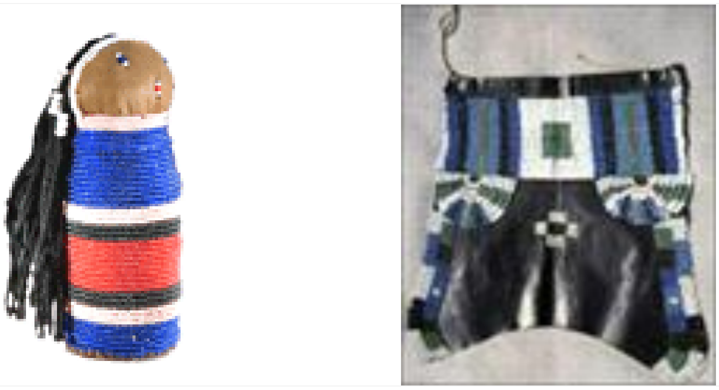


Figure 6-20 Zulu fertility doll and beaded apparel

(Smith & Kotzé 2010)

6.4.1 Design considerations

Results from the second iteration indicated that a user could conceptualise and produce hand-drawn signs to represent the actions of a toy car. I investigated in the third iteration whether a tangible program environment could be designed and implemented in which the user crafts personally meaningful program elements. Objects constructed in the first and second iterations incorporated only artificial materials. For the third iteration I wanted to explore the application of natural materials such as those used in stone sculptures (Colledge 1979; Patton 1985).

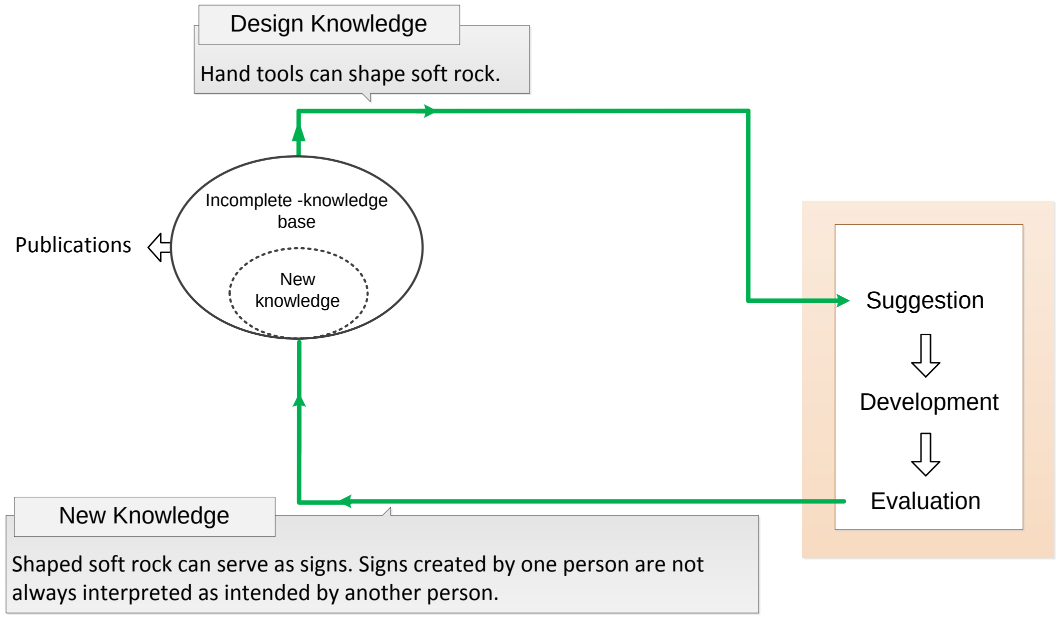

Design Knowledge
Hand tools can shape soft rock.
Publications
Incomplete -knowledge base
New knowledge
Suggestion
Development
Evaluation
New Knowledge
Shaped soft rock can serve as signs. Signs created by one person are not always interpreted as intended by another person.

Figure 6-21 The knowledge that informed the third iteration and the results

As I show in Figure 6-21, the knowledge of applying hand tools to shape soft rock influenced the design of the third artefact. I therefore chose soft rock as the medium in which to craft personally meaningful objects (Smith 2009b). If successful, this environment would fulfil Research Objective 2 (copied here from Chapter 1) that states:

Research Objective **2**	To devise a mechanism by which a personally meaningful tangible object can be used as a program element.

6.4.2 System design

The RockBlocks tangible programming environment is comprised of handcrafted signs, electronic circuitry, sensing tiles, and a toy car. I shaped arrows (Figure 6-22b) from soft rock using hand tools (Figure 6-22a) and mounted them onto wooden squares. Two magnets were also embedded inside each square (Figure 6-22c top) in an ‘L’ shape. In contrast to the objects in the first and second iterations where each object type is unique, all third iteration objects are the same and differ only in their appearance and the way the user orientates them on the Sensing Tiles.

Figure 6-22c (bottom) is a photograph of painted wooden blocks called *Sensing Tiles.* Each tile embeds three magnetic sensors in a ‘T’ configuration and every sensor is connected to circuitry that determines the orientation of the program object placed on top. The circuit also determines the sign sequence by decoding data received from all the Tiles.

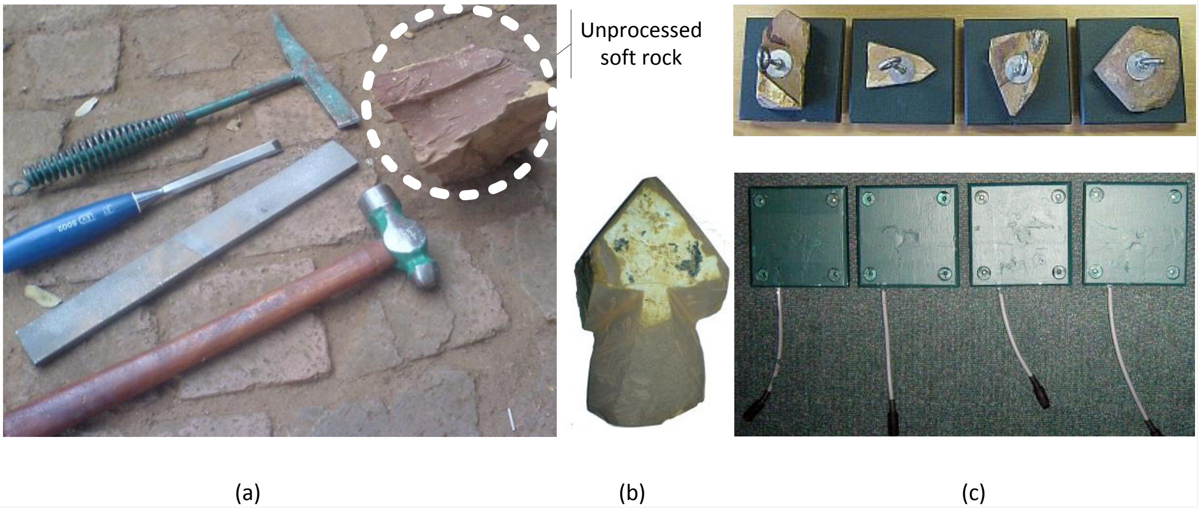


Figure 6-22 Natural rock and hand tools, the processed rock, programming objects, and Sensing Tiles

Based on Smith (2009a)

The user constructs a program by placing signs on the tiles in the desired execution sequence. When activated, the electronic circuitry sends corresponding signals to the car for immediate execution.

6.4.3 Tangibles design

Program elements are comprised of arrows that I have previously carved from natural rock and mounted onto wooden substrates. An eyebolt with nut secures the processed rock and serves as a handle with which the user can manipulate the element. I call the assembly consisting of rock, wood, magnets, and bolt, a *RockBlock*. The logical significance of this tangible program object lies in its orientation relative to the Sensing Tile on which it rests. The set of RockBlocks language signs is defined by the four directions in which a RockBlock may point. The four directions are relative to the user and as follows: pointing away from the user, pointing towards the user, to the left, and to the right. The signs represent the following corresponding logic functions: move forward, move backward, turn to the left, and turn to the right.

I next elaborate on the signs and their effects by means of Figure 6-23. When the object is placed on top of the Tile in such a way that the object points away from the user, it signifies that the car should move forward. Conversely, if the arrow points towards the user, it denotes that the car should reverse. A sign that points to the left signifies that the car should turn 90 degrees to the left. The result is similar for a RockBlock pointing to the right.

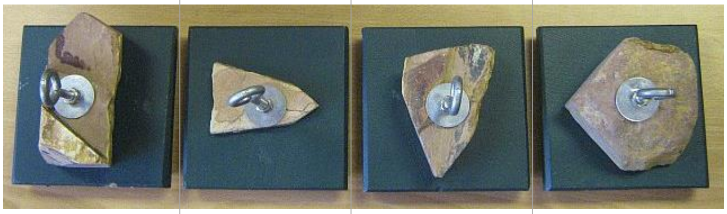
Program instruction	Move forward	Turn right	Move backward	Turn left
Program object				
Interpretation sequence	1st	2nd	3rd	4th

Figure 6-23 A program that exhibits the sign set

6.4.4 Evaluation

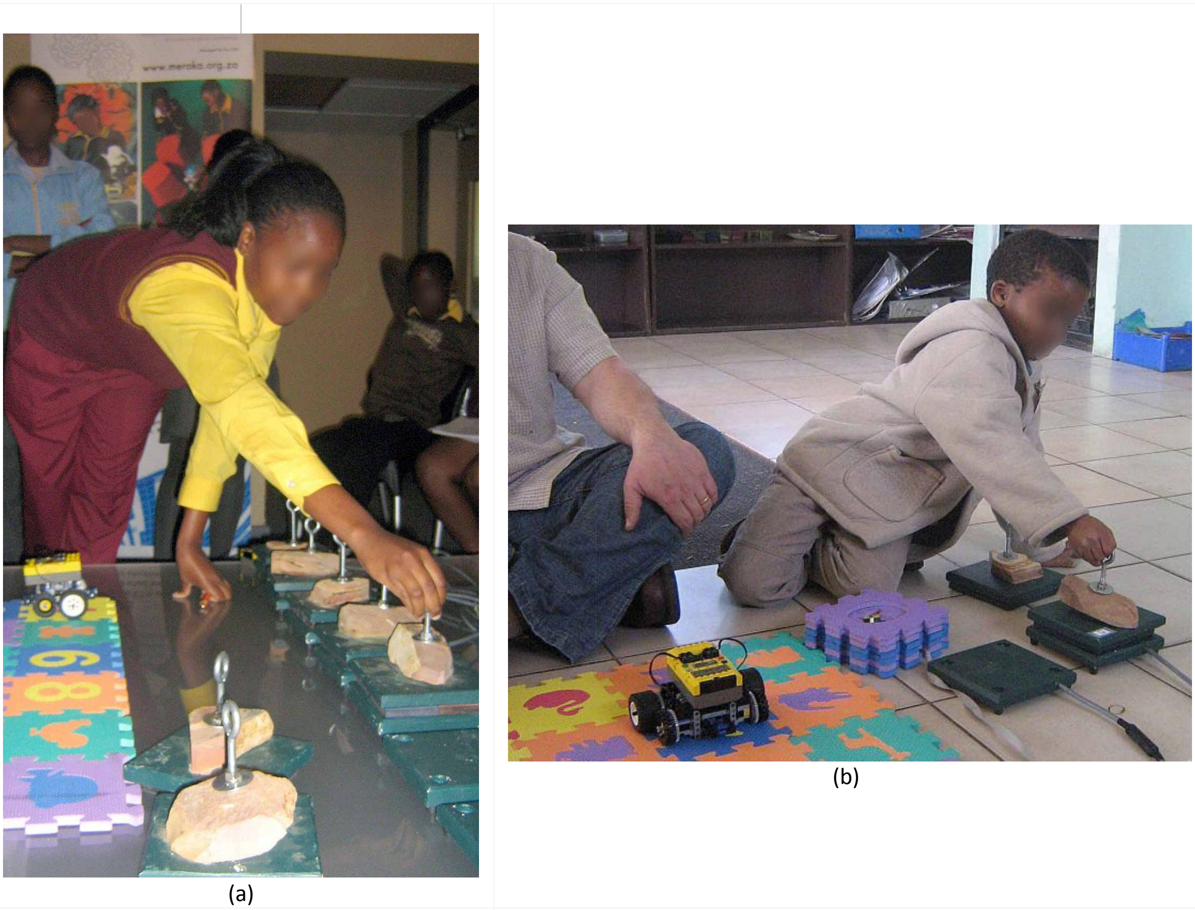
The third iteration was evaluated at two science events. The first evaluation was conducted at our Pretoria-based research institution with high school participants from the Giyani Science Centre, Limpopo Province. The learners were first introduced to the artefact and then invited to construct programs to achieve a predetermined objective (Figure 6-24a). The second evaluation took place in Grahamstown at the weeklong SciFest Africa (n.d.) event.

6.4.4.1 Evaluation design

At the onset of the workshop in Pretoria, participants received the evaluation form in Appendix I. Part 1 was filled out first while Part 2 was completed at the end of the workshop.

For the workshop in Grahamstown, the participants were requested to complete Part 1 of the second evaluation form in Appendix I. The participants were then introduced to the programming environment and given an opportunity to construct their own tangible programs to achieve a set objective. Finally, the participants completed Part 2. An average of 10 children contributed to each SciFest Africa workshop and the total number of participants for the week was approximately 100 children.

An evaluation of the RockBlocks program environment (including visual perception tests using these objects) was conducted with the assistance of pre-school children (Figure 6-24b) in the city of Pretoria and Putfontein town (Smith et al. 2008).



(a)

(b)

Figure 6-24 Constructing a RockBlock program

6.4.4.2 Evaluation results

Some participants found it challenging to align the signs to tiles with the result that the object's orientation or presence was not accurately sensed. One SciFest Africa participant's interpretation of the direction in which a particular arrow was pointing differed from my own. Visual perception discrepancies were also observed during the evaluations with pre-school children.

6.4.5 Discussion

The problem that some participants experienced in aligning the program object to the Sensing Tray could be addressed by incorporating a mechanical keying mechanism in such a way that the object easily aligns to the Sensing Tile (Smith et al. 2008). I have previously commented on my observation that individuals do not always perceive a sign in the same way. Observations at the Grahamstown workshops confirmed this problem. This supports my proposal to include the user in the design of the program object. The perception discrepancies led me to investigate further the meaning that individuals attach to signs and to let them choose or construct personally meaningful program elements.

By having shaped the arrows myself using simple materials and hand tools, I confirmed that it is plausible for users to construct their own program objects (Figure 6-22). This design iteration also served to confirm that a tangible programming language could be created with the aid of handcrafted signs. To explore the idea of user-constructed objects further, I constructed a clay representation of a car. Figure 6-25 (a) and Figure 6-25 (b) are photographs of the unshaped clay and the resultant car, respectively. Also shown in this figure is a conceptual programming environment that incorporates the clay car. Using this environment, the user constructs a program by placing copies of the car on recessed tiles.

The clay construction inspired future iterations by demonstrating another medium that the user can utilise when making her objects. My T-Logo system that is described later in this chapter is based on the results of this (RockBlocks) iteration and in particular the knowledge that a user can create her own objects. I will also I explicitly incorporate this knowledge in my model as presented in Chapter 7.

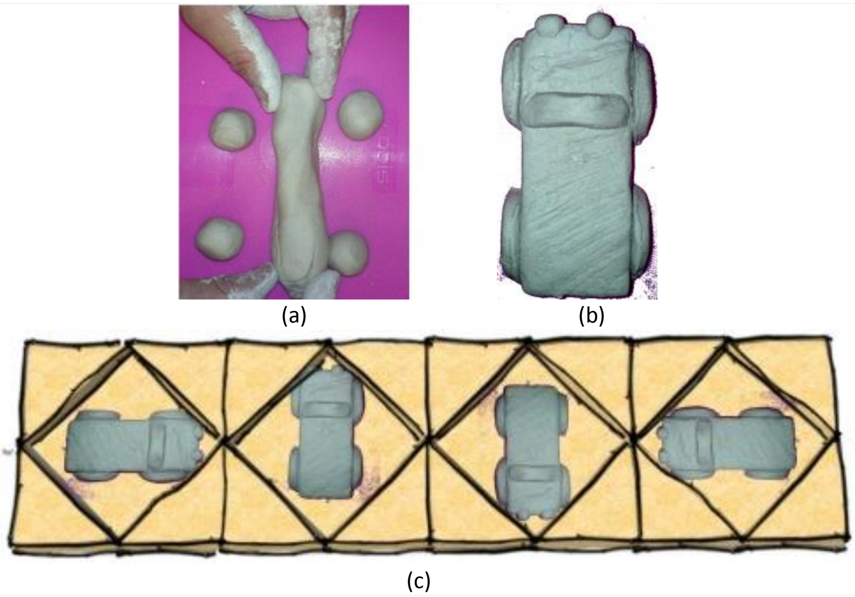


Figure 6-25 A tangible program object crafted from clay and the associated conceptual programming environment
(Smith 2008b)

6.5 Fourth iteration: Dialando

The first three iterations incorporated tangible objects into programs that affect the physical domain. The third iteration also confirmed that a user could construct a sign using natural materials. My objectives with the fourth iteration were to investigate what a tangible program environment could look like in which the program results where manifested in the digital domain and to determine if recycled materials can be used to construct program elements. To this end, I designed and implemented the *Dialando* programming environment in which the user applies five identical objects to code the movements of an on-screen character Figure 6-26 (c).

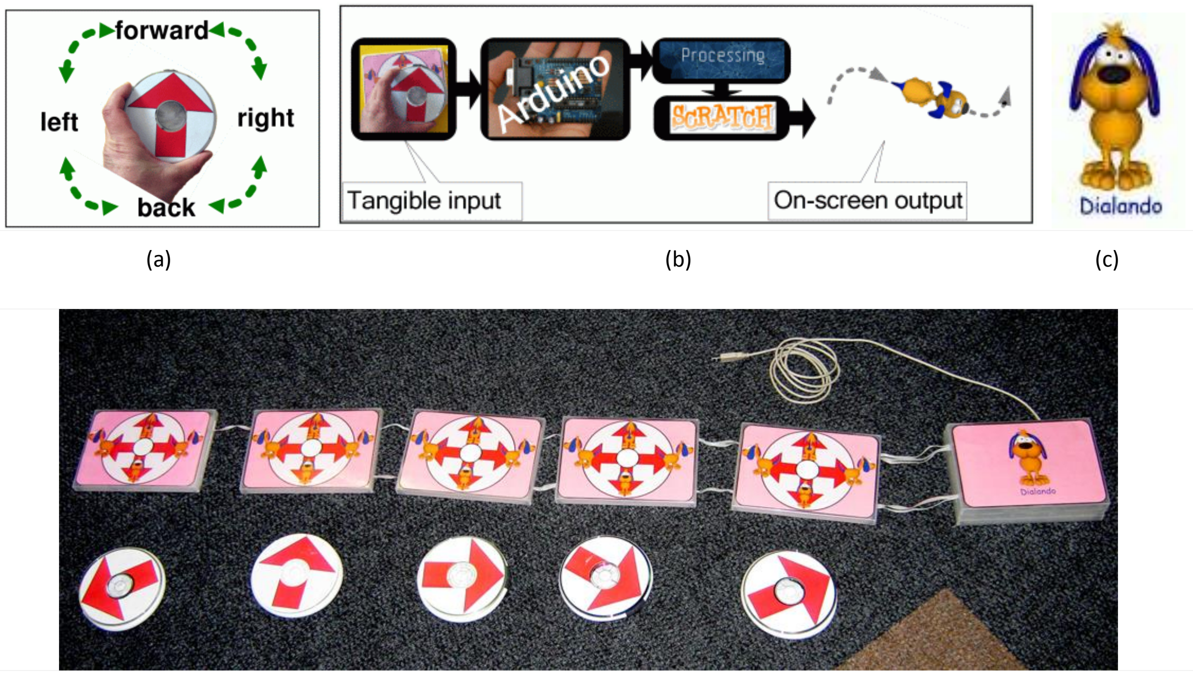
The name Dialando was derived from the words “dial and do” that also hint at how the programming environment is applied: The user constructs a program by “dialling” five actions and then the screen character “does” the actions.

The user interface is based on the orientation of five disks on top of five programming trays (Figure 6-26d). The pattern sequence of these disks represents the sequence of actions that the character will execute. An object on a tray represents one program instruction.

6.5.1 System design

The Dialando language consists of four signs and these are *Forward*, *Back*, *Right*, and *Left*. A sign is encoded by the direction in which the object points and the language syntax is comprised of the four directions in which an object can point (Figure 6-26a). I call these directions *North*, *South*, *East*, *and West,* respectively with North being the direction pointing away from the user. An object pointing North encodes a forward motion and this sign is called Forward. Conversely, a South pointing object represents a backward motion and the associated sign is called Backward. For an East-pointing object the resultant sign represents a 90 degree turn to the right and I call this sign Right. A similar convention holds when the object points to the West and I call this sign Left. The interpretation of the Forward and Backward signs results in the character covering a fixed linear distance on the computer screen. The Right and Left signs cause the character to rotate through 90 degrees.

Each programming surface can detect a sign and transmit this information to a program executing on an Arduino (Banzi 2009). The Arduino circuit in turn relays this information to a program executing on a computer and written using the Processing (Reas & Fry 2010) language. A second program, executing on the same computer and written in the Scratch (Resnick, Maloney, Monroy-Hernández, Rusk, Eastmond, Brennan, Millner, Rosenbaum, Silver, Silverman & Kafai 2009) language, generates the on-screen character avatar in response to instructions received from the Processing program. Figure 6-26b illustrates the flow of information from the tile, to the Arduino, to the Processing and Scratch programs, and finally the effect it has on the displayed image.



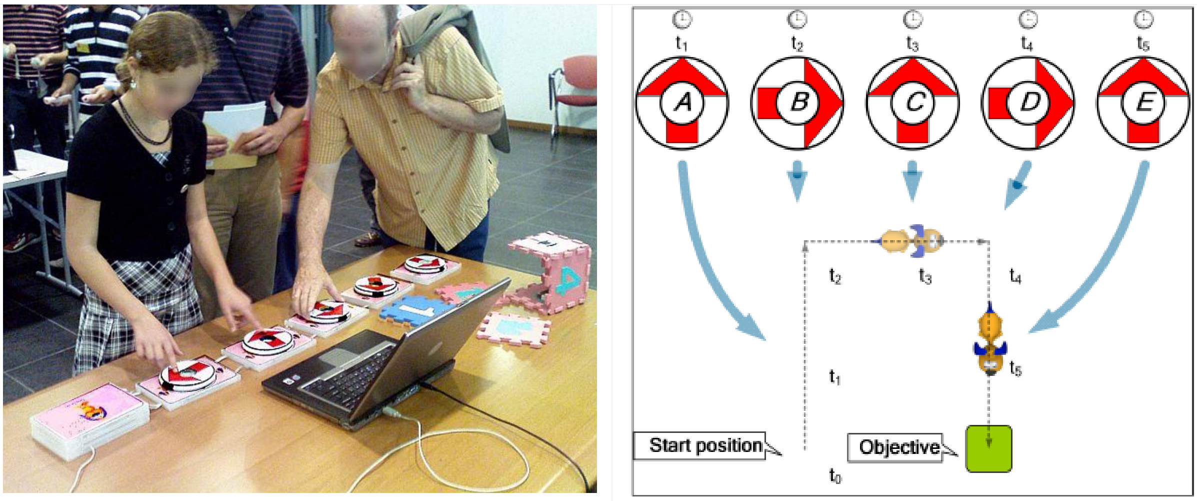
(a) (b) (c)

(d)

Figure 6-26 System components

(Smith 2010a)

The user programs a series of actions by placing (Figure 6-27a) signs onto the programming surfaces. Figure 6-27b illustrates an example of a program constructed with the aid of the Dialando programming environment. This figure captures the correlation that exists between the five program objects and the five on-screen actions. In this figure, objects A through E are interpreted at times t_1 through t_5 while the character executes movements that correspond to the sign being interpreted.



(a) (b)

Figure 6-27 The fourth iteration artefact in use and a program example in execution

(Smith 2010a)

6.5.2 Tangibles design

Each tangible object holds two magnets that are sandwiched between recycled compact disks (CDs). A short length of salvaged electric cable was inserted between the disks on the circumference to add mechanical strength to this assembly and to improve the object's aesthetic properties. A large printed arrow finishes the design.

Evaluation of the first three iterations identified that the user had difficulty in aligning the tangible object to the sensing surface. I addressed this by adding magnets in the centre of both the programming trays and the program objects to help align the two. Using this mechanism, the user rotates the object by hand to select a sign while the magnets keep the object and the tray properly aligned.

6.5.3 Discussion

Even though I did not evaluate this iteration with the help of children I observed adults and children when they interacted with the artefact at a conference presentation (Figure 6-27a). The new knowledge I gained is therefore the result of my experience in designing and implementing the artefact and my observations of persons using the artefact. I concluded that it is possible to control aspects of the digital domain by combining a tangible program, Arduino-based hardware, the Processing programming language, and the Scratch programming language. Figure 6-28 reflects my conclusion within the block labelled New Knowledge.

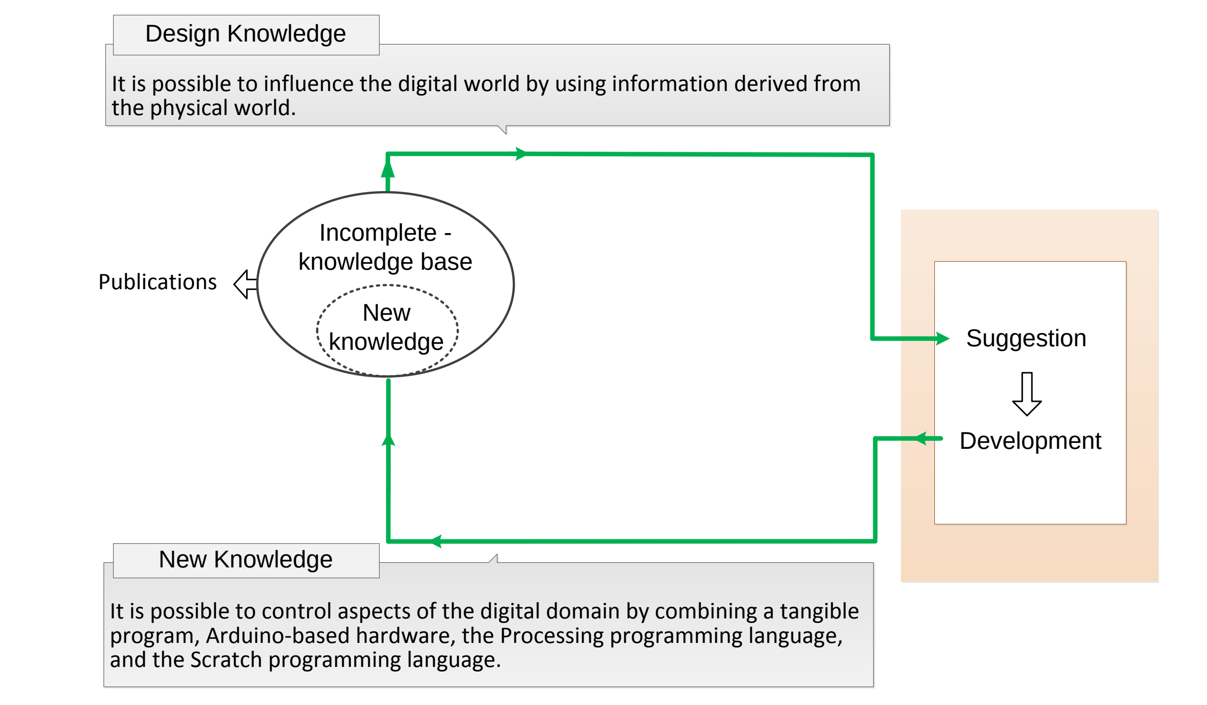

Design Knowledge
It is possible to influence the digital world by using information derived from the physical world.
Incomplete - knowledge base
New knowledge
Publications
Suggestion
Development
New Knowledge
It is possible to control aspects of the digital domain by combining a tangible program, Arduino-based hardware, the Processing programming language, and the Scratch programming language.

Figure 6-28 The knowledge that informed the fourth iteration and the results

As far as the objects are concerned, I confirmed that previously discarded everyday objects could be repurposed as tangible program elements. Finally, the simplicity of transporting this system across continents (as I had to do on one occasion) and setting it up at a conference for demonstration was an improvement over the same operation applied to the first iteration. This improvement can be ascribed to two factors. First, the results of tangible program execution can be observed without the addition of a physical toy that has to be transported. Second, the interface between the tangible program and the output device does not include custom-made electronic circuitry that can be damaged during transportation. These two observations prompted me to implement only digital outputs in my fifth iteration.

6.6 Design iteration five: T-Logo

This iteration addresses Research Objectives 2, 3, and 4 as stated in Chapter 1 and I repeat these here:

Research Objective **2**	To devise a mechanism by which a personally meaningful tangible object can be used as a program element.
Research Objective **3**	To devise a method by which the positions of one or more personally meaningful tangible objects can define a program statement.
Research Objective **4**	To devise a programming environment in which the relative positions of personally meaningful tangible objects are interpreted as a program.

Iterations one through four produced artefacts in which programs are composed of objects arranged in a linear fashion. In contrast to linear arrangements, the fifth iteration explores how object groupings define a program. The Gestalt principles of perceptual grouping predicts that humans associate objects with each other when they are together and it is this prediction that forms the foundation of my final iteration.

6.6.1 Design process

This iteration still incorporates the Gestalt principle of good continuity as was applied to the previous iterations, but now also adds the principle of grouping by proximity (it does not replace the previous principle). Figure 6-45 illustrates both principles applied in a tangible program.

I use the word *clustering* to describe the action when the user places objects in close proximity to each other and at the same time removes objects that are to be excluded from those assembled this way. Clustering therefore considers the spatial relationship that exists between objects. Based on my experience with the fourth iteration I concluded that it is possible to control events in the digital domain by combining a tangible program, Arduino (Blum 2013)-based hardware, the Processing (Reas & Fry 2007) language, and the Scratch (Resnick et al. 2009) language. As indicated in Figure 6-29, I applied this knowledge in the design of the fifth and final artefact.

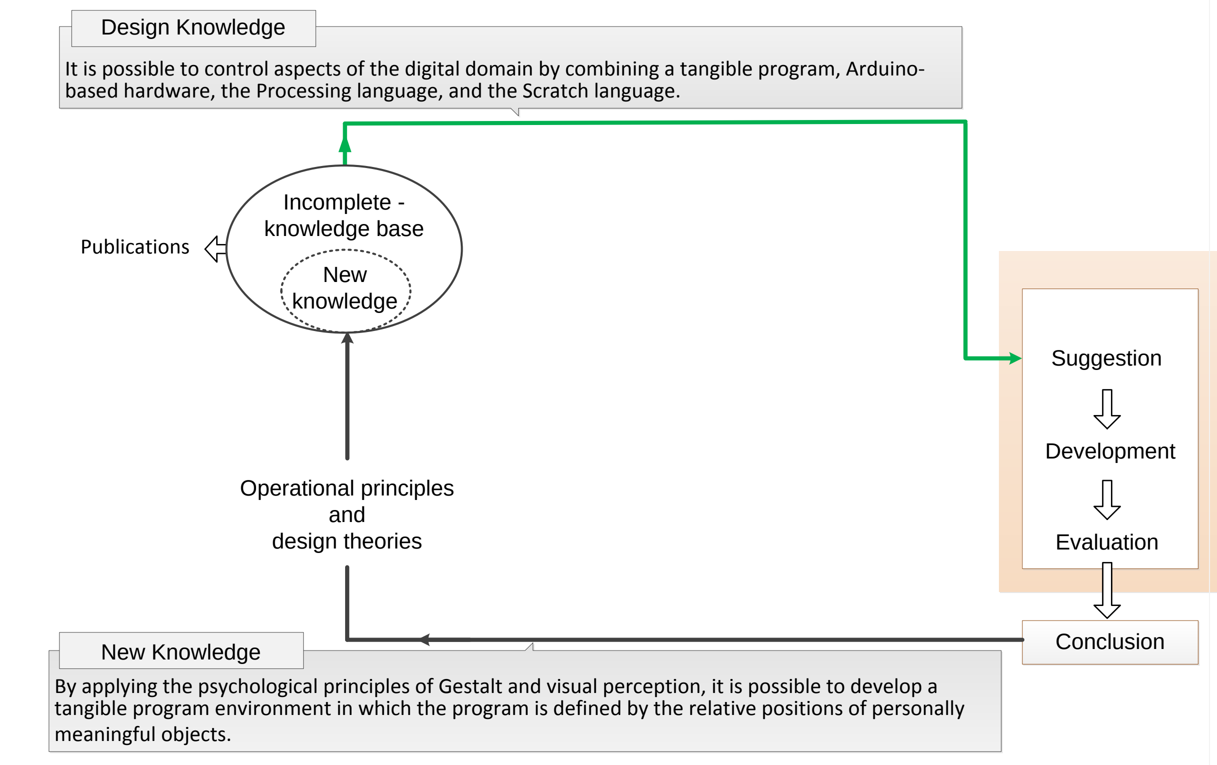

Design Knowledge
It is possible to control aspects of the digital domain by combining a tangible program, Arduino-based hardware, the Processing language, and the Scratch language.
Publications
Incomplete - knowledge base
New knowledge
Operational principles and design theories
Suggestion
Development
Evaluation
Conclusion
New Knowledge
By applying the psychological principles of Gestalt and visual perception, it is possible to develop a tangible program environment in which the program is defined by the relative positions of personally meaningful objects.

Figure 6-29 The knowledge that informed the fifth iteration and the results

6.6.2 Evaluation methodology

I compared the artefact of this iteration with TERN, an often cited tangible programming environment. I did this in the laboratory by identifying both language structures that are the same for both environments, and structures that were not. Program segments were then designed using the two languages.

Three criteria were used to analyse the evaluation data. The first criterion was formulated to determine how much effort the user has to exert when creating a program element from a personally meaningful object. I addressed this question by constructing objects using the cardboard packaging from household products and adding optical markers to their bases. In addition to the cardboard objects, I also constructed objects using wooden cubes and dowels used in children's construction sets. Paper notes with personally meaningful signs were attached to the dowels. Using the Mapping mode of the software I wrote, I then assigned either an action or a numerical value to the marker's identification number.

The second criterion was to determine the maximum usable object size for a given construction surface while still accommodating a useful number of objects. This knowledge is important since, if the marker is too small, the visioning system cannot identify the marker and if the objects are too large then the number of objects that can fit onto the input surface is reduced. The balance was found by using a fixed construction surface of a certain size and determining the smallest recognisable marker through experimentation.

Finally, I wanted to determine if my artefact could successfully implement program structures found in the TERN programming environment. I did this by selecting instruction set elements from the TERN environment. At the same time, I noticed that certain constructs mapped directly to my own language while other constructs did not.

6.6.3 Tangible program concepts

In this section, I discuss two concepts that are central to this iteration. The first considers the relationship that exists between user-created objects and a computer program. I then consider my approach to derive a digital representation of a tangible program. This section concludes with two implementations of this approach.

6.6.3.1 Physical and digital constructions

A text-based program depends on the existence of a programming language and a user to construct the program. Constructing a textual program therefore involves the following four key steps (Figure 6-30): First, a programming language architect designs the language and a developer implements it. The developer then publishes the associated signs and rules for the benefit of the user. A user can now construct a program by arranging the signs according to the rules. Finally, an interpreter produces the result.

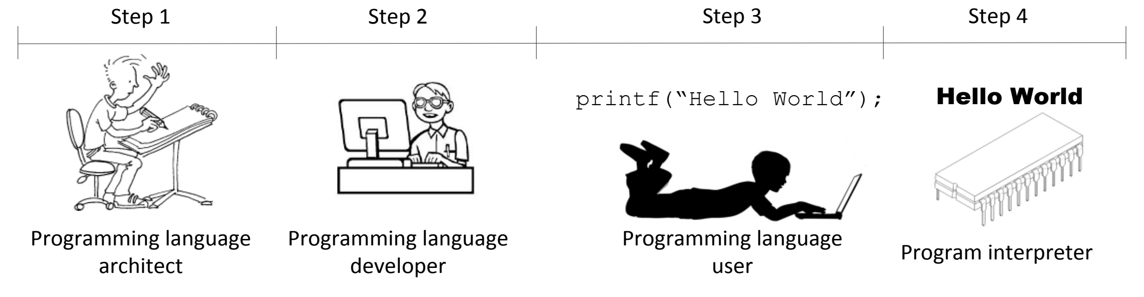

Step 1
Step 2
Step 3
Step 4
printf("Hello World");
Hello World
Programming language architect
Programming language developer
Programming language user
Program interpreter

Figure 6-30 The four key steps involved in constructing a text-based computer program

Computers interpret programs that are in digital form and cannot directly execute a program constructed using physical objects. Therefore, in order for a computer to interpret a tangible program and execute the result it is a requirement that a digital equivalent of the physical construction be determined first. Put succinctly, a set of digital instructions that represent the tangible program is necessary.

Figure 6-31 illustrates the construction of objects for use in a tangible program. The construction of tangible programs involves four steps: First (Step 1), the user creates tangible objects using raw materials. This step is optional since pre-existing artefacts may also be used. The tangible objects are then paired (Step 2) with written program instructions. In Step 3, the tangible objects are arranged into a particular configuration to create the desired program. The final step (Step 4) is to interpret the program and produce the output.

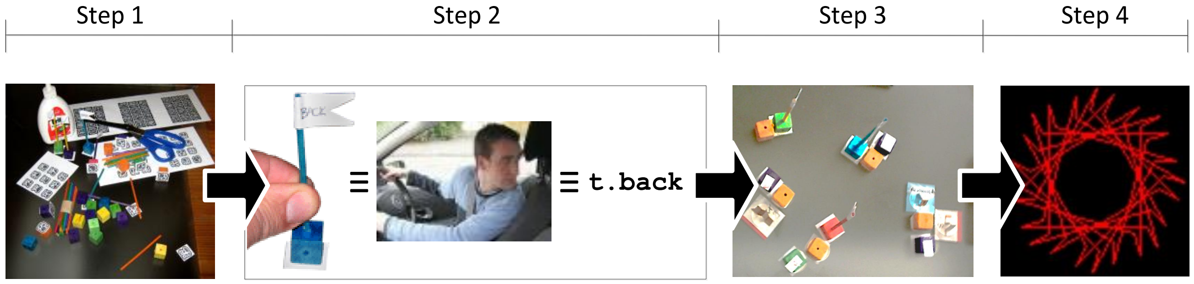


Figure 6-31 Four steps in constructing a tangible program

For the interpreter to function correctly it needs to know the position of each object on the construction surface. The program determines this position relative to a two-dimensional orthogonal axis in the plane of the construction surface. Figure 6-32 illustrates this concept.

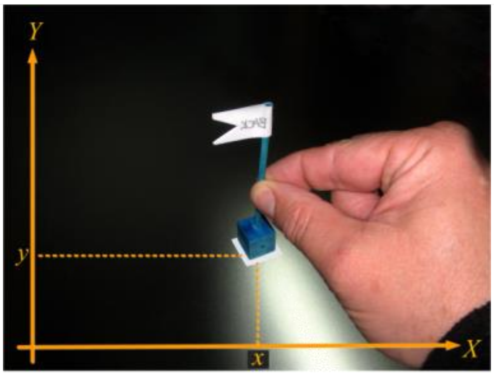


Figure 6-32 The general case of defining the object's position on the two dimensional construction surface

6.6.3.2 Mapping between objects, actions, and parameters

Programming language architects often design sign sets with which the user can represent program actions and parameters. An example taken from the C (Kernighan et al. 1988) language lexicon is the sign `printf`. A closely related sign in the JavaScript (Wootton 2001) language is `document.write`. A user can construct programs to render text using these signs.

A coder implements the language design as software routines and permanently links them to signs of his choice. The coder also creates a permanent link between the routines and the signs that represent them. My approach is different: Instead of letting the coder decide which signs to use, I let the user choose personally meaningful objects. I use the term *mapping* to describe the process of associating objects with actions and parameters.

6.6.3.3 Clustering, Cluster Marker, and Cluster Marker Zone

The concepts of *Clustering*, *Cluster Marker*, and *Cluster Marker Zone* are useful when one wishes to describe an environment in which object clusters are interpreted as tangible programs. I distinguish between physical and digital clustering.

Physical clustering is the gesture by which tangible objects are grouped together (Figure 6-33). I further use the word *digital clusterin*g to describe a software method that associates the discrete elements of grouped objects with each other. Optical markers attached to objects make digital clustering possible when all markers are embedded with identification (ID) numbers.

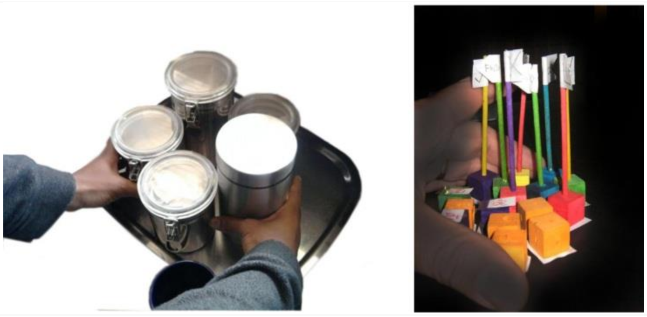


Figure 6-33 Physical clustering

A *Cluster Marker* (*CM*) is a predetermined tangible object that locates a point on a flat surface such as the glass construction surface used in this iteration. In this iteration, I have allocated an ID number of zero to any object that serves as a Cluster Marker. As is the case for all objects, the Cluster Marker is identified by using the combination of its position on the construction surface and ID. I use the notation $CM_{X,Y}$ to uniquely describe a Cluster Marker where X and Y are the horizontal and vertical positions, respectively.

I call the concentric area around the Cluster Marker with radius R, the *Cluster Marker Zone* (CMZ). The Cluster Marker Zone describes the Cluster Marker's 'area of influence' (the shaded area in Figure 6-34) and all objects that lie within the Cluster Marker Zone are associated with that particular Cluster Marker.

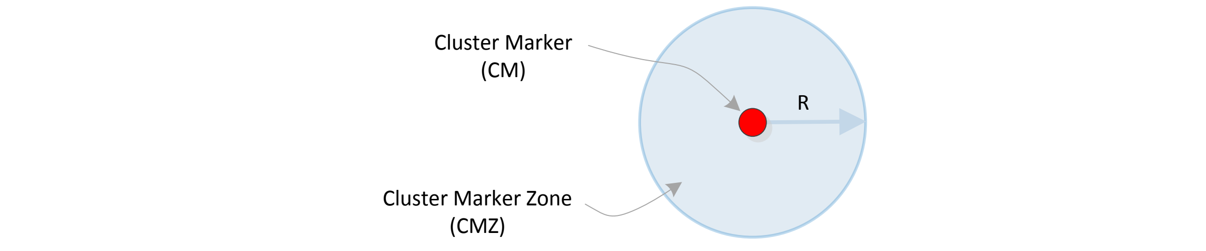


Figure 6-34 The relationship between the Cluster Marker, its radius, and the Cluster Marker Zone

6.6.4 Clustering topologies

I next consider two clustering topologies called *Clustering Topology #1* and *Clustering Topology #2*, respectively. I first developed Topology #1 and took it through two iterations but it was eventually abandoned due to an ambiguity problem and computational complexity. The second topology was eventually adopted and used to complete this research project. Section 6.6.4.1 and Section 6.6.4.2 discuss these topologies, respectively.

6.6.4.1 Topology #1: Grouping by common region

The first topology was inspired by the hand gestures in Figure 6-33 and is based on the Gestalt principle of grouping by common region. Its design considers the application of hoops to define common regions.

The topology assumes hoops with known radii are used and each has two identical markers at opposite ends. Physical clustering is achieved when objects are placed inside the hoop outline. The challenge is in developing an interpreter algorithm to correctly match markers to their corresponding hoops. Shown in Figure 6-35 (a) are examples of narrow and wide hoops. To get an idea of the theoretical limit of this topology I first discuss the theoretical case of infinitely narrow hoops. I then apply the algorithm to wide hoops.

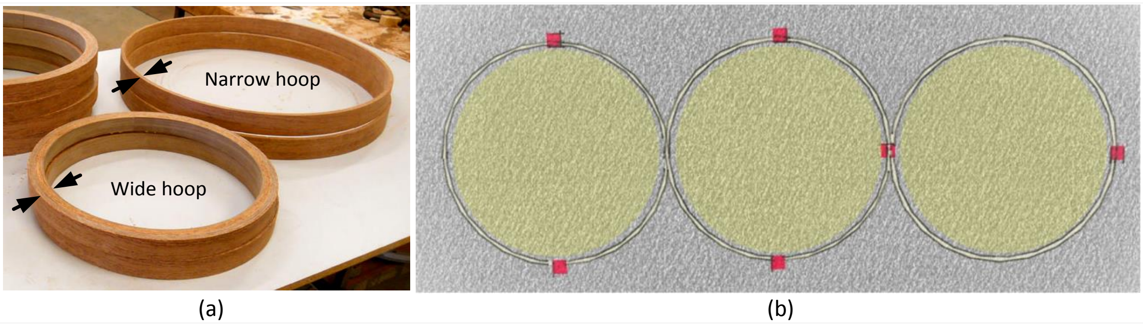


Figure 6-35 Hoops and markers

Figure 6-35 (b) is a schematic drawing of three hoops viewed from above where the red dots indicate markers along the circumference. Marker pairs delineate well-defined and mutually exclusive areas (highlighted) in this figure. The size of an area is directly proportional to the hoop radius and therefore also the distance between the two markers. My algorithm uses the radius with marker positions to derive where the areas are located and then digitally clusters together all the objects that lie within this zone.

The algorithm has two parts: The first step in identifying the clusters is to detect the markers and noting their positions. A unique uppercase letter is then assigned to each marker. The second step matches each marker with its companion and this is achieved as follows: For each marker, a virtual circle with radius R is constructed with its middle aligned with the marker. I call this marker the *primary marker*. This circle serves as the trajectory along which matching markers are searched. Markers that lie along this trajectory are called *candidate matching markers* and are assigned a lower case letter that corresponds to the letter assigned to the primary marker.

It is not possible to match unambiguously the markers when the hoops have no width. I illustrate the problem using Figure 6-36. In this figure, C, D, E, and F are primary markers that each has two candidate matching markers. These markers cannot be matched unambiguously without additional information. In an effort to overcome the ambiguity, I considered a more realistic scenario in which the hoops have significant width. Figure 6-37 illustrates the same configuration but now with wide hoops.

Again, I first determine whether a marker matching ambiguity could exist. Inspection of Figure 6-38 reveals that wider hoops solve the ambiguity that exists in the case of infinitely narrow hoops. This is because the primary markers unambiguously match their candidate markers. However, a second ambiguity is evident when the wide hoop configuration is changed: Figure 6-39 illustrates the ambiguity when the hoop on the right is moved away from the other hoops. In this figure, marker B is a matching candidate for the markers labelled A, C, and D. The general solution when using wide hoops is to first match all primary markers that have a single matching candidate marker and remove these pairs from the pool of matching candidate markers. The primary markers are labelled A, E, and F in this example. These two steps are repeated until all the markers have been paired. Having considered Clustering Topology #1’s mechanical requirements and finding these to be acceptable but complicated, I considered an alternative topology that I call Clustering Topology #2.

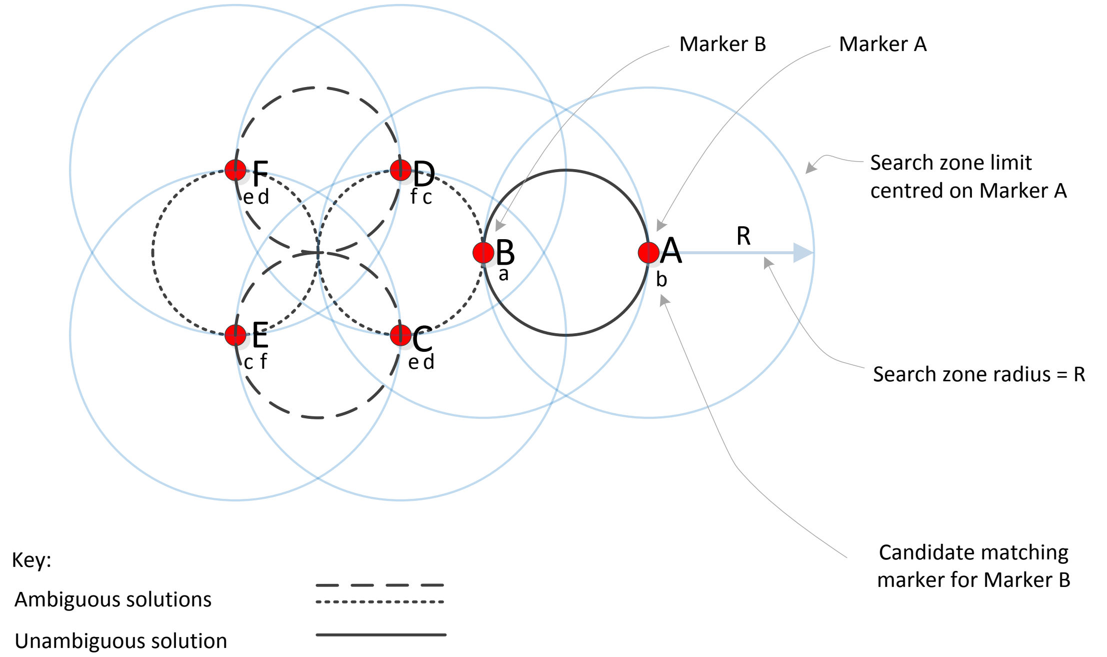


Figure 6-36 Algorithm ambiguity in the case of infinitely narrow rings

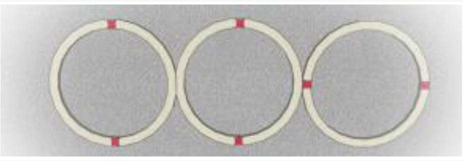


Figure 6-37 Three wide hoops with markers

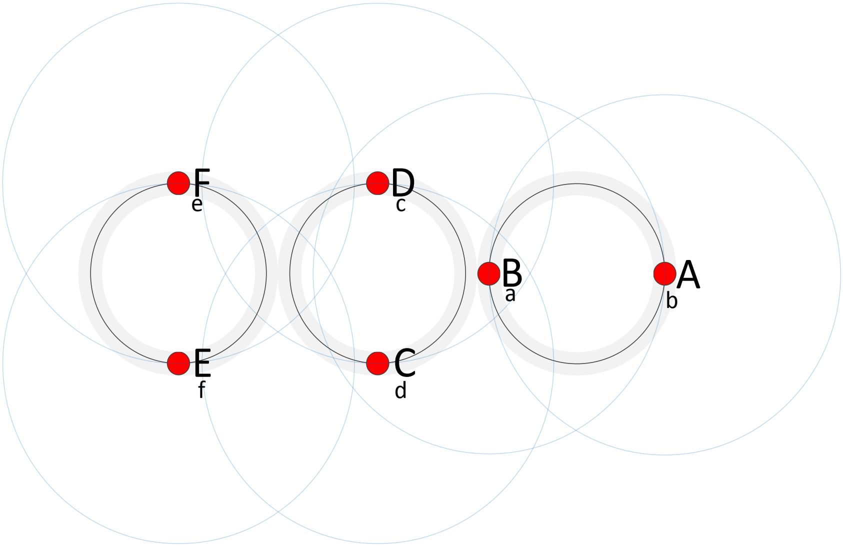


Figure 6-38 Wide hoops eliminate the initial ambiguity

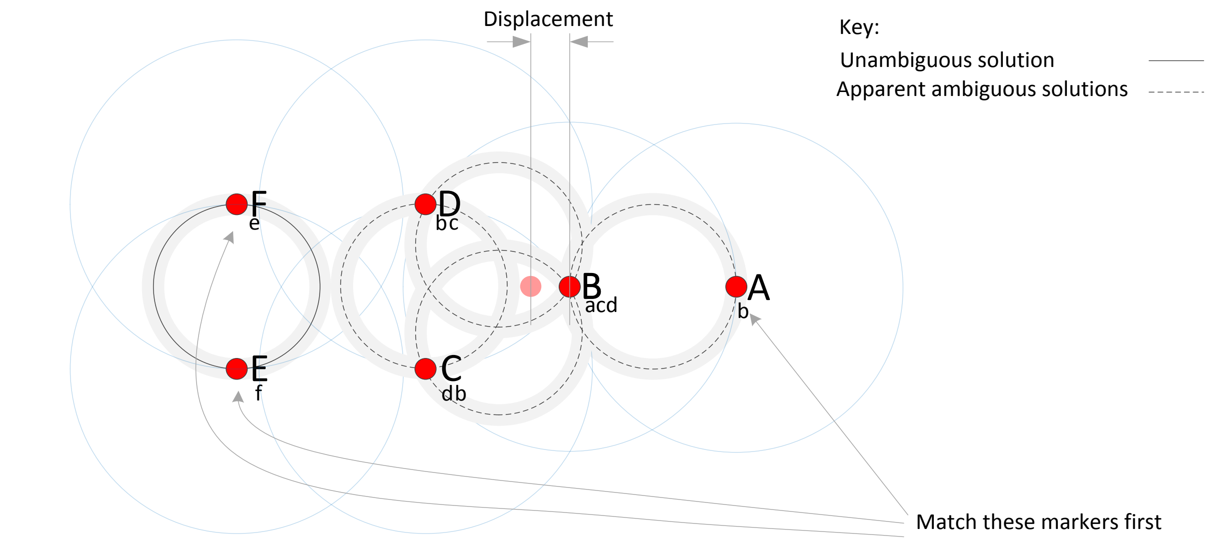


Figure 6-39 Apparent ambiguity in the case of three wide hoops

6.6.4.2 Topology #2: Grouping by proximity

The ambiguity problems of Topology #1 are due to identical markers on the rings. Topology #2 does not have this problem and it has the added benefit of simplified mathematical operations. The second clustering topology is based on objects assembled close to a Cluster Marker. This topology therefore relies on the Gestalt principle of grouping by proximity. My clustering algorithm is as follows: First, all Cluster Markers are identified and their Cluster Marker Zones calculated. The positions of all remaining markers are then compared with those of each Cluster Marker. Finally, markers are associated with a particular Cluster Marker if the marker is located within that particular Cluster Marker Zone. In the case where a marker lies within multiple Zones the marker is allocated to the first Cluster Marker candidate identified. My final iteration is based on this topology.

6.6.5 Language and system usage

The programming language consists of six actions with which on-screen drawings may be produced and these are *Paint*, *No Paint*, *Forward*, *Backward*, *Left*, and *Right*. Distances are specified in screen pixel units while angles are in degrees and whole numbers.

I next describe how to use the programming environment: The user is initially given a list of actions (Paint, No Paint, Forward, Backward, Left, and Right) that can be used to construct a program. She then creates Tangible Program Objects by combining wooden blocks, dowels, and paper markers. Once these objects have been mapped to actions and values, the Tangible Program Objects are viewed as Tangible Program Elements. Third, the user now arranges the Elements on the Construction Surface according to simple rules as described in Section 6.6.6. Finally, the user instructs the computer to execute the tangible program at which time a series of software applications interpret the Elements, create a text file containing computer executable code, and execute this code.

6.6.6 Programming rules

Ten programming rules apply when a program is constructed. First, an Element is ignored if it is not detected within a Cluster Marker Zone. Two, it is valid to have multiple command Elements in a cluster. Three, the numerical sum of all parameter Elements within a Cluster Marker Zone is calculated and used. Four, parameters are assumed to have a value of zero when no parameter object is detected within a Zone. Five, commands in a cluster share the parameter Elements detected in the cluster. Six, an object can be associated with multiple Zones when these overlap. Seven, the cluster is ignored if part of a command/parameter pair is missing. Eight, the program is executed repeatedly. Nine, the orientation and position of an Element within the Cluster Marker Zone are inconsequential. Finally, the interpretation sequence of the construction surface is from the top-left to the bottom-right.

6.6.7 System components

This design implementation consists of seven component types. The first is a tangible object with an attached marker (marked as ‘A’ in Figure 6-40). The rest are a flat translucent glass surface that I call the C*onstruction Surface* (B), a camera (C), a light source (D) to illuminate the surface from below, VCam (e2esoft 2012) camera control software, the reacTIVision (Kaltenbrunner & Bencina 2007) image analysis software, and my T-Logo application software.

6.6.8 Information transfer from the physical to digital domain

The following describes how tangible object (‘A‘ in Figure 6-40) information is copied to a digital representation of the program. First, the user places tangible objects with their markers onto the Construction Surface (‘B’ in Figure 6-40). Next, the camera captures images of these markers and the camera control software relays the images to the image analysis software. The image analysis software in turn determines the two-dimensional position and rotation angle of every marker. Finally, the position and orientation of all markers are sent to the T-Logo software for further processing.

6.6.9 Software modes

The T-Logo software operates in four modes and these are the *Map*, *Construct*, *Interpret*, and the *Execute* modes (Figure 6-41). This is also the sequence in which a user will initially interact with the T-Logo software. The user is the only person who interacts with the system depicted in this figure and may change the current mode by selecting an alternative one using the computer keyboard.

Construct is the default mode and from here the user can press ‘a’ to activate the Map mode, ‘i’ for Interpret, or ‘e’ for the Execute mode.

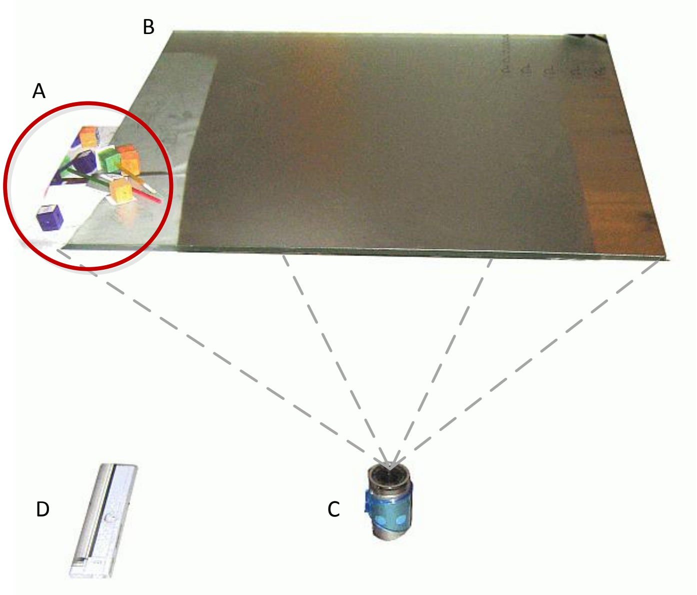


Figure 6-40 The T-Logo physical components

6.6.9.1 Map mode

The Mapping process combines a number of user actions and these entail selecting the Mapping mode, responding to on-screen prompts, providing computer input by means of the keyboard and mouse, and physically placing objects onto the Construction Surface.

The Map mode is the initial mode a user interacts with extensively and subsequent interactions may bypass it. When in this mode, a tangible object may be mapped to a predefined action or parameter after which the object can be used in a program. I refer to an object that has been mapped as a Tangible Program Object.

Tangible Program Objects are identified by their marker and a table is maintained in computer memory that maps markers to either an action or parameter. Mapping provides for multiple identities to be optionally mapped to a common action or parameter. However, a particular identity may be mapped to only one action or a single parameter. Figure 6-41 illustrates the states of the Map mode. States are shown using the sign S_n. The following paragraphs describe how a user applies the Map mode and Figure 6-42 illustrates the user actions. Transitions between states or modes are indicated using the sign T_n. These transitions correspond to those shown in Figure 6-41 and Figure 6-42.

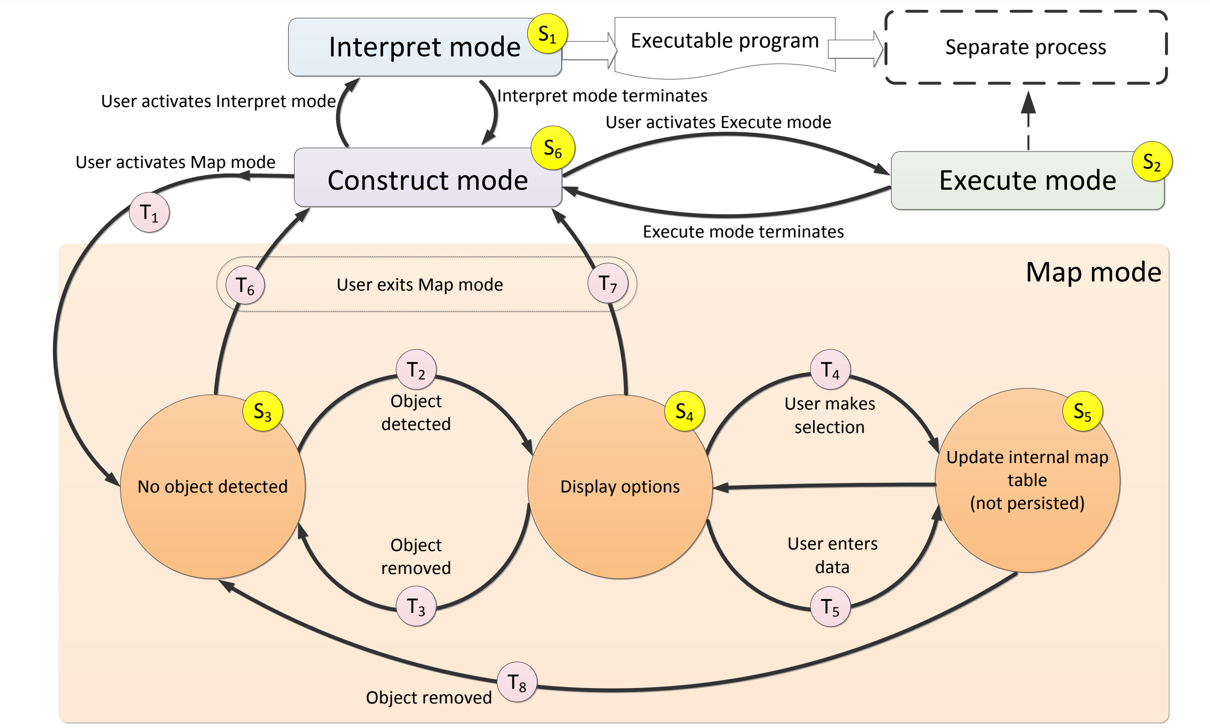
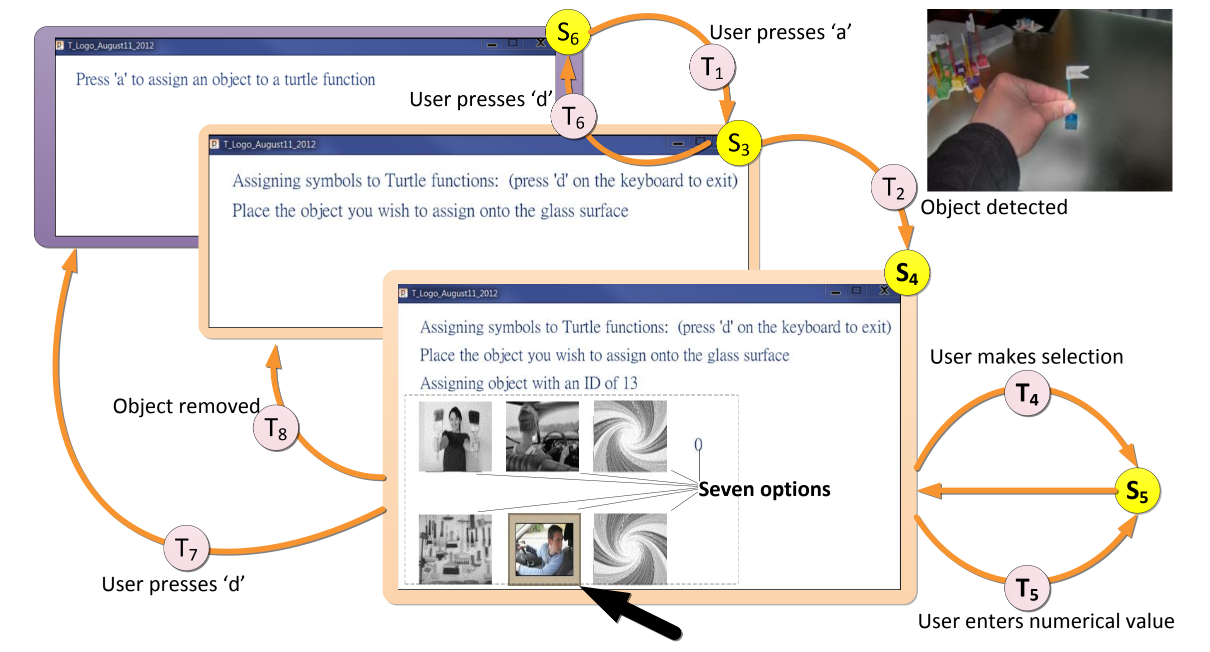


Figure 6-41 The four T-Logo software modes



T_Logo_August11_2012

Press 'a' to assign an object to a turtle function

T_Logo_August11_2012

Assigning symbols to Turtle functions: (press 'd' on the keyboard to exit)
Place the object you wish to assign onto the glass surface

T_Logo_August11_2012

Assigning symbols to Turtle functions: (press 'd' on the keyboard to exit)
Place the object you wish to assign onto the glass surface
Assigning object with an ID of 13

Figure 6-42 An example of a T-Logo mapping activity

To construct a program using T-Logo, the user activates the Map mode and maps objects to actions and parameters. First, the user sources or creates a tangible object that is a personally meaningful representation of the action or parameter. He then attaches a marker that contains a unique identity number to the bottom of this object, thereby creating a tangible object/marker pair. The identity number 10 is reserved for Cluster Markers. Third, the user selects the Map mode by pressing 'a' (representing "assign") on the keyboard (transition T_1). The software is now in state S_3 and the user can either exit the Map mode by pressing the 'd' key (for "done") or continue with the mapping process by placing a tangible object with its marker onto the construction surface. The T-Logo system will attempt to identify any markers on the surface and present to the user seven options on the computer screen (transition T_2 and state S_4). The options represent six actions and one numerical parameter. The user can navigate the options and makes a selection (T_4) using the mouse. Alternatively, he enters a numerical value using the keyboard (T_5). Finally, he confirms the selection by pressing 'd' on the keyboard (T_7) at which time the T-Logo software will update the mapping table according to the most recent selection and transit to state S_6.

The on-screen appearance of the seven options vary according to which of three selection states is active at the time. Selection states are *Dormant*, *Has focus*, and *Active* (Figure 6-43). Six of these options are *Paint*, *No Paint*, *Forward*, *Backward*, *Left*, and *Right*. The Paint and No Paint actions are equivalent to the Logo language's (Papert 1980) Pen Down and Pen Up instructions while the Forward, Backward, Left, Right actions are equivalent to the Logo language's Forward, Backward, Left, and Right instructions. The seventh option ('Value') is used when the keyboard is used to associate a numerical value with the tangible object/marker pair.

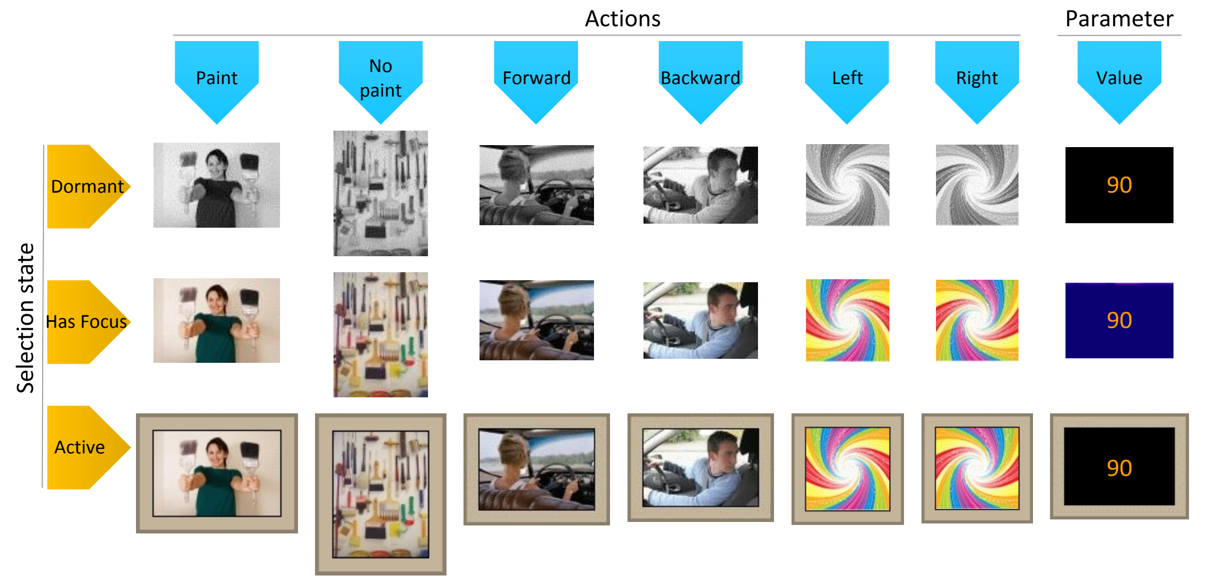


Figure 6-43 The on-screen appearance of the seven user-selectable mapping options

I implemented the command choices using selectable images and as follows: The Paint action is represented by a woman holding two paint brushes and No Paint is depicted as a wall panel with all the brushes hanging in their allocated places. The Forward and Backward actions are represented by images of persons driving a car. Finally, Left and Right turn actions are represented by abstract images that create the impression of left and right rotation, respectively.

The following example illustrates the mapping activity. Figure 6-42 shows the various prompts to which the user has to respond. The default prompt instructs the user to press ‘a’ on the keyboard:

```
Press ‘a’ to assign an object to a turtle function.
```

When the user responds by entering ‘a’ on the keyboard, the program changes state (transition T_1) and the user is prompted as follows:

```
Assigning signs to Turtle functions:(press ‘d’ on the keyboard to exit).
      Place the object you wish to assign onto the glass surface.
```

The user then places an object (with ID=13 in this example) on the construction surface and the software changes state (transition T_2) and confirms that an object is being assigned to a command or parameter, as appropriate:

```
Assigning object with an ID of 13.
```

In this example, the user selects (T_4) the `Backward` graphic using the computer mouse. In response, the software changes to state S_4 and highlights the selected option. The user may now choose a different option. Once the user is satisfied with the selection she presses the ‘d’ key and exits (T_7) the Map mode.

6.6.9.2 Program Construct mode

The user has no access to any Tangible Program Objects when the environment is used for the first time. The reason being that the Map mode has to be activated first and the mapping activity completed before the Construct mode can be used. Once this has been accomplished, the user needs only to interact with the Construct, Interpret, and Execute modes. Once the user has created a number of tangible programming objects, he is now in a position to construct a tangible program and he does this by placing the objects onto the construction surface.

6.6.9.3 Program Interpret and Execute modes

The tangible program is interpreted when the user selects the Interpret mode by pressing the ‘i’ key while in the Construct mode. In brief, the T-Logo interpreter then locates all the Tangible Programing Objects, groups them by proximity, and creates a text file containing Processing (Greenberg 2007) language instructions. I now explain the interpretation process in more detail: First, clusters are sorted according to a top-bottom, left-right sequence. To illustrate, consider the Cluster Markers in Figure 6-44 where the sorting process produces the following Cluster Marker sequence: CM_1 ⇨ CM_2 ⇨ CM_3. Two, a program pre-amble (written in the Processing language) is then sent to a target file. Each sorted cluster is then evaluated according to their sorted sequence. When multiple numerical values are present in one cluster, the values are totalled and the result used in further operations. A Processing line of code is sent to the target file for each Action detected in a cluster. If the T-Logo language prescribes that a numerical value must accompany the Action, then all the parameter objects within the current cluster are totalled and sent to the target file. Parameter objects are ignored where none is required. A second fixed set of predefined Processing language instructions is then sent to the target file.

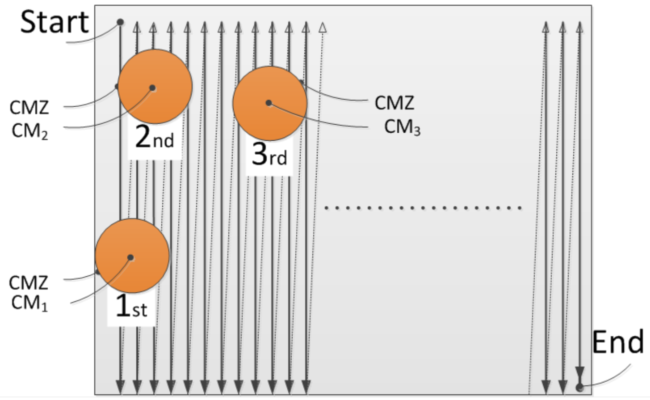


Figure 6-44 The sequence in which objects on the construction surface are processed

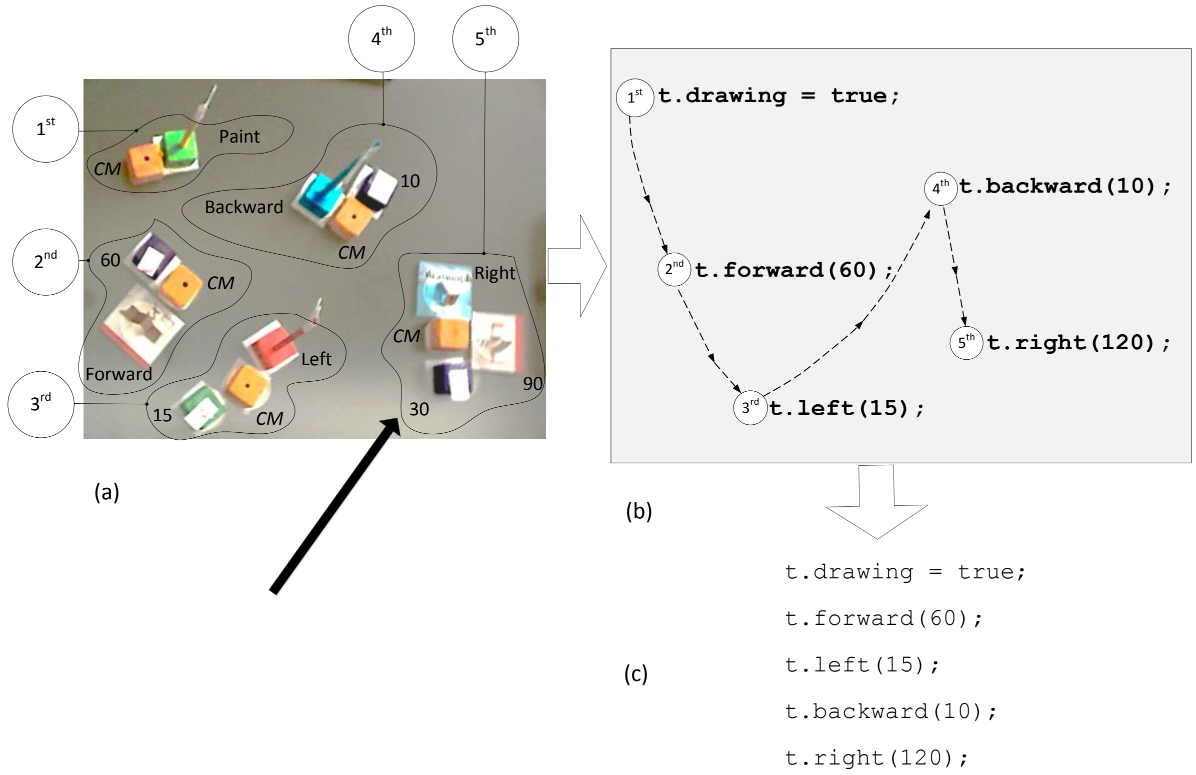
The system is reverted back to the Construct mode and the user can now press the 'e' key to select the Execute mode. A separate process is then activated and a new instance of the Processing environment is launched with the target file as command line parameter. The result is the execution of the target file as an independent software process.

6.6.10 Program example

I now give an example of a user constructed tangible program using the T-Logo environment. It explains the procedures to follow when constructing a tangible program that, when executed, results in an on-screen pattern. The steps follow the sequence detailed in Section 6.6.5.

First, the user is given a list of actions. The user then creates tangible program elements by combining wooden blocks and dowels with paper markers and matches the program elements to the list. She then arranges the Tangible Program Objects on the construction surface according to simple rules, keeping cognisance of the desired result. Finally, the user instructs the computer to execute the program. A series of software applications then interpret the objects and create a text file containing computer executable code. This code is then executed and the result displayed.

In the program example of Figure 6-45 (a), Cluster Marker Zones are shown using superimposed circles and mapped action/parameters are given next to each object. The order (1st, 2nd ...) in which the clusters will be executed is as indicated outside the photograph borders. Figure 6-45b illustrates the results once the interpreter has processed the program. In this figure, clusters have been replaced using equivalent Processing language statements. Figure 6-45 (c) shows the results formatted as is customary for text-based programs.



```
t.drawing = true;
t.forward(60);
t.left(15);
t.backward(10);
t.right(120);
```

```
t.drawing = true;
t.forward(60);
t.left(15);
t.backward(10);
t.right(120);
```

Figure 6-45 A program, the interpretation order, and the corresponding Processing language statements

Of particular interest is the cluster highlighted at the bottom right in Figure 6-45 (a) and copied in Figure 6-46 (a). This cluster consists of three Tangible Programming Objects that together represent an action (‘Right’) and two parameters (30 and 90) as shown in Figure 6-46 (b). Figure 6-46 (c) illustrates that the value (120) of the combined parameters has been calculated and is now used as a parameter to the Processing language statement: `t.right (120)`. The result is an executable program containing the Processing programming language code and is shown in Figure 6-47 (a). Also shown is the resultant graphic when this code is executed once (Figure 6-47b), and multiple times (Figure 6-47c), respectively.

6.6.11 Evaluation

The programming environment was designed to address research objectives 2, 3, and 4. I designed and executed a laboratory evaluation to determine the extent to which these objectives have been satisfied. I also compared my system with another prominent tangible programming environment called TERN.

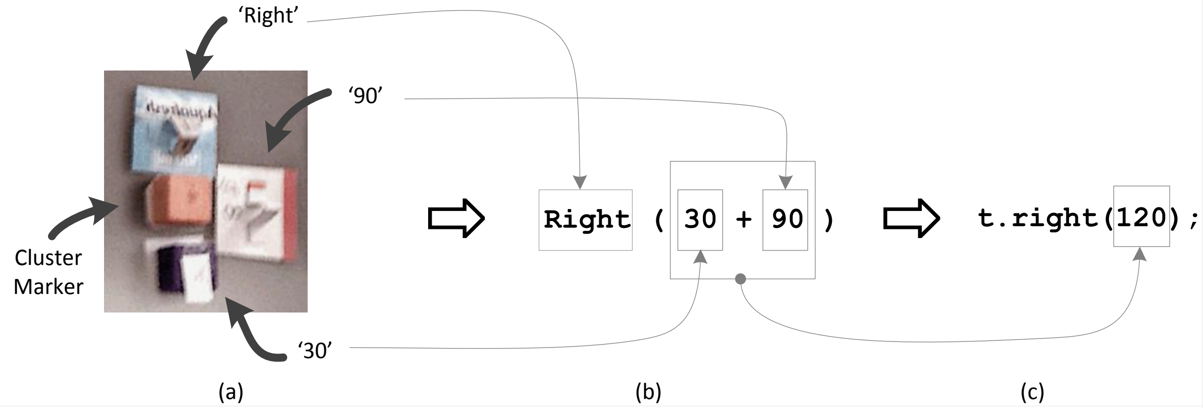
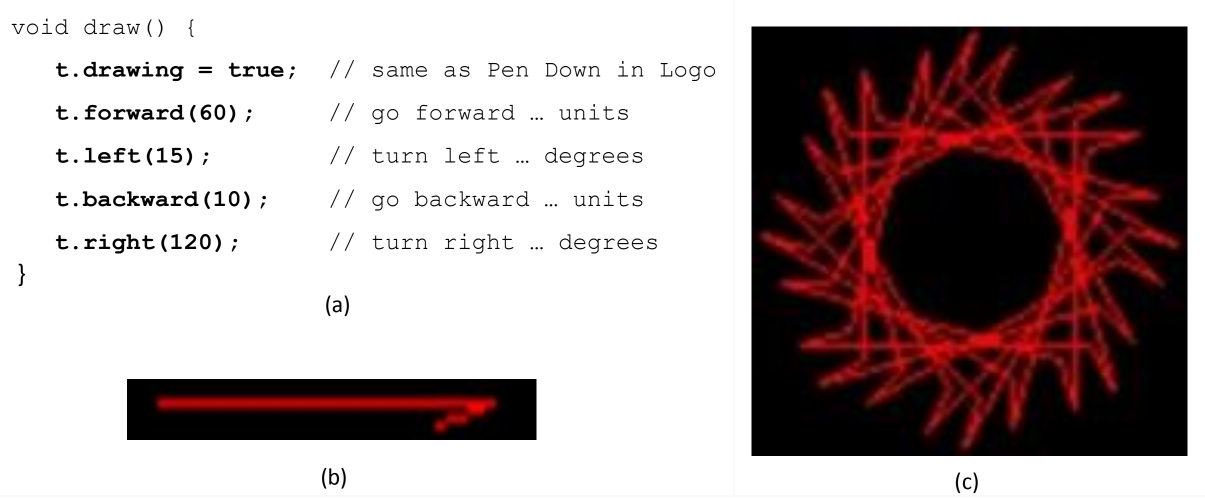


Figure 6-46 An example to illustrate how the logical values of two parameter objects are totalled



```
void draw() {
   t.drawing = true;   // same as Pen Down in Logo
   t.forward(60);      // go forward … units
   t.left(15);         // turn left … degrees
   t.backward(10);     // go backward … units
   t.right(120);       // turn right … degrees
}
```

Figure 6-47 A Processing language software routine and resultant graphics

6.6.11.1 Laboratory evaluation design

To successfully evaluate the design, the four T-Logo modes were sequentially activated. First, using the Map mode, I associated tangible objects with program elements. The program elements are actions, parameters, and Cluster Markers. I mapped the objects to elements as annotated in Figure 6-48.

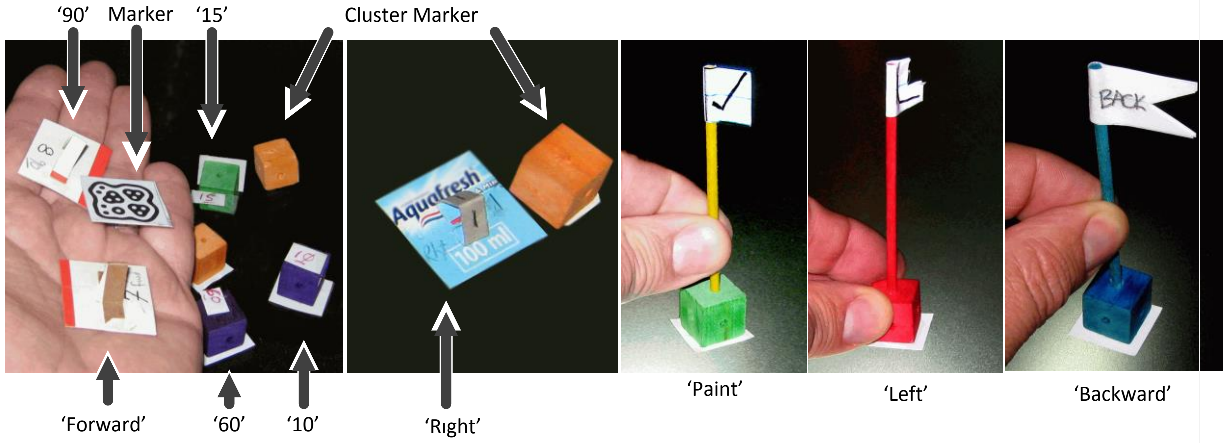
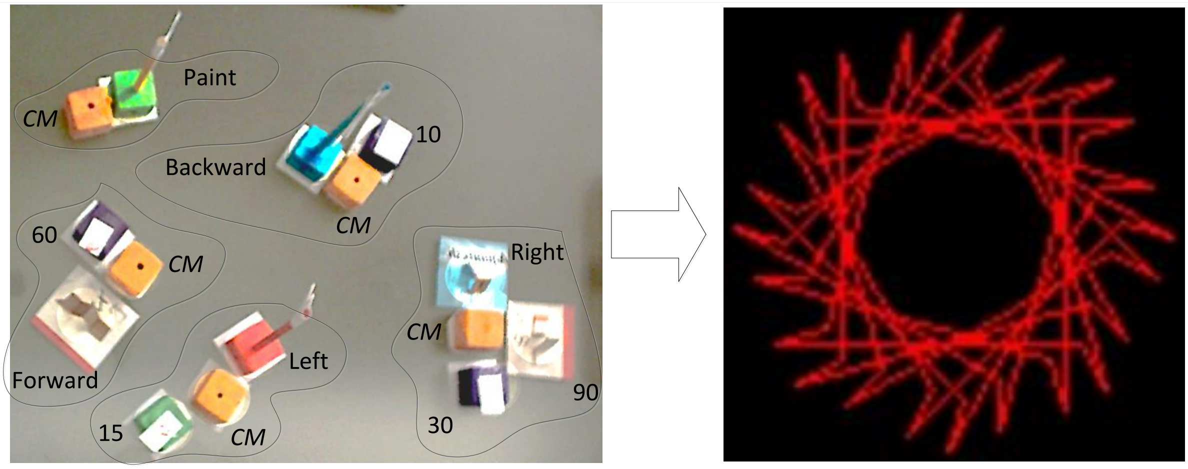


Figure 6-48 Objects and the commands and values they represent

Some objects were cut from cardboard stock while others were made using pens, scissors, paper and coloured wooden cubes and dowels. I then attached printed paper markers to the bottom of each object. Second, the T-Logo software Construct mode was activated. In this mode, I placed 10 Tangible Programming Objects and five Cluster Markers onto the construction surface (marked B in Figure 6-40) and arranged these into clusters. Figure 6-49 (a) shows the completed construction and ready to be interpreted. Finally, I activated the Interpret and Execute modes. Figure 6-49 (b) illustrates the visual output that resulted from this program.



(a)

(b)

Figure 6-49 A tangible program with annotations and the execution result

6.6.11.2 Laboratory evaluation results

The size of the Cluster Marker Zone is defined using a pre-set value in the interpreter software and I adjusted the value by trial and error until I achieved a usable system. Three variables need to be considered when the zone is adjusted. First, the Zone should be large enough to encompass multiple objects placed on the construction surface. Second, the Zone should not occupy an excessively large portion of the surface but allow sufficient space for other zones. Third, the minimum marker size is determined by both the usable resolution of the camera and the prevailing lighting conditions. Using my laboratory setup, I was able to arrange five clusters on a surface of approximately 400 x 400mm in size. Figure 6-49 (a) is an example of a five-cluster arrangement. I marked the Backward, Left, and Paint objects with paper flags to identify their functions (Figure 6-48).

The final observation is not a consequence of the T-Logo design but a result of the design implementation: A notable delay is evident between the time when the execution mode is activated and when the result is visible on the computer display. This is because my software makes multiple calls to the underlying computer operating system and every call delays the display update by approximately one second. I predict that this delay can be reduced by an order of magnitude if appropriate software engineering thinking is applied to the T-Logo implementation.

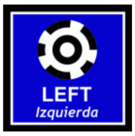
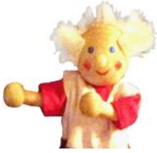
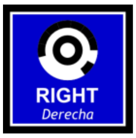
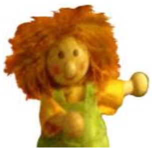
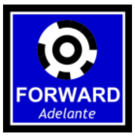
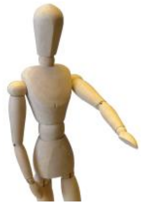
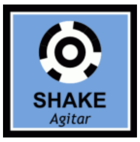
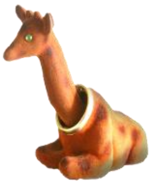
6.6.11.3 Comparison to the TERN language

I now compare the TERN (Horn 2009) language to my T-Logo by demonstrating how programs written using TERN can be expressed using my language. I chose TERN because it has previously been evaluated with the assistance of children in both the classroom (Horn 2009) and at an interactive science museum exhibition (Horn et al. 2008). Horn (2009) explains that the Karel the Robot (Pattis 1995) programming language inspired his own TERN language syntax. T-Logo, in turn, was inspired by the Logo (Harvey 2000) language. Partly due to their simple syntax, educators often choose either Karel the Robot or Logo as the language with which to introduce children to computer programming. A second reason these languages are popular amongst novices is that visually engaging designs are possible with very few instructions. Figure 6-49 (a) is an example of how an intricate pattern (Figure 6-49b) can be created using only a few instructions.

It is the user who decides what the T-Logo objects should look like. In the discussion that follows, I copied examples of personally meaningful objects from Smith and Gelderblom (2016) to illustrate how TERN sequences can be coded using T-Logo. I chose these objects because, to me, they represent the associated actions. For example, the toy giraffe's head is attached to its body using a spring and when touched its head shakes. A second example is the head of a dog I constructed out of modelling clay. I associate this object with a growling sound. Other T-Logo users may substitute the objects with their own personally meaningful signs.

Table 6-1 maps a subset of TERN actions to T-Logo objects whereas Table 6-2 does the same for the control structure in the examples. Table 6-3 illustrates how TERN sensors are mapped to their equivalents in the T-Logo language. Finally, Table 6-4 introduces TERN and T-Logo elements that do not fit the other tables. For example, I use a toy dog to indicate that the program should terminate.

Table 6-1 Tern actions and T-Logo objects

Description	TERN faceplate	T-Logo object	Description	TERN faceplate	T-Logo object
The programmed device will turn left.			The programmed device will turn right.		
The programmed device will move forward.			The programmed device will shake.		

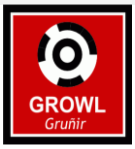
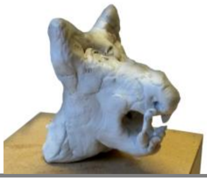
Description	TERN faceplate	T-Logo object	Description	TERN faceplate	T-Logo object
Make a growling sound.					

Table 6-2 Tern and T-Logo control structures

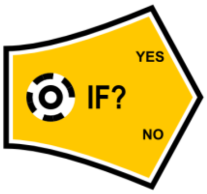
Description	TERN faceplate	T-Logo object
A conditional statement that tests the specified parameter.		Not applicable

Table 6-3 TERN sensing elements and their T-Logo equivalents

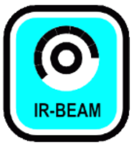
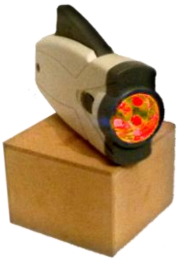
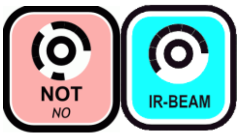
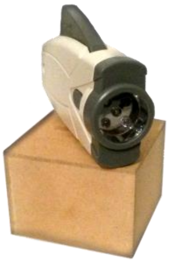
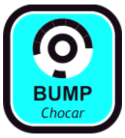
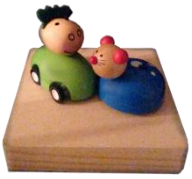
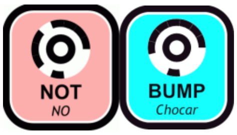
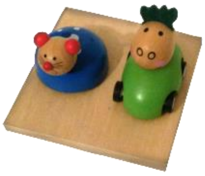
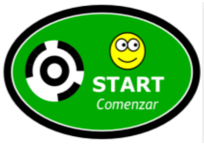
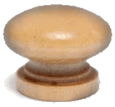
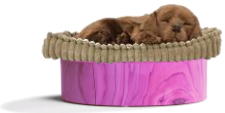
Description	TERN faceplate	T-Logo object	Description	TERN faceplate	T-Logo object
The IR-beam is triggered.			The IR-beam is not triggered.		
The bump sensor is triggered.			The bump sensor is not triggered.		

Table 6-4 Miscellaneous TERN and T-Logo elements

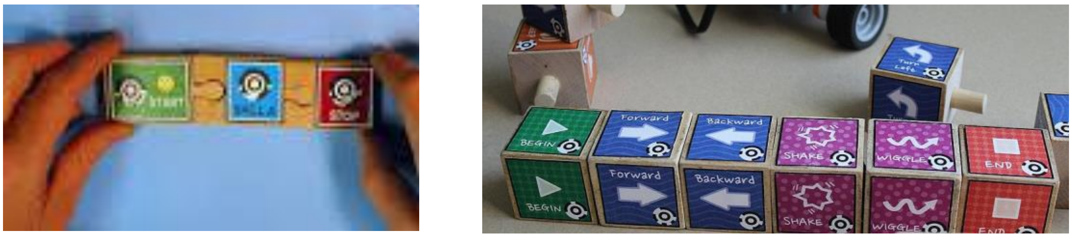
Description	TERN faceplate	T-Logo object	Description	TERN faceplate	T-Logo object
Indicates where the program begins.		Not applicable	The T-Logo Cluster Marker.	Not applicable	
			Indicates that the program should terminate at this position.	Not applicable	

6.6.11.4 TERN language elements

Horn (2009) adapted his TERN language and its physical design twice to suit his evaluation environments. For example, the TERN implementation used in a science museum comprises flat interlocking objects and includes control structures. In contrast, the classroom version is a simplified implementation consisting of cubes with dowels that push into each other. The dowel affords one cube to link with another. Not only did Horn adjust the physical appearance of the classroom version according to the user group but he also removed all control structures.

A prominent difference between TERN and T-Logo is the requirement to add objects that indicate the beginning and end of the program. In his kindergarten version, Horn uses green Start and red Stop objects for this. T-Logo does not use such indicators and instead assumes that the program originates on the left and continues to the right until no more clusters are detected. Horn's approach is arguably more robust as far as the interpreter is concerned and performs well in visually cluttered programming areas.

TERN's interlocking object design allows the image-processing algorithm to correctly identify the sequence in which objects are connected. The algorithm does this by searching for linked objects in the image. The examples given here do not explicitly show the puzzle-like physical constraints of Horn's implementation. Figure 6-50 (a) and Figure 6-50 (b) illustrate the respective mechanisms Horn devised for his science museum exhibition and the classroom application. Both these mechanisms result in a chain of objects on which the vision algorithm is based. T-Logo does not rely on mechanical links to define the chain of objects. Instead, it uses the Gestalt principle of proximity and good continuation to determine the object sequence.



(a) (b) (Horn 2009)

Figure 6-50 TERN puzzle and cubes examples

6.6.11.5 T-Logo language elements

The T-Logo interpreter produces executable code that implicitly repeats forever and is based on the Processing language's interpreter operation. The result is that, by default, the user's program executes repeatedly until stopped either explicitly by the user or by a programmed condition. In contrast, the TERN interpreter does not implicitly produce code that repeats but the user can code this behaviour into the program using statements such as REPEAT and WHILE along with their associated termination parameters. T-Logo's continuous execution design eliminates the need for an explicit over-all loop and therefore reduces the user's coding burden. However, the user may include a termination instruction if he wants the program to terminate at a specified point. I assume that the advantage of reduced program complexity outweighs any disadvantages and therefore based the T-Logo interpreter on an implied looped execution design.

6.6.11.6 Logic expression using the T-Logo language

Using the T-Logo language, all objects that represent logic conditions in a cluster must be true for actions in that cluster to be executed. Using pseudocode, this requirement can be expressed as

```
IF
(CONDITION-A = TRUE)  AND  (CONDITION-B = TRUE) AND  ...
THEN
do ACTION-1  and  do ACTION-2 and  ....
```

Clusters are evaluated independently of each other and therefore the logic represented in each cluster is independent of the logic in other clusters. The logic in a cluster is assumed to evaluate to TRUE when no logic operators are present. For example, when a cluster contains only an action then that action will always be executed. Conversely, when a cluster consists of both an action and a logic condition, then the action will only be executed when the logic evaluates to TRUE. The following three examples illustrate T-Logo equivalents of programs written using the TERN language.

Example 1 – Basic linear program

The first example is a simple sequential program with the TERN sequence in Figure 6-51 (a) copied from Horn's (2009) documentation where he includes a Start but not a Stop object. The absence of the red Stop object is consistent with the TERN language specification. The green Start element on the left indicates to the interpreter where the sequence starts and is followed by the Forward, Growl, Right, Forward, Shake, and Left instructions. The T-Logo sequence in Figure 6-51 (b) is interpreted from left to right and produces the same outcome as the TERN sequence but now using personally meaningful objects.

Tern's 'Bump' element detects if a bumper sensor has been activated. I use this to illustrate the T-Logo equivalent of TERN's 'IF' conditional statement. The TERN program in Figure 6-52 (a) loops indefinitely and instructs the toy to turn right if an obstacle is detected but move forwards otherwise. The same logic is implemented using my T-Logo language using two clusters (Figure 6-52b). The first cluster instructs the toy to turn right when a bump is detected and the second causes the toy to move forward if no bump is detected. There is no need for an element comparable to TERN's 'REPEAT' structure since the T-Logo interpreter always generates code that loops.

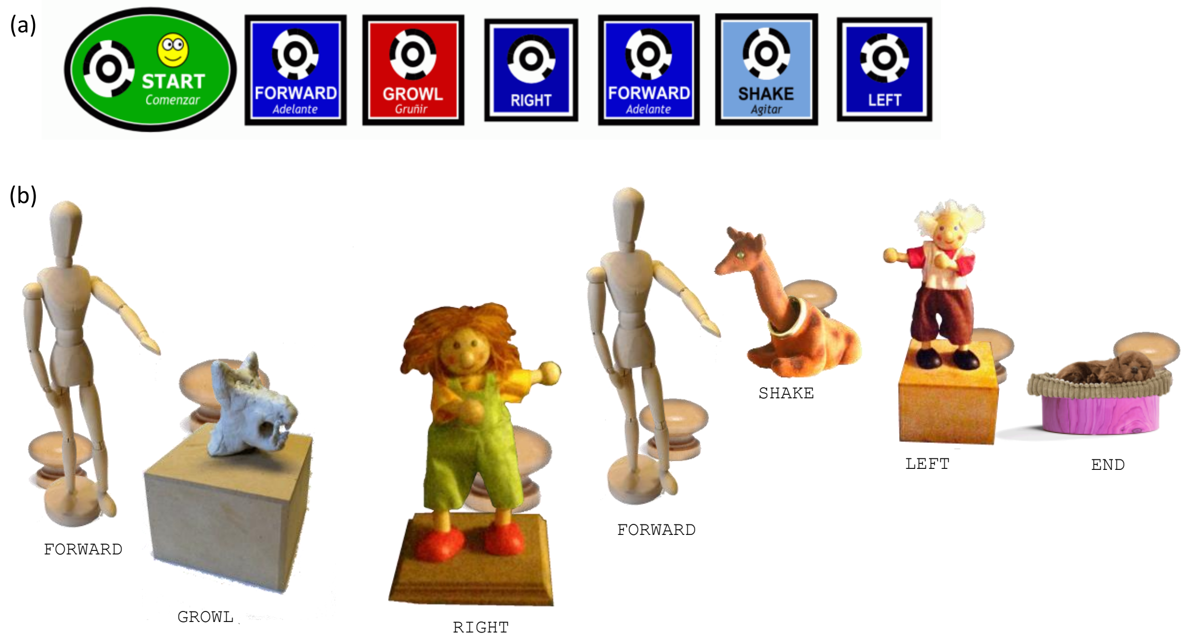


Figure 6-51 Example of a linear program

Based on Smith and Gelderblom (2016)

Example 2 - Conditional execution

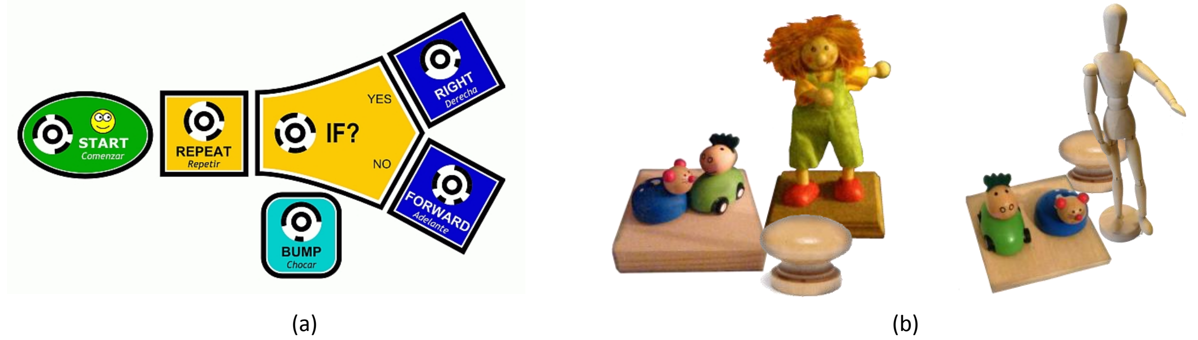


Figure 6-52 Example of a conditional statement in a loop

Based on Smith and Gelderblom (2016)

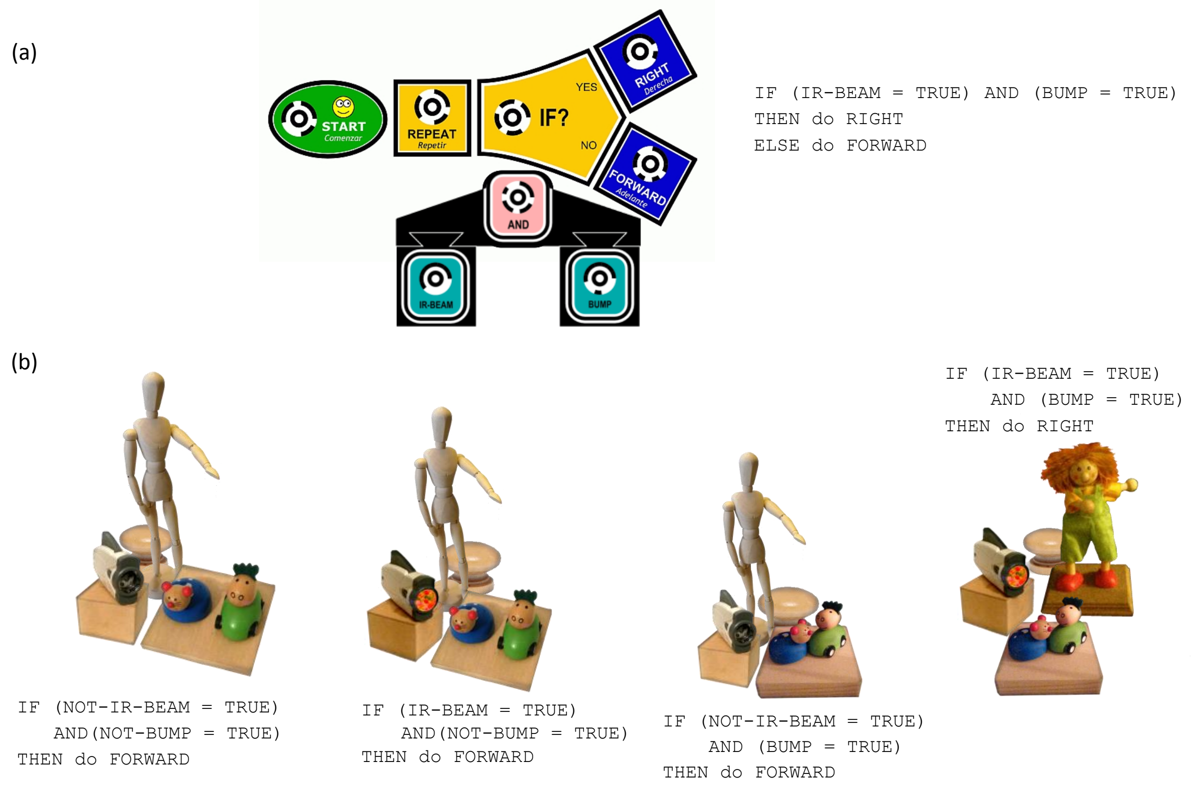
Example 3 - The logic AND conditional statement

The TERN AND expression in Figure 6-53 (a) can be written as follows using pseudocode:

```
IF (IR-BEAM = TRUE)   AND   (BUMP = TRUE)  THEN  do RIGHT
ELSE do FORWARD
```

The expression can also be written using OR operators and implemented as shown in Figure 6-53 (b):

```
IF (NOT-IR-BEAM = TRUE) AND (NOT-BUMP = TRUE) THEN do FORWARD
IF (IR-BEAM = TRUE) AND (NOT-BUMP = TRUE) THEN do FORWARD
IF (NOT-IR-BEAM = TRUE) AND (BUMP = TRUE) THEN do FORWARD
IF (IR-BEAM = TRUE) AND (BUMP = TRUE) THEN do RIGHT
```



```
IF (IR-BEAM = TRUE) AND (BUMP = TRUE)
THEN do RIGHT
ELSE do FORWARD
```

```
IF (NOT-IR-BEAM = TRUE)
    AND(NOT-BUMP = TRUE)
THEN do FORWARD
```

```
IF (IR-BEAM = TRUE)
    AND(NOT-BUMP = TRUE)
THEN do FORWARD
```

```
IF (NOT-IR-BEAM = TRUE)
    AND (BUMP = TRUE)
THEN do FORWARD
```

```
IF (IR-BEAM = TRUE)
    AND (BUMP = TRUE)
THEN do RIGHT
```

Figure 6-53 Example of the logic AND conditional statement

Based on Smith and Gelderblom (2016)

6.6.12 Critical discussion

T-Logo differs from other tabletop programming systems such as ReacTable (Jordà et al. 2010) and Turtan (Gallardo et al. 2008) in that users have a choice to either apply existing objects or create their own. I call these *personally meaningful objects* and I use the terminology *personally meaningful association* to describe the association action. Personally meaningful objects stand in contrast with other objects that already hold meaning in a community. I use the term *objective objects* to describe the latter.

The following illustrate objective and personally meaningful objects: The Eiffel tower is an objective object since members of Western societies will positively identify it with the city of Paris when shown a photograph of this landmark. Vincent van Gogh's 1886 oil-on-canvas painting titled "Shoes" is another example. Schapiro (1998) explores multiple interpretations of this painting, exemplifying the subjectivity an object can hold for the observer and thus supporting my own classification of this object.

I have shown that it is possible create an environment in which a user can assign tangible objects to program commands and parameters, arrange these on a construction surface, and then activate a series of software processes to interpret and execute this arrangement as a computer program. The resultant image on the computer screen was a direct result of the arrangement itself, combined with the meaning of the objects that constitute this arrangement. I have also given examples to illustrate that programs written using TERN (a prominent tangible language) can be coded using my T-Logo environment.

Objective languages and their signs are often well documented with the result that it is possible for one person to understand programs written by another. Using personally meaningful objects can be problematic since a third party has to apply great effort to decode someone else’s program when it includes personally meaningful objects. I anticipate that the only person capable of deriving the logic of a program constructed using personally meaningful objects is the author of the program. On the positive side, when the need for programmatic confidentiality arises it is possible to obscure the program logic and use personally meaningful objects.

Finally, The Design Science Research methodology allows for the discovery of new knowledge as iterations are completed. However, the methodology does not prescribe that the new knowledge generated from an iteration be incorporated into the immediate next iteration, or that the knowledge be incorporated at all. In my research, new knowledge from the first iteration was used to inform the second iteration. For example, the third iteration includes knowledge that emerged from the first two, but also incorporates additional knowledge with this being “Hand tools can shape soft rock”. The fourth iteration (Dialando) also incorporates further new knowledge that did not originate in the previous iterations. Therefore, it may appear that iterations do not “flow” from knowledge that emerged from the immediate previous iteration. However, knowledge from all iterations was “carried over” and informed the final iteration.

6.7 Conclusion

I applied the Design Science Research methodology to answer the three secondary research questions:

a. What program elements are suitable for a tangible programming environment in which the programmer can incorporate personally meaningful tangible objects?
b. How can a user associate personally meaningful tangible objects with program elements?
c. How can the arrangement of these personally meaningful objects be interpreted as program statements?

Although the overall research project was aimed at children of all ages, it emerged that each design iteration suits some users better than others. I base this on my observation that at the end of a particular design iteration it occasionally transpired that the user group that would derive most benefit from a design differs from the group initially considered at the onset of that particular iteration. Figure 6-54 illustrates both the designed-for and a prediction on the appropriate user groups across the five iterations. The study did not include a test to confirm this prediction.

At the onset of this research project, the tangible programming environment was conceptualised as using large objects with abstract signs. This original design concept was realised in Iteration 1 in the form of acrylic cubes and incorporated in the GameBlocks I programming environment. Evaluation of this programming environment revealed a number of problems. Most notable are the fragility of the acrylic material and the abstractness of the graphic signs. Knowledge thus gained informed the second iteration that I call GameBlocks II. Evaluation of the second iteration confirmed that the two problems with the first design had been satisfactorily addressed. Iteration 3 (RockBlocks) explored the application of handcrafted natural rock as a program object. Evaluation results revealed that individuals interpret objects differently. This observation was also made during the first evaluation sessions. The objective of the fourth (Dialando) design iteration was to confirm that it is possible to influence the digital domain using a tangible program.

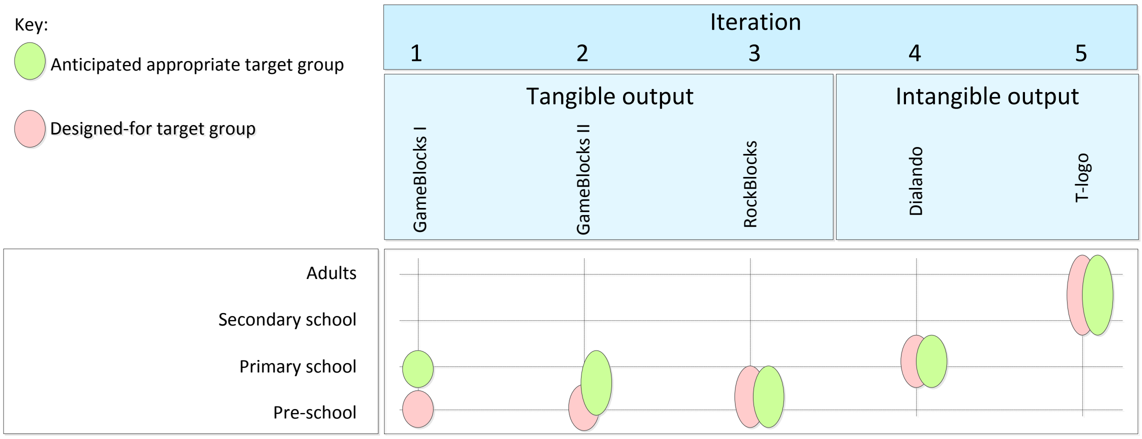


Figure 6-54 Design iterations, their target groups, and the nature of their outputs

The final design iteration that I call T-Logo introduced two design concepts that I call *cluster-based tangible programming* and *user-defined mapping*. It incorporates new knowledge that emerged from all the earlier iterations and in particular, the knowledge that fundamental Logo-like program elements can successfully be included in tangible programming environments. This confirmation resolves the first of the two secondary questions. I addressed individual interpretation differences in the final iteration through an association mechanism by which the user can incorporate personally meaningful objects. This mechanism is also the answer to the secondary research question that probes how a user can associate personally meaningful tangible objects with program elements.

To conclude, I have shown by means of the final iteration that it is feasible to implement a system that will simultaneously interpret five tangible object groupings and produce a visual pattern on a computer screen. I also confirmed that it is possible for a user to choose objects and then associate them with programmatic elements, have the user arrange the objects on a construction surface, and have a series of software processes interpret the arrangement as a program. Finally, I demonstrated a working system with which to construct tangible programs that incorporate both personally meaningful objects and the Gestalt principle of perceptual grouping by proximity. I will abstract in the Chapter 7 my findings as a model that reflects my derived constructs in a tangible program environment along with their relationships.

CHAPTER 7

RESEARCH CONTRIBUTION: A MODEL FOR A TANGIBLE PROGRAMMING ENVIRONMENT

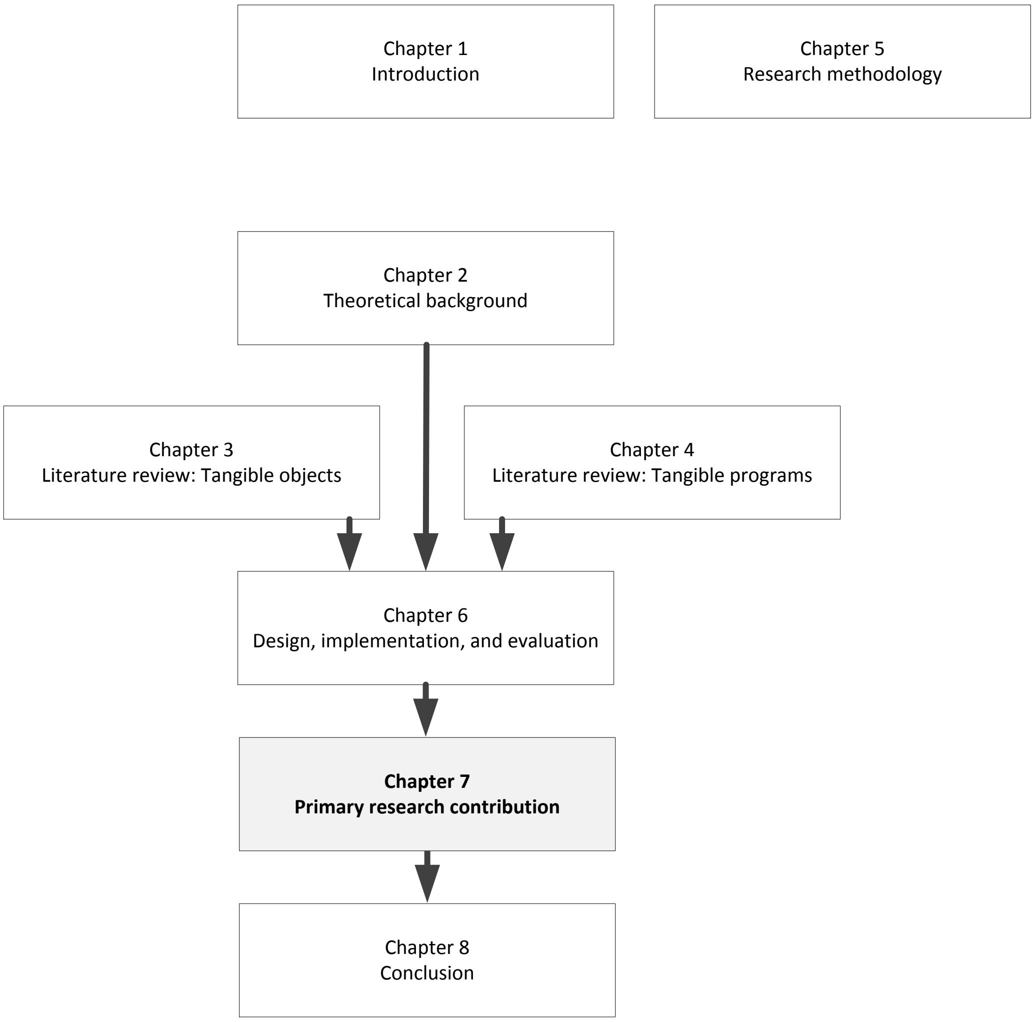

Chapter 1
Introduction
Chapter 5
Research methodology
Chapter 2
Theoretical background
Chapter 3
Literature review: Tangible objects
Chapter 4
Literature review: Tangible programs
Chapter 6
Design, implementation, and evaluation
Chapter 7
Primary research contribution
Chapter 8
Conclusion

Figure 7-1 Document structure

7.1 Introduction

In Chapter 6, I addressed the three secondary research questions:

a. What program elements are suitable for a tangible programming environment in which the programmer can incorporate personally meaningful tangible objects?
b. How can a user associate personally meaningful tangible objects with program elements?
c. How can the arrangement of these personally meaningful objects be interpreted as program statements?

I did this by iteratively designing, implementing, and evaluating tangible programming environments. The designs were informed both by outcomes of earlier iterations and by knowledge gained from my literature review and theoretical background chapters.

I demonstrated in Chapter 6 a system that can interpret groupings of tangible objects as a program. This was achieved by showing how a user can choose personally meaningful objects and associate them with program elements, arrange the objects, and let the system interpret the arrangement based on Gestalt principles.

I will now abstract the new knowledge that emerged in this study as a model to answer the primary research question as stated in Chapter 1:

In the context of existing tangible programming environments and considering how tangible objects are currently used when interacting with data, what are the constructs to incorporate into a model for creating tangible programming environments in which the relative positions of personally meaningful objects define the program, and how do these constructs interact and relate to one another?

The model identifies four actors, 16 constructs, and the interactions between the constructs. All actors share constructs but certain constructs relate to only a single actor. In the sections that follow, I will describe the model, demonstrate its application using three scenarios, and derive a programming language based on the scenarios.

7.2 Programming environment model

As I explained in Chapter 5, the Design Science Research process model (Vaishnavi & Kuechler 2013) includes operational principles and design theory as outputs of the research process. The output of this thesis is a tangible programming environment model. Figure 7-2 illustrates my model, including its actors, constructs and the relationships between the constructs. The four actors are the user, a language architect, a software developer, and a computing system. Constructs belong to one or more actors.

In the diagrams that follow, arrows indicate the direction of information flow between constructs and I apply Everest's (1976) fork notation (also known as Crow's Foot notation) to indicate when a construct relates to multiple instances of another construct. I next describe each actor and its constructs.

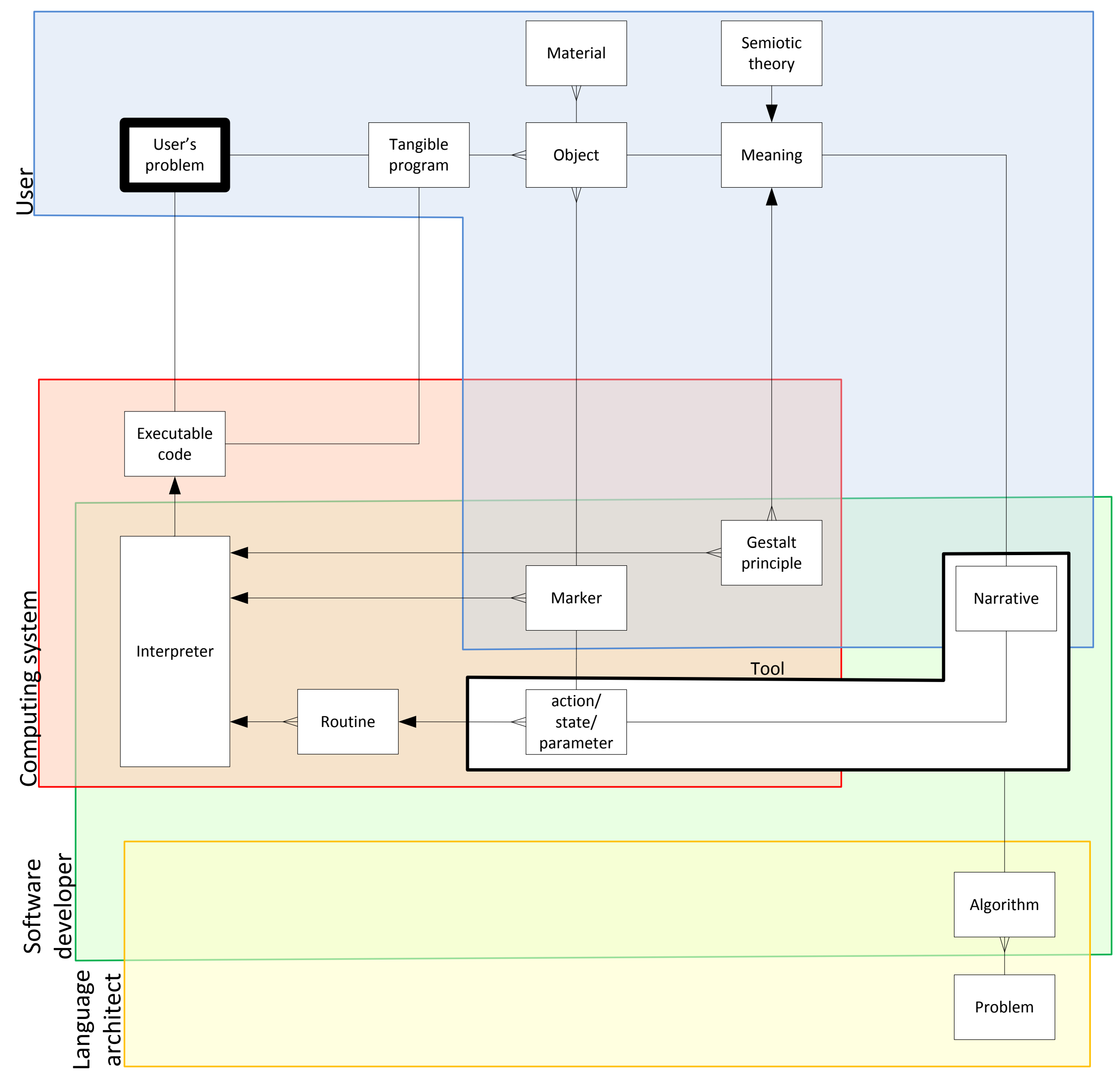


Figure 7-2 Model of a tangible programming environment

7.2.1 Language architect

The model is based on the assumption that the language architect is aware of the user's problem and interprets it. I further assume that the language architect is responsible for the design of algorithms that can help solve the problem. To illustrate the language architect's role consider the scenario in which a student is studying a text book in his room. In this scenario, the sunlight through the window is sufficient to read a book but the problem the student faces is the lack of natural light after sunset. Since the language architect is aware of the problem, he formulates an algorithm to turn on the desk lamp when the ambient light level drops below a set threshold. The role of the language architect is limited to formulating an algorithm that addresses the user's problem while the implementation of the algorithm remains the developer's responsibility. I define the algorithm and the problem as the first two constructs in my model (Figure 7-3).

More than one algorithm can address the same problem. For example, consider the scenario when the user wants security lights around his home turned on to deter criminals at night. One solution is an algorithm that takes as input the time of day and the times the sun sets and rises. It then specifies that the light be turned on if the time of day is between sunset and sunrise. An alternative algorithm is to track the outdoor ambient light level and switch the lights on and off as appropriate. Two algorithms therefore address the same user problem.

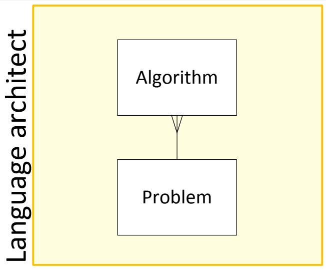


Figure 7-3 Language architect in the model

7.2.2 Software developer

The software developer actor (Figure 7-4) interprets the language architect's algorithm and he then derives a set of tools consisting of actions, states, and parameters. He also documents narratives that describe the tools. States reflect world reality while actions make changes to the world. Examples of states include room temperature and the time of day, while starting a line drawing and ending the drawing are examples of actions. Parameters refine the states of interest and give details on how actions should be applied to the world. For example, the software developer may decide to create a 'drawing' action that requires parameters to define the line colour, width, and texture. Another example extends the student scenario discussed earlier. In that scenario, the software developer may introduce parameters to control certain aspects of the desk lamp such as the switching threshold, the speed at which the light fades on and off, and the colour of the light.

The software developer then codes software routines based on the actions, states, and parameters. He also creates narratives that describe the tools in a way that the user can understand. Finally, the software developer instantiates a programming environment with which the user can link tangible objects to actions, states, and parameters. The linking process involves two steps. First, the user attaches a marker to his chosen object. He then activates the mapping function and selects which action, state, or parameter should be associated with the marker. The result is a tangible object that is linked to a tool.

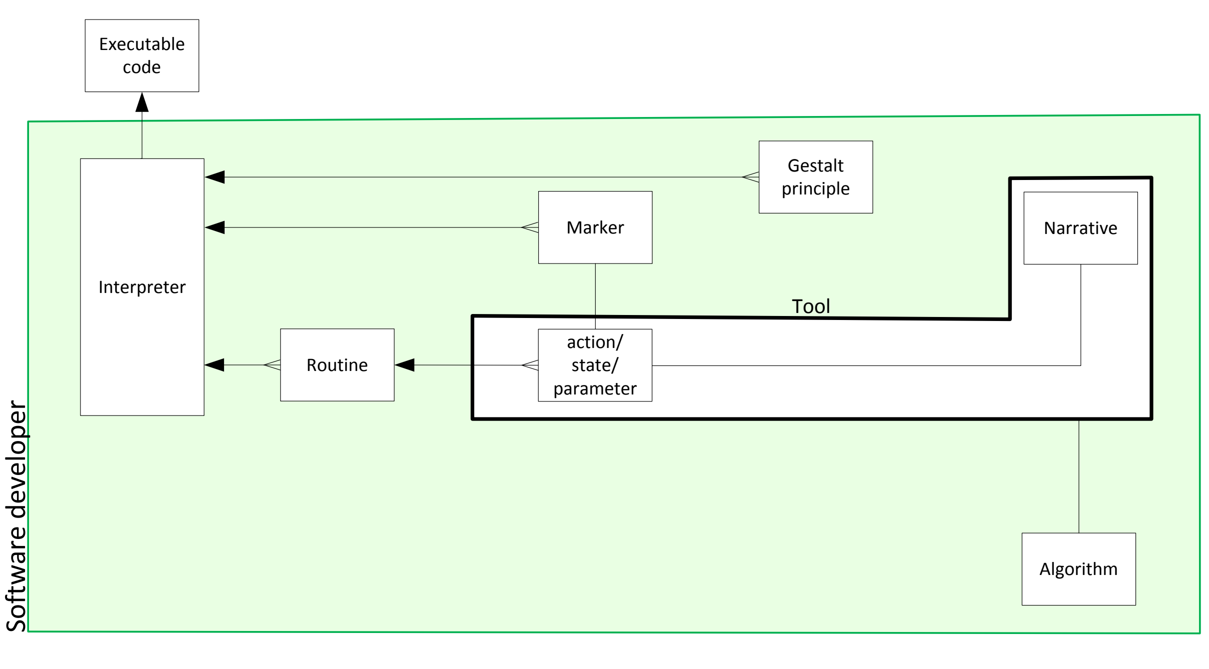


Figure 7-4 Software developer in the model

He also designs and implements an interpreter. The interpreter produces executable code by combining the routines. Routines are combined according to Gestalt principles, the particular tool an object represents, and the object sequence.

7.2.3 User

I discussed in Chapter 3 how objects are used to represent data. I also showed how data can be manipulated using tangible objects. In Chapter 4, I discussed systems that define programs by linking tangible objects. All these systems are rooted in knowledge bases originating from the domains of Engineering, Computer Science, and Information Technology.

In my research, I add to these systems knowledge from the Psychology research domain and in particular Gestalt principles and Semiotic theory. This knowledge helps describe and predict the way humans interpret objects. In Section 2.5.2, I used Saussure's linguistic sign model to demonstrate that a program element can be expressed in multiple ways. Based on the strength of Saussure's model, I argue that more than one object can represent a single program element. Moreover, I put it that a representation can hold personal meaning for the user.

By incorporating Gestalt principles and Semiotic theory into my model, I make it explicit that a user can either create a tangible object using available materials or choose an existing object to instantiate a personally meaningful tool representation. This establishes in the user's mind an association between the object and an action, state, or parameter. The user (Figure 7-5) therefore addresses his problem by constructing a program that incorporates personally meaningful objects that each represents a tool as described in the narrative.

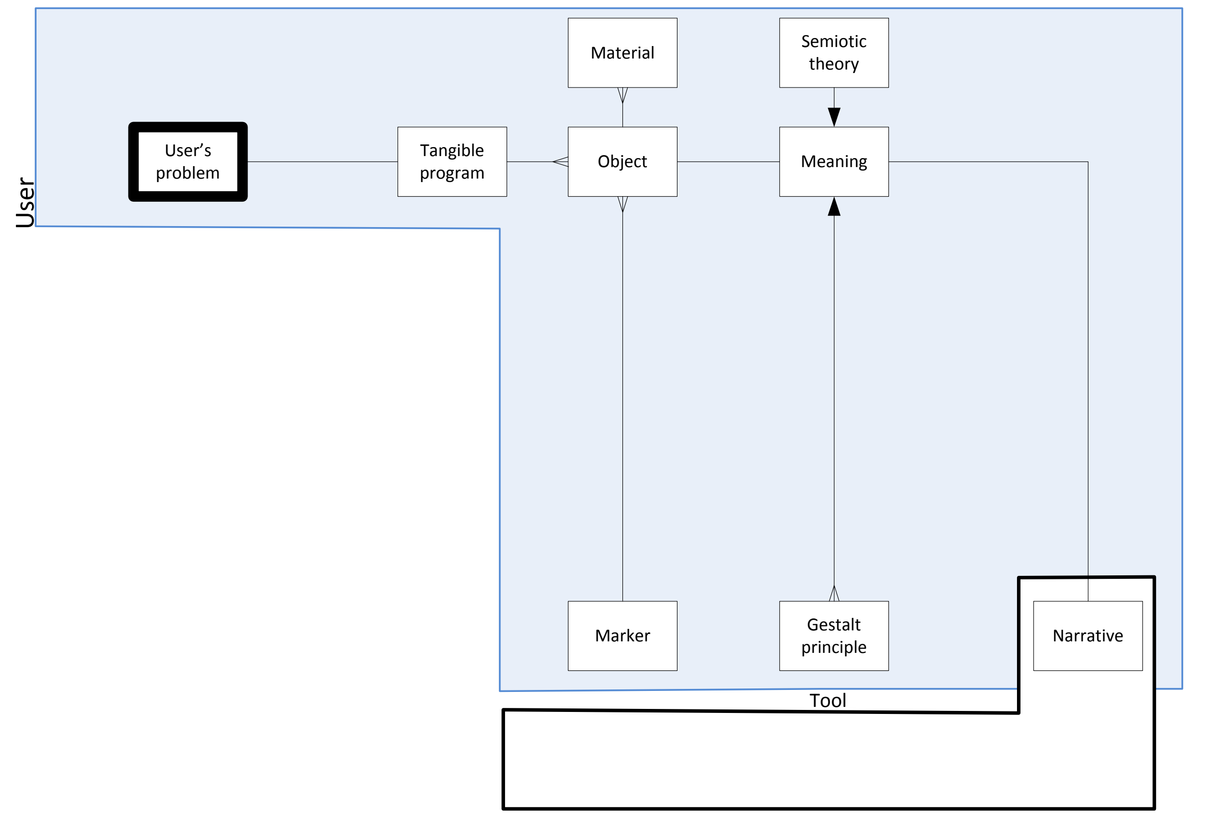


Figure 7-5 User in the model

I described Peirce's second trichotomy in Section 2.3.3.7 and explained that the sign representamen can relate to the semiotic object as an icon, an index, or a symbol. In terms of Peirce's sign model (Section 2.3.3), the tangible object in my model is the sign-representamen, the user is the interpretant, while the meaning that is attached to the object is Peirce's semiotic object. I elaborated that the icon has some perceived resemblance to its semiotic object, the index has a direct connection, and the symbol has a learned connection to the semiotic object.

However, and as I have elaborated in Section 2.3, the meaning that the user attaches to the object may not be the same for everybody. This could be the case if the user creates a symbol representamen, but less so when the representamen is an icon or index. This presents both an opportunity and a problem. An opportunity exists for the user to create confidential programs that are not trivial to decode by another person but the challenge is in maintaining a cohesive team when multiple users collaborate on a single program.

Although the user is free to select his object, the computing system has no way to identify it. A mechanism to overcome this problem is to attach an identifying marker to the object. The user therefore first attaches a marker to the object and then instructs the computing system to associate the marker with a narrative of his choice.

Both an object and its relative position to other objects hold meaning for the user. For example, when an object that represents time is in close proximity to another object that represents a numerical value, then the combination may represent the time of day to a user. I next give three examples that elaborate on the association procedure. In the first example, the user associates an object with the narrative “select an object to represent 23 degrees Celsius”. When this has been done, the object serves as a physical representation of a temperature parameter. For the second example, an object will represent the action of turning the student’s desk lamp on, and another object will be used to represent the state of a setting sun. When used together, these objects indicate that the desk lamp should turn on (an action) at sunset (a state). Finally, the developer may decide to make provision for two additional objects that will serve to indicate the rate at which the lamp should fade from its off condition to its on condition. In this case, two narratives can prompt the user and these are respectively “select an object to represent a quick change” and “select an object to represent slow change”. The third and fourth objects then represent a parameter to specify the rate of fading. By combining the first two objects with one of the last two, the user can define two rules. One rule dictates that the lamp should turn on slowly when the sun sets, while the second rule states that the lamp should turn on quickly.

7.2.4 Computing system

I described in the previous section how a user applies a marker to establish a logical link between an object and a tool. The user and the computing system differ in their perspectives of the object/marker combination. Whereas the user views a tangible program as an arrangement of one or more objects, the computing system (Figure 7-6) instead considers the program to be a collection of one or more markers. Unlike Horn’s (2009) TERN that prescribes the programming object’s shape and appearance and thereby also the program’s shape and appearance, my approach separates the user and computing system views. The result is that, when using a programming environment based on my approach, the user is free to change the object’s shape and aesthetic qualities without affecting the logic of his program.

In addition to fixing an object’s shape and appearance, TERN also dictates the relative positions in which objects should be placed. Horn’s designs often enforce these requirements using mechanical interlocking mechanisms. In contrast to these limitations, programming environments based on my model allow free-form object arrangements. This is possible since my model uses Gestalt principles to determine the sequence in which program elements are interpreted. The same principles help identify which elements belong together. I demonstrated in Section 2.4.4 that the user may arrange within limits the textual program elements according to personal preference. In addition, my model does not dictate the exact arrangement of program elements. Based on this description, I classify tangible programming languages derived from my model as being free-format languages.

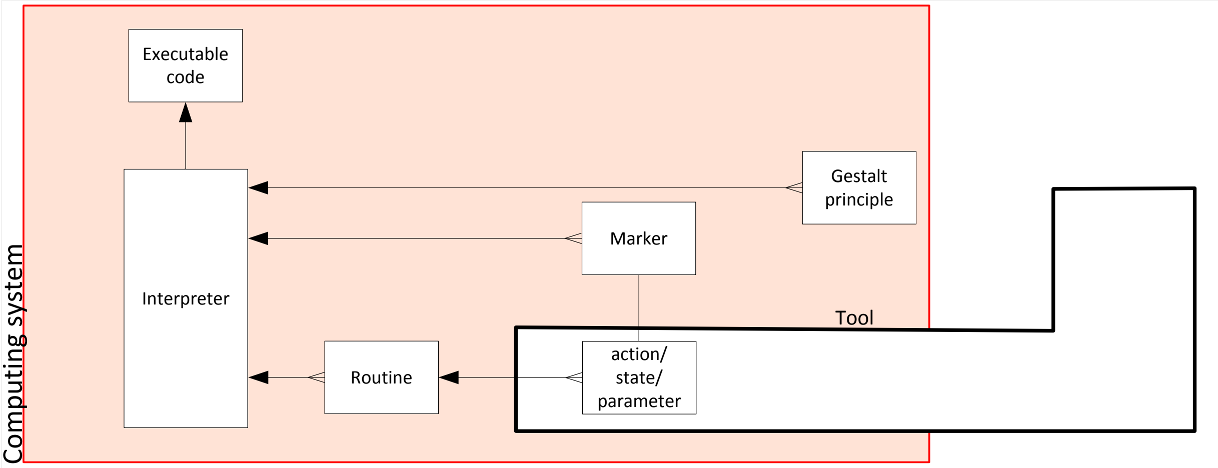


Figure 7-6 Computing system in the model

Using information from a sensor that can detect markers, the interpreter identifies the markers that constitute the program and their relative positions. The interpreter segments the collection of markers using the Gestalt principle of good continuation and the principle of grouping by proximity. It then produces executable code by concatenating the routines already associated with the markers. The outcome is executable code that addresses the user's problem.

7.3 Applying the model

I now demonstrate the model using three scenarios in a domestic environment. I first explain the user's problem and then give the language architect's algorithm. I then list the tool components and narratives that reflect the software developer's interpretation of the algorithm. I use pseudocode to express the software routines based on the tool components and narratives. Later, I derive the tangible programming language by combining the individual narratives. Finally, I express the language using a notation that the user can understand.

7.3.1 First scenario

In the first example, the language architect knows that the house occupant wants to maintain the indoor temperature at a chosen level. The language architect first interprets this need and then designs a language that includes a variable to represent temperature. He further specifies that positive integer values can be assigned to the temperature variable. The language definition also states that when the ambient temperature exceeds the desired level then the environment must be cooled. On the other hand, when the temperature is below the user's set value then the heater will be turned on. Using pseudo language, the architect's algorithm can be expressed as shown in Table 7-1.

Table 7-1 Programming the temperature

Algorithm			
Drive the room temperature to the user's set level.			
Tool components			
Action	Set the temperature to <desired_temperature>.	**<desired_temperature>**	A natural number.
Narrative		**Routine**	

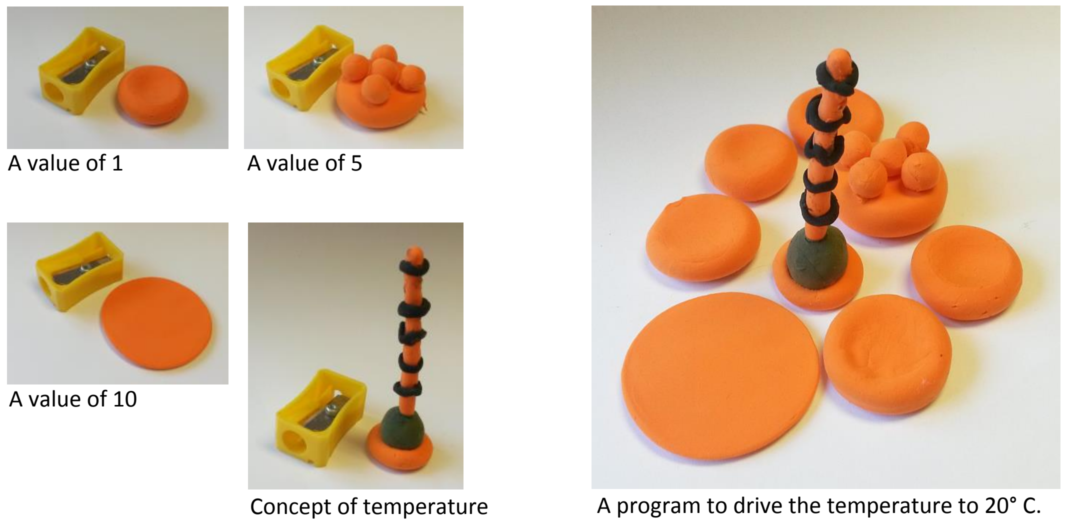
Make or use an existing object to represent
• *The concept of temperature. Use this object to indicate that you want to set the room temperature.*
• *1°C, 5°C, and 10°C respectively.*
Combine these objects to indicate the desired room temperature.
		`TemperatureControl (desired_temperature)`
		`  IF current_temperature > desired_temperature`
		`  THEN turn_heater_off and turn_cooler_on`
		`  ELSE turn_heater_on and turn_cooler_off`	

The software developer interprets the algorithm and creates a corresponding tool that has two components. The first component is an action and it represents the concept of setting the temperature. The second component represents the desired temperature value. This information is shared with the user by means of the given narrative. The components exist in both the physical and digital domains yet the user only interacts with the physical instances.

The software developer creates a software routine that compares the desired temperature to the current ambient temperature and controls the cooler and heater using support routines. These routines are turn_heater_off and turn_heater_on. I do not show the details of these routines.

The TemperatureControl routine and the desired_temperature parameter are digital domain components. The TemperatureControl routine is incorporated into the executable code only if both the action and the desired_temperature components are present in the tangible program.

The user makes or chooses personally meaningful objects to instantiate the tool components. Figure 7-7 displays objects I made for this example (the pencil sharpener has been included to show scale). She then attaches a marker to the object and activates the computing system's mapping mode. This mode presents to the user a list of tool components as described in the narrative. The user now places the object onto the input surface and selects a component from the list. The computing system then associates the marker with the selected component. This action establishes a logical link between the marker and the tool component. The user repeats this procedure until all the components have matching objects. Figure 7-7 includes a user program that drives the temperature to 20° C.



A value of 1

A value of 5

A value of 10

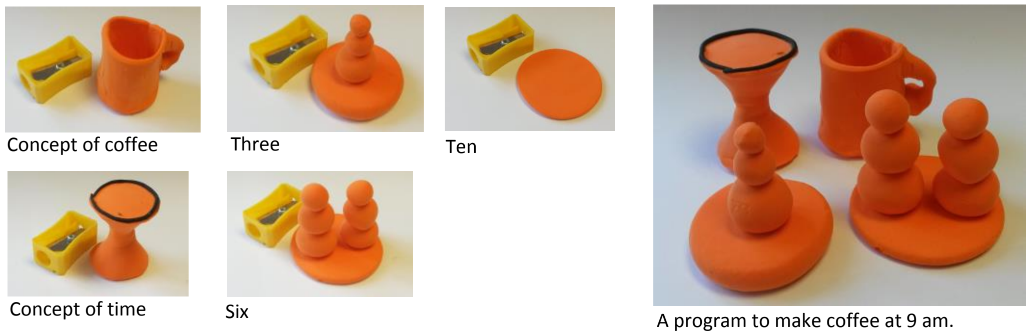
Concept of temperature

A program to drive the temperature to 20° C.

Figure 7-7 Examples of objects and a program to control the temperature

7.3.2 Second scenario

For the second example, I assume that the language architect knows that the user wants a cup of coffee during the day. To address the user's problem, the language architect designs an algorithm that will allow the user to specify the time when the coffee maker switches on. Table 7-2 captures the second algorithm, tool components, narrative, and routine. The tool consists of three components and these are the action to turn on the coffee maker, the state of the day, and the time of day (a parameter). A support routine that I call turn_coffee_maker_on interacts directly with the coffee maker. I do not elaborate on this routine. Figure 7-8 shows examples of objects I made for this scenario. It also demonstrates a program that turns on the coffee maker at 9 am.



Concept of coffee

Three

Ten

Concept of time

Six

A program to make coffee at 9 am.

Figure 7-8 Examples of objects and a program to make coffee

Table 7-2 Programming a coffee maker

Algorithm			
Turn on the coffee maker at a time the user specifies.			
Tool components			
Action	Turn on the coffee maker when <state>.	**<state>** **<time_value>**	The hour of day is <time_value>. A natural number.
Narrative		**Routine**	
"Make or use an existing object to represent			

• A coffee maker. Use this object when you want coffee.
• The concept of time. Use this object and the hour-indicators to set the time you want coffee.
• 3, 6, and 10 hours.
Combine these objects to indicate the time of day you want coffee.”
	`Coffee (state)`
	`  IF current_time = state`
	`  THEN turn_coffee_maker_on`

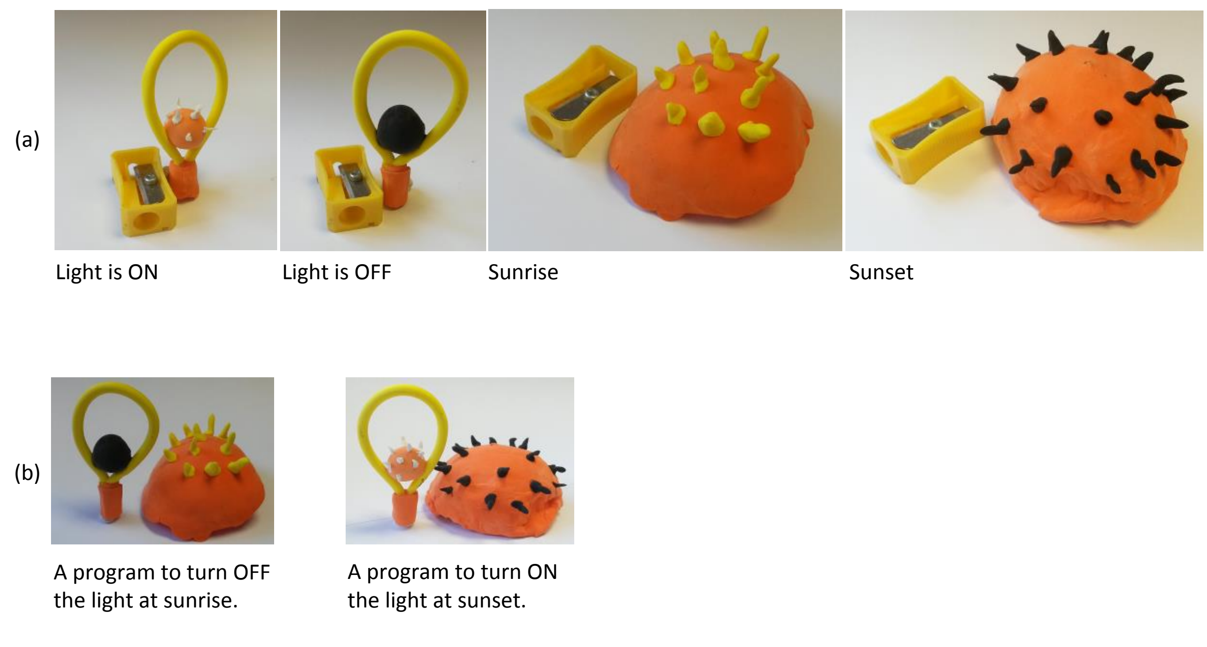
7.3.3 Third scenario

The final example is an algorithm to control a security light. It turns on the light when the ambient light level is low. The algorithm also turns off the light at sunrise. Table 7-3 documents the algorithm. The software developer implemented the algorithm using the Light_on and Light_off routines. He also defined two states and these are sunrise and sunset. Two supporting routines interact directly with the light with these being turn_light_on and turn_light_off. I do not describe them further. I created a set of objects that can be used to construct a program. Figure 7-9a shows these objects while Figure 7-9b includes two programs that incorporate the objects.

Table 7-3 Programming a security light

<table>
<tr><td colspan="4"><b>Algorithm</b></td></tr>
<tr><td colspan="4">Turn the security light on or off at sunset or sunrise as specified by the user.</td></tr>
<tr><td colspan="4"><b>Tool components</b></td></tr>
<tr><td><b>Action</b></td><td>Switch ON the light when <state&gt;.</td><td rowspan="2"><b>&lt;state&gt;</b></td><td rowspan="2">sunrise | sunset</td></tr>
<tr><td><b>Action</b></td><td>Switch OFF the light when <state&gt;.</td></tr>
<tr><td colspan="2"><b>Narrative</b></td><td colspan="2"><b>Routines</b></td></tr>
<tr><td colspan="2">“Make or use an existing object to represent
• The security light in its ON state.
• The security light in its OFF state.
• Sunrise.
• Sunset.
Combine these objects to indicate when the security light should be switched ON and OFF.”</td><td colspan="2"><code>Light_on (state)</code>
<code> IF current_state_of_the_sun = state</code>
<code> THEN turn_light_on</code>

<code>Light_off (state)</code>
<code> IF current_state_of_the_sun = state</code>
<code> THEN turn_light_off</code></td></tr>
</table>



Light is ON

Light is OFF

Sunrise

Sunset

A program to turn OFF the light at sunrise.

A program to turn ON the light at sunset.

Figure 7-9 Examples of objects and a program to control a security light

7.3.4 The derived language

It is possible to derive a programming language that can address the three user problems by combining the tool components. The resultant language consists of four actions, two value types, and three state types. Table 7-4 summarises the language. Finally and for the benefit of the user, the language can be expressed as a set of narratives. The narratives are derived using Table 7-1, Table 7-2, and Table 7-3 while Table 7-5 summarises the result.

Table 7-4 Language summary

Actions	Values	States
Set the temperature to <temperature_value>. Turn the coffee maker ON when <state>. Switch light ON when <state>. Switch light OFF when <state>.	temperature_value time_value	The hour of day is <time_value>. Sunrise Sunset

Table 7-5 The language as narratives

To achieve this...	Do this...
Control the temperature	Make or use an existing object to represent • The concept of temperature. Use this object to indicate that you want to set the room temperature. • 1°C, 5°C, and 10°C respectively. Combine these objects to indicate the desired room temperature.
Control the coffee maker	Make or use an existing object to represent • A coffee maker. Use this object when you want coffee. • The concept of time. Use this object and the hour-indicators to set the time you want coffee. • 3, 6, and 10 hours. Combine these objects to indicate the time of day you want coffee.
Control the security light	Make or use an existing object to represent • The security light when ON. • The security light when OFF. • Sunrise. • Sunset. Combine these objects to indicate when the security light should be switched ON and OFF.

7.4 Conclusion

I described in this chapter a model to guide the development of tangible programming environments. The model includes four actors and 16 constructs. The Gestalt principle and Semiotic theory constructs are particularly significant since this is the first programming model that explicitly incorporates these concepts. Finally, I demonstrated how the model may be applied using three scenarios in a domestic environment and I derived a tangible programming language to address the user’s problems.

CHAPTER 8

FINAL DISCUSSION AND CONCLUSIONS

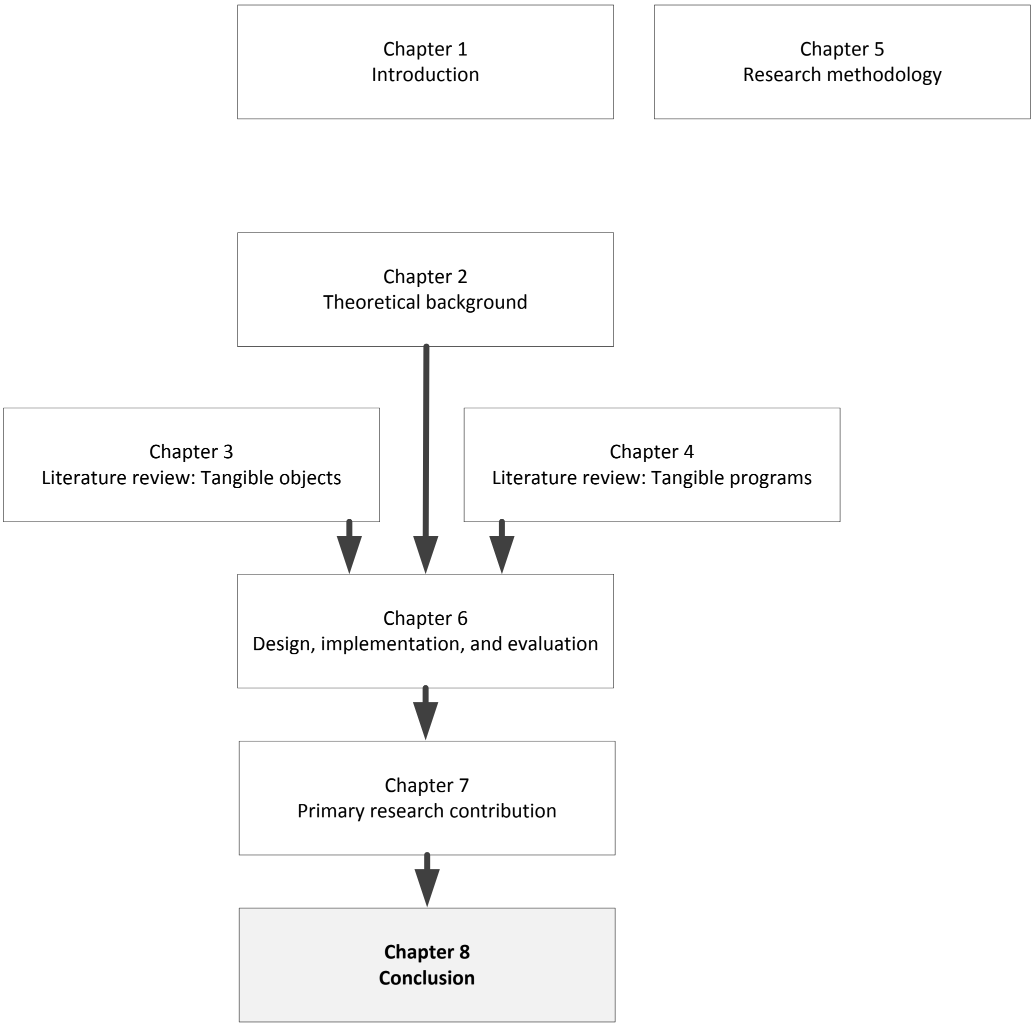

Chapter 1
Introduction
Chapter 5
Research methodology
Chapter 2
Theoretical background
Chapter 3
Literature review: Tangible objects
Chapter 4
Literature review: Tangible programs
Chapter 6
Design, implementation, and evaluation
Chapter 7
Primary research contribution
Chapter 8
Conclusion

Figure 8-1 Document structure

8.1 Introduction

In this research I set out to defend the following assertion:

> It is possible to derive a model for a tangible programming environment in which the relative positions of personally meaningful objects define the program.

To test my assertion, I formulated in Chapter 1 five research objectives. I copy them here and describe how they were achieved.

Research Objective
1

My first Research Objective was to determine a set of elements suited to a programming environment that incorporates personally meaningful tangible objects.

Research Objective 1 was achieved by studying the language elements used in tangible programming environments for young children. I determined that directional commands to control motorised toys are often used. I then asked children to make personally meaningful drawings (Section 6.3) that represent some of these commands. The results confirmed that the same command can have different representations that are each personally meaningful to the individual. Finally, I shaped artist clay (Section 6.4) by hand to verify that the commands can be instantiated as tangible objects.

Research Objective
2

This objective aimed to devise a mechanism by which a personally meaningful tangible object can be used as a program element.

Research Objective 2 was achieved by attaching printed fiducials to the object and assigning it to one of six actions or assigning a numerical value to it. This is possible using the software I developed.

Research Objective
3

Objective three was to devise a method by which an arrangement of one or more personally meaningful tangible objects can define a program statement.

I achieved Research Objective 3 by applying the Gestalt principle of grouping by proximity. I calibrated my software to determine the distance between objects and a special object called a Cluster Marker. All objects detected within a pre-set distance of the Cluster Marker are considered belonging to one group. A program statement is then derived using all the action and value objects belonging to that group.

Research Objective
4

My fourth objective was to devise a programming environment in which an arrangement of personally meaningful tangible objects can be interpreted as a program.

This objective was achieved by writing program statements to a text file as they are detected. The symbols captured to the file were then interpreted using the Processing language interpreter.

Research Objective **5**	Finally, I set out to develop a model that guides the creation of tangible programing environments where an arrangement of personally meaningful objects defines the program. I achieved this by identifying the theory and principles that describe and predict how individuals assign personal meaning to objects. I then combined this with the constructs that flowed from Research Objective 4.

8.2 Summary of findings

These are the main findings of my research:

1. I identified that parallel program threads can be constructed if the interpreter incorporates the Gestalt principle of good continuation when the objects are analysed. I illustrated the concept in Figure 4-5.
2. It became evident from my interaction with children that the order in which the system interprets objects should be simple. This prompted me to modify the original two-row configuration to one with a single row. I also observed that children have no difficulty when the sequence is interpreted from left to right along a straight trajectory (Section 6.2.4).
3. The meaning an object holds for an individual may not be the same as the meaning the same object holds for another person (Section 6.4). This observation prompted me to investigate programming environments in which the user chooses his objects.
4. The camera resolution dictates that a compromise between the object size and the number of program elements must be reached. This means that the larger the items are the fewer objects can be included in a program.
5. I determined that it is possible to create a system in which the user selects personally meaningful objects to represent program actions, parameters, and states as defined by another person.
6. My experiments demonstrated that it is possible to construct a program using personally meaningful objects in a static arrangement.
7. I confirmed in my workshops with children that it is possible to create programs using large objects that require both hands to manipulate. My final iteration confirmed that the same is possible using smaller objects that fit in the palm of the hand.
8. It is feasible to design a programming environment that incorporates the Gestalt principles of grouping by common region and grouping by proximity, yet I found that grouping by proximity (Section 6.6.2) is the simplest to implement.

9. My experiments confirmed that it is realistic to group program actions, states, and parameters by proximity and that the order of objects within a group is inconsequential and can be left to the user.
10. Text based environments prescribe that symbols are interpreted in a linear fashion. By design, the interpreters of these environments require that symbols that belong together are delimited using spaces or other predetermined symbols. Symbols cannot be arranged in an arbitrary fashion; instead, they must follow each other from left to right and top to bottom. Visual programming environments do not have this requirement and support free-form placement of program objects on the screen. In these environments, connections to a particular object may approach it from multiple directions but the links between objects must be visually explicit. My research has shown that, by applying the Gestalt principle of grouping by proximity, objects that belong together may be placed in any configuration within an arrangement as long as they remain separate from other arrangements.

My research also revealed a number of problems and I list them here:

1. My T-Logo environment in Chapter 6 is based on the Gestalt principles of good continuation and grouping by proximity. The design assumes that clusters are interpreted independently of each other. In addition, a group is only executed if the logic it represents evaluates true. Therefore, all logic expressions using the T-Logo language are AND logic composites (Section 6.6.9.6). All problems must therefore be expressed in this way. The result is that certain conditional expressions are cumbersome to express using T-Logo. Example 3 in Section 6.6.9.6 demonstrates the problem.
2. Having the user choose his objects is interesting since the result is a personalised program. However, this approach has the disadvantage that another person will find it challenging to decode a program that incorporates personally meaningful objects. I discuss this is Section 6.6.10.
3. Users were encouraged to express their thoughts while they used the system but participants found this uncomfortable and I subsequently abandoned this approach.

8.3 Conclusions

The purpose of my first secondary research question was to determine what program elements are suitable for a tangible programming environment in which the programmer can incorporate personally meaningful objects. To address this question, I derived a set of suitable program elements by combining my study of tangible programming environments in Chapter 4 with the knowledge I had gained from conducting tangible programming workshops with children in Chapter 6. This approach revealed that sequential statements consisting of elementary instructions work well when a user constructs a program using personally meaningful objects.

I formulated my next secondary research question to determine how a user can associate personally meaningful tangible objects with program elements. The way I answered this question was to develop in Chapter 6 software that links objects to program elements and then stores the links in a table. I explained the process in Section 6.6.7.1 and summarise it here. First, the software prompts the user with a narrative, thereby inviting her to select or construct an object that is personally meaningful. The user then attaches a marker to the bottom of the object and places the object onto a glass surface. Finally, the software detects the number embedded within the marker and updates the table. The result is a table that links tangible objects to program elements. Using this process, I was able to demonstrate in Chapter 6 how the user can associate personally meaningful objects with program elements.

My final secondary research question considered how program statements can be derived from an arrangement of personally meaningful objects. I addressed this by using the answer to my previous secondary research question to link objects to actions, states, and parameters. I then developed interpreter software to identify objects that represent actions/states/parameters and group them according to the Gestalt principle of grouping by proximity as described in Section 2.2.6.1. I explained in Section 6.6.7.3 that for all objects within a group, the software first identifies the action program elements and outputs corresponding textual statements to a file. If an action requires states or parameters, the interpreter will search for these within the group and append the result to the file. The action is discarded when the required state or parameter cannot be located and the next action in the group is processed. The interpreter considers the next group once all the actions within the current group have been processed. The interpreter terminates when all groups have been considered.

Having now answered my secondary research questions, I next addressed the primary question. My primary research question considered what constructs to include in a tangible programming environment model in which the relative positions of personal meaningful objects define the program, and how the constructs relate and interact with one another. The answer is contained in Chapter 2, Chapter 3, Chapter 4, and Chapter 6: I know from Chapter 2 that humans interpret objects differently and that the way objects are arranged influence the meaning we attach to them. This knowledge prompted me to include Semiotic theory and Gestalt principles in my model. Further, my study of tangible objects in Chapter 3 and tangible programming in Chapter 4 revealed that there is a need to link objects to program elements. A marker serves this purpose well and I therefore include it in my model. At the time when I developed my final iteration as described in Chapter 6, I became aware that four actors could be identified in a tangible programming environment. For example, I realised that it is not always a single actor who analyses a user's problem and develops a solution and I therefore distinguish between the language architect and the developer. My model appropriately differentiates between the architect, developer, and user actors. The computing system is the fourth actor and links the other three. Finally, I reflected that actions/states/parameters and narratives can independently describe program languages. I viewed actions/states/parameters and narratives as being two forms of the same tool but each with its own objective: The actions, states, and parameters are used by the interpreter whereas the narratives describe the language to the user. I therefore answered my primary research question by including in my model actors to represent the user, developer, architect, and computing system as well as primary constructs that include Semiotic theory, Gestalt principles, and tools.

I also identified secondary constructs that are either supported by or feed into those already mentioned. Secondary constructs directly related to the user's problem are the architect's interpretation of the problem, a tangible program that addresses the problem, software routines that are combined to implement a solution to the problem, and executable code that addresses the problem. Constructs related to the tangible object but not yet listed are objects, the materials from which objects are crafted, and markers to identify the objects. The final constructs that complete the model are the architect's algorithms, the narratives and the action/state/parameter sub-constructs, the meaning that a user attaches to the objects, and the interpreter.

Based on the outcomes of the three secondary research questions, the primary research question, and the subsequent model that I created in Chapter 7, I can confirm that it is possible to develop a model of a tangible programming environment in which the relative positions of personally meaningful objects define a program. I was therefore able to assert my thesis statement as formulated in Chapter 1.

8.4 Summary of contributions

I first list the new knowledge that emerged from my research and then discuss the implications of this knowledge.

8.4.1 New knowledge

It emerged from my literature review that there exist in tangible systems mechanisms that may be used for data exchange and for identity/position encoding. I derived from this a classification matrix (Section 3.2.3) that maps tangible systems to four quadrants. The quadrants serve to classify systems according to tethered/untethered data exchange mechanisms and passive/active encoding mechanisms.

Krippendorff (1989) uses a model to differentiate between the role of the designer and that of the user. My research considered the individual as being both the object's designer and its user. I adapted Krippendorff's model in Section 3.3 to reflect this dual role.

I proposed three new terminologies in Section 6.6.10 to describe tangible objects and how these are associated with digital counterparts. The first is "personally meaningful objects" and it refers to objects that have significance for the individual without influence from someone else. The second describes the process of associating a personally meaningful object to a specific digital counterpart, and I call this "personally meaningful association". Finally, "objective objects" refers to objects that hold a common meaning within a community. An example is a red cross on a white background that is most often associated with a hospital.

Ishii (2009) does not elaborate on the origin of the objects in his model as shown in Figure 3.3. My research extends his model by demonstrating that the user may design personally meaningful objects.

Objects are by default not suitable for use in a tangible program since no inherent link exists between the physical object and its digital equivalent and therefore such an object has no programmatic purpose. However, I demonstrated that a physical object can serve as a program element if the user first associates the object with a digital element. Once the association has been done I call the physical object a Tangible Programming Object and the user may access the associated digital element using the physical object.

I developed a Tangible Programming Environment model that includes Gestalt principles and Semiotic theory. Semiotic theory explains that the user may choose a physical representation of the program element that carries personal meaning whereas the Gestalt principle of grouping by proximity may be used to link program elements together.

Finally, I explained how a tangible object may represent actions, states and values in programs.

8.4.2 Implications of the new knowledge

I have shown that it is possible to construct a program in which the Gestalt principle of grouping by proximity is applied to define a program rule. Since the object sequence and relative positions within the grouping are inconsequential, the user is free to decide on the relative positions of objects in a group and any change in the relative positions will not affect the program logic.

Grouping objects according to their relative proximity frees the user from using mechanical constraints when constructing a program. It also provides more room for expression by giving the user the option to orientate the objects to personal preference.

Visually unique programs are possible if the user is given an opportunity to design the objects. A consequence is that programs can be visually interesting, distinct in appearance, and showcase the user's creative ability while still expressing valid program logic. This may be useful when the same program logic is applied in different environments that each has specific aesthetic requirements. The physical appearance of the program can be adjusted to suit the decorative theme and the objects designed to be similar to other objects already present. This is possible since the language symbols can be designed to aesthetically fit the environment using shapes, colours, and materials familiar to the user.

Objects can be tailored to persons with physical disabilities including arthritis and Parkinson's disease whereas visually impaired users can use their sense of touch to construct programs. The shape, size, colour, and texture are customisable to the individual.

The program construction process does not require electrical power at a time after an object has been associated with its digital counterpart. In addition, the program is not destroyed when power is removed and may be studied at a time when no power is available.

8.5 Suggestions for further research

My T-logo artefact in Chapter 6 does not exploit all the features that a fiducial marker offers. An example is the ability to derive the orientation of the object. If exploited, the program construction can be decoupled from the current optical scanning sequence as illustrated in Figure 6-43. It may then be possible for the tangible objects themselves to indicate the order of interpretation using their own orientation relative to other objects by "pointing" in the direction of the next object to be interpreted. I did not explore this concept further.

My model is based on markers that establish links between objects and program elements. In Chapter 6, I used markers defined by black patterns printed on white paper. Instead of a paper marker, it may be possible to use intrinsic properties of the object itself. For example, the outline or texture of the object's base may serve as an identifying marker.

Although not implemented, I considered other hand crafted artefact options (Smith 2014b). These include a wire-frame bicycle and a canoe with three rowers, an arrangement of rocks in the garden, a table-top Zen garden, and decorated rocks in the garden. The wire-frame bicycle in Figure 8-2 (a) is an abstraction of the program interpreter while the passenger is the program passing instructions to the cyclist. A similar abstraction holds for the canoe (Figure 8-2b) with its three rowers (the program) who direct the course of the canoe. These two concepts are in their infancy and need further exploration to determine their viability as programming metaphors.

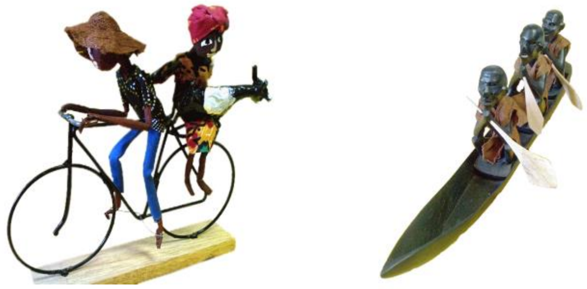


Figure 8-2 Figurines as programs

(Smith 2014b)

It may be possible to apply additional Gestalt principles when assigning actions, states, and parameters to objects. Based on Patten at al.'s (2001) suggested mechanisms that I describe in Section 3.2.4.1, it may be interesting to bind objects according to the Gestalt principle of grouping by proximity to another tangible object, grouping by proximity to a projected image, and grouping by common region when an object is placed within an area.

It is possible to make objects that indicate a direction and my third iteration (discussed in Section 6.4) is an example. It may be interesting to develop a programming environment in which a program rule "points" to the next object to be interpreted. That object would in turn point to another, and so on. Program environments designed in this way will not be based on the Gestalt principle of good continuation but instead on another principle still to be identified.

My implementations do not consider all the physical properties that an object possesses. It may be worthwhile to consider how to apply additional object properties to guide the programming process by incorporating mechanical constraints and other Gestalt principles.

Finally, I have shown that program elements include actions, states, and parameters. A suitable set of elements is most often determined by either the language architect or the developer who implements the programming language. Even though some programming environments allow the user to extend the element set, it remains the user's burden to understand the meaning that the developer has associated with a sign. This research considered a programming environment that allows the user to choose personally meaningful objects to represent program elements thereby removing the requirement for the user to use another person's signs. A question that remains unanswered is whether the language should be designed by a user who has personally experienced the problems being addressed, or should this be done by a language architect? My model in Chapter 6 is based on having the architect make language design decisions while a software developer expresses the result using narratives for the benefit of the user. The alternative is for the user to design the language and have a software developer implement it.

To conclude, in this thesis I have shown that it is possible to develop a model of a tangible programming environment that includes Gestalt principles and Semiotic theory where Semiotic theory explains that the user can choose a physical representation of the program element that carries personal meaning while the Gestalt principle of grouping by proximity predicts that objects can be arranged to appear as if linked to each other.

REFERENCES

Aghaei, M.B., 2015. Influence of Peirce’s semiotics on the signification of literary discourse. *Linguistics and Literature Studies*, 3(1), pp.24–30.

Aho, A.V., Lam, M.S., Sethi, R. & Ullman, J.D., 2007. *Compilers: Principles, techniques, and tools*, Pearson Addison Wesley.

Aish, R., Frankel, J.L., Frazer, J.H. & Patera, A.T., 2001. Computational construction kits for geometric modeling and design (Panel Abstract). *Proceedings of the 2001 symposium on Interactive 3D graphics*, pp.125–128. Available at: http://66.102.1.104/scholar?num=100&hl=en&lr=&safe=active&q=cache:1zrmpRP2VJgJ:pages.cpsc.ucalgary.ca/ saul/601.56.puis/marks-paper2.pdf+.

Alborzi, H., Druin, A., Montemayor, J., Platner, M., Porteous, J., Sherman, L., Boltman, A., Taxén, G., Best, J., Hammer, J., Kruskal, A., Lal, A., Schwenn, T.P., Sumida, L., Wagner, R. & Hendler, J., 2000. Designing StoryRooms: Interactive storytelling spaces for children. In *DIS ’00 proceedings of the conference on designing interactive systems*. New York City, New York, United States: ACM Press, pp. 95–104.

Andersen, P.B., 1997. *A theory of computer semiotics: Semiotic approaches to construction and assessment of computer systems*, Cambridge: Cambridge University Press.

Anderson, D., Frankel, J.L., Marks, J., Agarwala, A., Beardsley, P., Hodgins, J., Leigh, D., Ryall, K., Sullivan, E. & Yedidia, J.S., 2000. Tangible interaction + graphical interpretation: A new approach to 3D modeling. In *SIGGRAPH ’00 proceedings of the 27th annual conference on computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., pp. 393–402.

Aronson, L., 2011. *HTML manual of style: A clear, concise reference for hypertext markup language (including HTML5)* fourth., Tokyo: Addison-Wesley.

Asch, M., 2002. *Textbook of cognitive psychology*, New Delhi: Ivy Publishing House.

Banich, M.T. & Heller, W., 1998. Evolving perspectives on lateralization of function. *Current directions in psychological science*, 7(1), pp.1–2.

Banzi, M., 2009. *Getting started with Arduino*, O’Reilly.

Barthes, R., 1982. *Camera Lucisa: Reflections on photography*, New York: Hill and Wang.

Beckmann, C. & Dey, A., 2003. *Siteview: tangibly programming active environments with predictive visualization*, Intel Research Berkeley.

Bekker, A. & Kruger, C., 2006. *Usability and educational review on: GameBlocks and Body PingPong*, Test and Data Services.

Bencina, R., Kaltenbrunner, M. & Jorda, S., 2005. Improved topological fiducial tracking in the reacTIVision system. *Computer vision and pattern recognition workshop*, p.99.

Bentley, J., 1986. Programming pearls: Little languages. *Communication of the ACM*, 29(8), pp.711–721. Available at: http://doi.acm.org/10.1145/6424.315691.

Berger, A.A., 2012. *Media analysis techniques*, California: Sage Publications.

Bernstein, D. & Nash, P.W., 2008. *Essentials of psychology*, Boston: Houghton Mifflin Company.

Bers, M.U. & Horn, M.S., 2010. Tangible programming in early childhood. *High-tech tots: childhood in a digital world*, p.49.

Blackburn, S., 1996. *The Oxford dictionary of philosophy: Oxford paperback reference*, Oxford University Press.

Blackwell, A.F., 2002. First steps in programming: A rationale for attention investment models. In *Proceedings of the IEEE 2002 symposia on human centric computing languages and environments*. IEEE, pp. 2–10.

Blackwell, A.F., 2006. Psychological issues in end-user programming. In H. Lieberman, F. Paterno, & V. Wulf, eds. *End user development*. Human-computer interaction series. Springer Netherlands, pp. 9–30.

Blackwell, A.F. & Hague, R., 2001a. AutoHAN: an architecture for programming the home. In *Human-centric computing languages and environments, 2001. Proceedings IEEE symposia on*. IEEE, pp. 150–157.

Blackwell, A.F. & Hague, R., 2001b. Designing a programming language for home automation. *Proceedings of the 13th annual workshop of the psychology of programming interest group (PPIG 2001)*, pp.85–103.

Blum, J., 2013. *Exploring Arduino: Tools and techniques for engineering wizardry*, Indianapolis: John Wiley & Sons, Inc.

Bopry, J., 2002. Semiotics, epistemology, and inquiry. *Teaching & learning*, 17(1), pp.5–18.

Boradkar, P., 2010. *Designing things: A critical introduction to the culture of objects*, Berg Publishers.

Boyd, F., 1994. Humane slaughter of poultry: The case against the use of electrical stunning devices. *Journal of agricultural and environmental ethics*, 7(2), pp.221–236. Available at: http://dx.doi.org/10.1007/BF02349038.

Bregman, A.S., 1994. *Auditory scene analysis: The perceptual organization of sound*, London: The MIT Press.

Brookshear, J.G., 2012. *Computer Science: An overview* eleventh., Pearson Education, Inc.

Buechley, L., Quilt Snaps. Available at: http://web.media.mit.edu/ leah/grad_work/projects/snaps/quilt_snaps.html. Accessed 18 October 2013.

Buechley, L., Elumeze, N., Dodson, C. & Eisenberg, M., 2005. Quilt Snaps: A fabric based computational construction kit. In *IEEE international workshop on wireless and mobile technologies in education*. IEEE Computer Society.

Buechley, L., Elumeze, N. & Eisenberg, M., 2006. Electronic/computational textiles and children's crafts. In *Proceedings of the 2006 conference on Interaction design and children*.

Burrell, G. & Morgan, G., 2005. *Sociological paradigms and organisational analysis*, Hants: Ashgate Publishing Limited.

Camarata, K., Do, E.Y.-L., Johnson, B.R. & Gross, M.D., 2002. Navigational blocks: Navigating information space with tangible media. In *IUI '02 proceedings of intelligent user interfaces*. San Francisco, California, USA: ACM Press, pp. 31–38.

Cantor, R.M., 2003. Roentgen semiotic grammar. *Semiotica*, 146, pp.69–79.

Chandler, D., 2007. *Semiotics: The basics* second., Abingdon, Oxon: Routledge.

Chung, K., Shilman, M., Merrill, C. & Ishii, H., 2010. OnObject: gestural play with tagged everyday objects. In *Adjunct Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*. UIST '10. New York, New York, USA: ACM, pp. 379–380. Available at: http://doi.acm.org/10.1145/1866218.1866229.

Cohn, N., 2013. *The visual language of comics: Introduction to the structure and cognition of sequential images*, London: Bloomsbury.

Colledge, M.A.R., 1979. Sculptors' stone-carving techniques in Seleucid and Parthian Iran, and their place in the "Parthian" cultural milieu: Some preliminary observations. *East and West*, 29(1/4), pp.221–240. Available at: http://www.jstor.org/stable/29756517.

Cooper, D.R. & Schindler, P.S., 2006. *Business research methods* ninth., New York, NY: McGarw-Hill/Irwin.

Danesi, M., 2004. *Messages, signs, and meanings: A basic textbook in semiotics and communication theory* M. Danesi, ed., Toronto: Canadian Scholars' Press Inc.

Danesi, M. ed., 2009. *Dictionary of media and communications*, London: M.E. Sharpe.

Dawson, C., 2009. *Introduction to research methods: A practical guide for anyone undertaking a research project* fourth., Oxford: How To Books.

Deely, J.N., 1990. *Basics of semiotics advances in semiotics*, Indiana University Press.

Desolneux, A., Moisan, L. & Morel, J.-M., 2008. *From Gestalt theory to image analysis: A probabilistic approach*, Springer.

DeVilliers, M.R., 2012. Research methodologies, innovations and philosophies in software systems engineering and information systems. In M. Mora, O. Gelman, A. Steenkamp, & M. S. Raisinghani, eds. Hershey: Information Science Reference.

Dietz, P.H. & Eidelson, B.D., 2009. SurfaceWare: Dynamic tagging for Microsoft Surface. In *Proceedings of the 3rd international conference on tangible and embedded interaction*. TEI '09. Cambridge, United Kingdom: ACM, pp. 249–254. Available at: http://doi.acm.org/10.1145/1517664.1517717.

Dix, A., Finlay, J., Abowd, G.D. & Beale, R., 2004. *Human-computer interaction*, Pearson Prentice Hall.

Do, E.Y.-L. & Gross, M.D., 2007. Environments for creativity: A lab for making things. In *C&C '07 proceedings of the 6th ACM SIGCHI conference on creativity & cognition*. Washington, DC, USA: ACM, pp. 27–36.

Dollery, B., 2003. Understanding the psychology of programming. Available at: http://www.devx.com/DevX/Article/11659.

Druin, A., 1999. Cooperative inquiry: developing new technologies for children with children. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*. Pittsburgh, Pennsylvania, United States: ACM Press, pp. 592–599.

Druin, A., 2002. The role of children in the design of new technology. *Behaviour and information technology*, 21(1), pp.1–25.

Druin, A., 2009. *Mobile technology for children: Designing for interaction and learning*, Morgan Kaufmann.

e2esoft, 2012. Available at: http://www.e2esoft.cn/vcam/, accessed 20 January 2017.

Eco, U., 1976. *A theory of semiotics*, Bloomington: Indiana University Press.

Everaert-Desmedt, N., 2011. Peirce's semiotics. In L. Hébert, ed. *Signo*. Rimouski (Quebec). Available at: http://www.signosemio.com/peirce/semiotics.asp.

Everest, G., 1976. Basic data structure models explained with a common example. In *Proc. Fifth Texas conference on computing systems (Austin, TX)*. Long Beach, CA: IEEE Computer Society publications office, pp. 39–45.

Eysenck, M.W. & Keane, M., 2000. *Cognitive psychology: A student's handbook* fourth., Psychology Press.

Farina, G., 2014. Some reflections on the phenomenological method. *Dialogues in philosophy, mental and neuro sciences*, 2(7), pp.50–62. Available at: http://www.crossingdialogues.com/Ms-A14-07.htm.

Fawcett, R.P., 1992. Review of "A theory of computer semiotics: Semiotic approaches to construction and assessment of computer systems" by P. B. Andersen. Cambridge University Press 1990. *Computational linguistics*, 18(4), pp.555–562. Available at: http://dl.acm.org/citation.cfm?id=176313.976487.

Fernaeus, Y., 2007. *Let's make a digital patchwork: Designing for children's creative play with programming materials*. PhD thesis, Stockholm University.

Fernaeus, Y. & Tholander, J., 2006. Finding design qualities in a tangible programming space. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*. New York, NY, USA: ACM Press, pp. 447–456.

Fischer, G., 2001. User modeling in human–computer interaction. *User modeling and user-adapted interaction*, 11(1-2), pp.65–86.

Fischer, G. & Scharff, E., 2000. Meta-design: Design for designers. In *Proceedings of the 3rd conference on designing interactive systems: Processes, practices, methods, and techniques*. ACM, pp. 396–405.

Fischer, T. & Lau, W., 2006. Marble track music sequencers for children. In *IDC '06: Proceeding of the 2006 conference on interaction design and children*. New York, NY, USA: ACM Press, pp. 141–144.

Fiske, J., 1990. *Introduction to communication studies* second., London: Routledge.

Fitzmaurice, G.W., Ishii, H. & Buxton, W.A.S., 1995. Bricks: Laying the foundations for graspable user interfaces. In *CHI '95 proceedings of the SIGCHI conference on human factors in computing systems*. Denver, Colorado, United States: ACM Press/Addison-Wesley Publishing Co., pp. 442–449.

Franks, D.D., 2006. Handbook of the sociology of emotions: The neuroscience of emotions. In Springer, pp. 38–62.

Frazer, J., 1995. *An evolutionary architecture*, London: Architectural Association.

Frei, P., Su, V., Mikhak, B. & Ishii, H., 2000. Curlybot: designing a new class of computational toys. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '00. The Hague, The Netherlands: ACM, pp. 129–136. Available at: http://doi.acm.org/10.1145/332040.332416.

Gallardo, D., Julia, C.F. & Jorda, S., 2008. TurTan: A tangible programming language for creative exploration. In *TABLETOP 2008. 3rd IEEE International workshop on horizontal interactive human computer systems*. pp. 89–92.

Gardener, M., 2012. *Beginning R: The statistical programming language*, Indianapolis, IN: John Wiley & Sons, Inc.

Ghaoui, C. ed., 2005. *Encyclopedia of human computer interaction*,

Gift, N. & Jones, J.M., 2008. *Python for Unix and Linux system administration*, O'Reilly Media, Inc.

Goldstein, E.B., 2010. *Sensation and perception* eighth., Belmont: Wadsworth, Cengage Learning.

Gopnik, A., Griffiths, T.L. & Lucas, C.G., 2015. When younger learners can be better (or at least more open-minded) than older ones. *Current directions in psychological science*, 24(2), pp.87–92.

Gorbet, M.G., 1998. *Beyond input devices: A new conceptual framework for the design of physical-digital objects*. Masters thesis, MIT Media Lab.

Gorbet, M.G. & Orth, M., 1997a. Triangles and the digital veil. *FleshFactor: Informationsmacschine mensch (Ars Electronica festival catalog)*, pp.280–283.

Gorbet, M.G. & Orth, M., 1997b. Triangles: Design of a physical/digital construction kit. In *DIS '97 proceedings of the conference on designing interactive systems*. Amsterdam, The Netherlands: ACM Press, pp. 125–128.

Gorbet, M.G., Orth, M. & Ishii, H., 1998. Triangles: Tangible interface for manipulation and exploration of digital information topography. In *CHI '98 proceedings of the SIGCHI conference on human factors in computing systems*. Los Angeles, California, United States: ACM Press/Addison-Wesley Publishing Co., pp. 49–56.

Gordon, I.E., 2004. *Theories of visual perception* third., Hove: Psychology Press.

Greenberg, I., 2007. *Processing: Creative coding and computational art*, friendsofED.

Gregory, R.L. & Zangwill, O.L. eds., 1987. *The Oxford companion to the mind*, Oxford: Oxford University Press.

Griffin, T., 2010. *The art of Lego Mindstorms: NXT-G programming*, No Starch Press.

Grönvall, E., Marti, P., Pollini, A. & Rullo, A., 2006. Active Surfaces: a novel concept for end-user composition. In *NordiCHI '06: Proceedings of the 4th Nordic conference on Human-computer interaction*. ACM Press, pp. 96–104.

Groome, D., Dewart, H., Esgate, A., Kemp, R., Towell, N. & Gurney, K., 1999. *An introduction to cognitive psychology*, Psychology Press.

De Guzman, E. & Hsieh, G., 2003. *Function composition in physical chaining applications*, Berkeley, CA: UC Berkeley. Available at: http://hci.stanford.edu/research/papier-mache/pubs/cs260paper.pdf.

Hague, R., Robinson, P. & Blackwell, A., 2003. Towards ubiquitous end-user programming. In *Adjunct Proceedings of UbiComp*. pp. 169–170.

Hall, S., 2007. *This means this, this means that: A user's guide to semiotics*, Laurence King.

Hansen, M., Goldstone, R.L. & Lumsdaine, A., 2013. What makes code hard to understand? *ArXiv e-prints*, (1304.5257). Available at: http://arxiv.org/abs/1304.5257.

Harvey, B., 2000. Logo. In *Encyclopedia of computer science*. Chichester, UK: John Wiley and Sons Ltd., pp. 1035–1038. Available at: http://dl.acm.org/citation.cfm?id=1074100.1074558.

Hein, J.L., 1996. *Theory of computation: An introduction*, London: Jones and Bartlett Publishers International.

Helm, P.A. Van der, 2011. The influence of perception on the distribution of multiple symmetries in nature and art. *Symmetry*, 3(1), pp.54–71. Available at: www.mdpi.com/journal/symmetry.

Helm, P.A. van der, 2014. Oxford handbook of perceptual organization. In J. Wagemans, ed. Oxford, United Kingdom: Oxford University Press.

Henle, M., 1985. Rediscovering Gestalt psychology. In S. Koch & D. E. Leary, eds. *A century of psychology as science*. Washington: American Psychological Association, pp. 100–120.

Henle, M., 1989. Some new Gestalt psychologies. *Psychological Research*, 51(2).

Hevner, A.R., March, S.T., Park, J. & Ram, S., 2004. Design science in information systems research. *MIS Quarterly*, 28(1), pp.75–105. Available at: http://www.jstor.org/stable/25148625.

Hochberg, J., 2007. *In the mind's eye: Julian Hochberg on the perception of pictures, films, and the world* M. A. Peterson, B. Gillam, & H. A. Sedgwick, eds., New York: Oxford University Press, Inc.

Hofstee, E., 2006. *Constructing a good dissertation: A practical guide to finishing a Master's, MBA or PhD on schedule*, Sandton, South Africa: EPE.

Holmquist, L., Redström, J. & Ljungstrand, P., 1999. Token-based access to digital information. In H.-W. Gellersen, ed. *Handheld and ubiquitous computing*. Lecture notes in computer science.

Springer Berlin / Heidelberg, pp. 234–245. Available at: http://dx.doi.org/10.1007/3-540-48157-5_22.

Horn, M., 2007. Topcodes. Available at: http://users.eecs.northwestern.edu/ mhorn/topcodes/, Accessed 17 June 2014.

Horn, M.S., 2009. *Tangible computer programming: Exploring the use of emerging technology in classrooms and science museums*. PhD thesis, Computer Science, Tufts University.

Horn, M.S. & Jacob, R.J.K., 2006. Tangible programming in the classroom: A practical approach. In *CHI '06 extended abstracts on human factors in computing systems*. New York, NY, USA: ACM Press, pp. 869–874.

Horn, M.S. & Jacob, R.J.K., 2007. Tangible programming in the classroom with Tern. In *CHI '07 extended abstracts on human factors in computing systems*. San Jose, CA, USA: ACM Press, pp. 1965–1970.

Horn, M.S., Solovey, E.T. & Jacob, R.J.K., 2008. Tangible programming and informal science learning: Making TUIs work for museums. In *Proceedings of the 7th international conference on interaction design and children*. IDC '08. Chicago, Illinois: ACM, pp. 194–201. Available at: http://doi.acm.org/10.1145/1463689.1463756.

Hoven, E. van den & Eggen, B., 2004. Tangible Computing in Everyday Life: Extending Current Frameworks for Tangible User Interfaces with Personal Objects. *EUSAI, LNCS*, 3295, pp.230–242.

Hugdahl, K. & J.Davidson, R., 2003. *The asymmetrical brain*, London, England: The MIT Press.

Ishii, H., 2004. Bottles: A transparent interface as a tribute to Mark Weiser. *IEICE Transactions on Information and Systems*, E87-D(6), pp.1299–1311.

Ishii, H., 2008a. Tangible bits: Beyond pixels. In *TEI '08 proceedings of the 2nd international conference on tangible and embedded interaction*. Bonn, Germany: ACM, pp. xv–xxv.

Ishii, H., 2008b. The tangible user interface and its evolution. *Communications of the ACM*, 51(6), pp.32–36.

Ishii, H., 2009. Human-computer interaction: Design issues, solutions and applications. In A. Sears & J. A. Jacko, eds. CRC Press, pp. 141–159.

Ishii, H., Fletcher, H.R., Lee, J., Choo, S., Berzowska, J., Wisneski, C., Cano, C., Hernandez, A. & Bulthaup, C., 1999. musicBottles. In *SIGGRAPH '99: ACM SIGGRAPH 99 conference abstracts and applications*. Los Angeles, California, United States: ACM, p. 174.

Ishii, H., Mazalek, A. & Lee, J., 2001. Bottles as a minimal interface to access digital information. In *CHI '01 extended abstracts on human factors in computing systems*. CHI EA '01. Seattle, Washington: ACM, pp. 187–188. Available at: http://doi.acm.org/10.1145/634067.634180.

Ishii, H., Ratti, C., Piper, B., Wang, Y., Biderman, A. & Ben-Joseph, E., 2004. Bringing clay and sand into digital design: Continuous tangible user interfaces. *BT technology journal*, Volume 22, Number 4, pp.287–299.

Ishii, H. & Ullmer, B., 1997. Tangible bits: Towards seamless interfaces between people, bits and atoms. In *CHI ’97 proceedings of the SIGCHI conference on human factors in computing systems*. Atlanta, Georgia, United States: ACM Press, pp. 234–241.

Ishii, H., Wisneski, C., Brave, S., Dahley, A., Gorbet, M., Ullmer, B. & Yarin, P., 1998. ambientROOM: Integrating ambient media with architectural space. In *CHI 98 Conference Summary on Human Factors in Computing Systems*. CHI ’98. Los Angeles, California, USA: ACM, pp. 173–174. Available at: http://doi.acm.org/10.1145/286498.286652.

Jacoby, S., Josman, N., Jacoby, D., Koike, M., Itoh, Y., Kawai, N., Kitamura, Y., Sharlin, E. & Weiss, P.L., 2006. Tangible user interfaces: Tools to examine, assess and treat dynamic constructional processes in children with developmental coordination disorders. In *Proceedings of the 6th international conference on disability, virtual reality & associated technology*. ICDVRAT/University of Reading.

Jacucci, C., 2007. *Media literacy in responsive physical environments*. PhD thesis, Institute for communicating and collaborative systems, School of Informatics, University of Edinburgh.

Jacucci, C., Jacucci, G., Wagner, I. & Psik, T., 2005. A manifesto for the performative development of ubiquitous media. In *Proceedings of the 4th decennial conference on critical computing: Between sense and sensibility*. CC ’05. Aarhus, Denmark: ACM, pp. 19–28. Available at: http://doi.acm.org/10.1145/1094562.1094566.

Jacucci, C., Pain, H. & Lee, J., 2006. Media co-authoring practices in responsive physical environments. In *People and Computers XIX—The Bigger Picture*. Springer, pp. 391–407.

Jappy, T., 2013. *Introduction to Peircean visual semiotics*, London: Bloomsbury Academic.

Jetsu, I., 2008. *Tangible user interfaces and programming*. University of Joensuu. Available at: ftp://ftp.cs.joensuu.fi/pub/Theses/2008_MSc_Jetsu_Ilja.pdf.

Jones, S.P., Blackwell, A.F. & Burnett, M., 2003. A user-centred approach to functions in Excel. In *Proceedings of the eighth ACM SIGPLAN international conference on functional programming*. ICFP ’03. Uppsala, Sweden: ACM, pp. 165–176. Available at: http://doi.acm.org/10.1145/944705.944721.

Jordà, S., Julià, C.F. & Gallardo, D., 2010. Interactive surfaces and tangibles. *XRDS*, 16(4), pp.21–28.

Jorda, S., Kaltenbrunner, M., Geiger, G. & Bencina, R., 2005. The reacTable. In *Proceedings of the International Computer Music Conference (ICMC 2005)*. Barcelona, Spain.

Kahn, K., 1996. Drawings on napkins, video-game animation, and other ways to program computers. *Commun. ACM*, 39(8), pp.49–59.

Kaltenbrunner, M., 2009. reacTIVision and TUIO: A tangible tabletop toolkit. In *Proceedings of the ACM international conference on interactive tabletops and surfaces*. ITS ’09. Banff, Alberta, Canada: ACM, pp. 9–16. Available at: http://doi.acm.org/10.1145/1731903.1731906.

Kaltenbrunner, M. & Bencina, R., 2007. reacTIVision: A computer-vision framework for table-based tangible interaction. In *TEI ’07 proceedings of the 1st international conference on tangible and embedded interaction*. ACM, pp. 69–74.

Kasschau, R.A., 2003. *Understanding psychology*, Glencoe/McGraw-Hill.

Katai, Z., Juhász, K. & Adorjáni, A.K., 2008. On the role of senses in education. *Computers & Education*, 51(4), pp.1707–1717.

Kay, A., 1984. Computer software. *Scientific American*, 251(3), pp.53–59.

Kelleher, C. & Pausch, R., 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM computing surveys*, 37(2), pp.83–137.

Kernighan, B.W., Ritchie, D.M. & Ejeklint, P., 1988. *The C programming language*, Englewood Cliffs: Prentice-Hall.

Kimchi, R., Behrmann, M. & Olson, C.R. eds., 2003. *Perceptual organization in vision–behavioral and neural perspectives*, Lawrence Erlbaum Associates, Inc.

Kimura, D., 1993. Neuromotor mechanisms in human communication: Asymmetry. In New York, NY: University Press.

Kitamura, Y., Itoh, Y. & Kishino, F., 2001. Real-time 3D interaction with ActiveCube. In *CHI '01 extended abstracts on human factors in computing systems*. Seattle, Washington: ACM Press, pp. 355–356.

Knoll, M., Weis, T., Ulbrich, A. & Brändle, A., 2006. Scripting your home. In *Location-and Context-Awareness*. Springer, pp. 274–288.

Knudsen, J., 1999. *The unofficial guide to Lego Mindstorms robots*, Cambridge: O'Reilly & Associates, Inc.

Ko, A.J., Abraham, R., Beckwith, L., Blackwell, A.F., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M.B., Rothermel, G., Shaw, M. & Wiedenbeck, S., 2011. The state of the art in end-user software engineering. *ACM computing surveys*, 43(3), pp.21:1–21:44.

Koffka, K., 1935. *Principles of Gestalt psychology*, New York: Harcourt, Brace and Company.

Köhler, W., 1938. A source book of Gestalt psychology. In W. D. Ellis, ed. London: Kegan Paul, Trench, Trubner & Company, pp. 17–54.

Kosslyn, S.M., 1996. *Image and brain: The resolution of the imagery debate*, Bradford Books.

Krampen, M., Oehler, K., Posner, R., Sebeok, T.A. & Uexkull, T. von eds., 1987. *Classics of semiotics*, New York: Springer Science+Business Media.

Krippendorff, K., 1989. On the essential contexts of artifacts or on the proposition that design is making sense (of things). *Design issues*, 5(2), pp.9–39. Available at: http://0-www.jstor.org.oasis.unisa.ac.za/stable/1511512.

Kubovy, M., Holcombe, A.O. & Wagemans, J., 1998. On the lawfulness of grouping by proximity. *Cognitive psychology*, 35, pp.71–98.

Lannoch, H. & Lannoch, H.-J., 1989. Toward a semantic notion of space. *Design Issues*, 5(2), pp.40–50.

Larsen, S.E., 1994. The encyclopaedia of language and linguistics. In R. E. Asher, ed. Oxford: Pergamon Press, pp. 3821–3832.

Lee, B., 1997. *Talking heads: Language, metalanguage, and the semiotics of subjectivity*, Durham: Duje University Press.

Lee, J., Vargas, G., Tang, M. & Ishii, H., 2012. Rainbottles: Gathering raindrops of data from the cloud. In *CHI '12 extended abstracts on human factors in computing systems*. CHI EA '12. Austin, Texas, USA: ACM, pp. 1901–1906. Available at: http://doi.acm.org/10.1145/2212776.2223726.

Lee, S.-H. & Blake, R., 1999. Detection of temporal structure depends on spatial structure. *Vision research*, 39(18), pp.3033–3048. Available at: http://www.sciencedirect.com/science/article/pii/S0042698998003332.

Leymarie, F.F., 2006. Aesthetic computing and shape. In A. Bassi, M. Bauer, M. Fiedler, T. Kramp, R. van Kranenburg, S. Lange, & S. Meissner, eds. *Aesthetic computing*. The MIT Press, pp. 259–313.

Leyton, M., 2006. Aesthetic computing. In P. A. Fishwick, ed. The MIT Press, pp. 289–313.

Liszka, J.J., 1996. *A general introduction to the semeiotic of Charles Sanders Peirce*, Bloomington: Indiana University Press.

Ljungstrand, P., Björk, S. & Falk, J., 1999. The WearBoy: A platform for low-cost public wearable devices. *The third international symposium on wearable computers, 1999. Digest of Papers.*

Ljungstrand, P. & Holmquist, L.E., 1999. WebStickers: Using physical objects as WWW bookmarks. In *CHI '99 extended abstracts on human factors in computing systems*. CHI EA '99. Pittsburgh, Pennsylvania: ACM, pp. 332–333. Available at: http://doi.acm.org/10.1145/632716.632916.

Lucas, C.G., Bridgers, S., Griffiths, T.L. & Gopnik, A., 2014. When children are better (or at least more open-minded) learners than adults: Developmental differences in learning the forms of causal relationships. *Cognition*, 131(2), pp.284–299.

Lund, H.H., 2003. Intelligent artefacts. In *Proceedings of the 8 th International symposium on artificial life and robotics, ISAROB*. Citeseer, pp. I11–I14.

Lund, H.H., 2004. Modern artificial intelligence for human-robot interaction. *Proceedings of the IEEE*, 92(11), pp.1821–1838.

Marais, M.A., Smith, A.C. & Duveskog, M., 2007. TekkiKids: Technology clubs for children. In *Meraka innovate conference*. Pretoria, p. 8. Available at: http://hdl.handle.net/10204/2256.

March, S. & Smith, G., 1995. Design and natural science research on information technology. *Decision support system*, 15(4), pp.251–266.

Marco, J., Cerezo, E., Baldasarri, S., Mazzone, E. & Read, J.C., 2009. User-oriented design and tangible interaction for kindergarten children. In *IDC '09 proceedings of the 8th international conference on interaction design and children*. Como, Italy: ACM Press, pp. 190–193.

Marco, J., Cerezo, E. & Baldassarri, S., 2012. ToyVision: A toolkit for prototyping tabletop tangible games. In *Proceedings of the 4th ACM SIGCHI symposium on engineering interactive* *computing systems*. EICS ’12. Copenhagen, Denmark: ACM, pp. 71–80. Available at: http://doi.acm.org/10.1145/2305484.2305498.

Marco, R.J., 2011. *Design, implementation and evaluation of tangible design intefaces for children*. , PhD thesis, Zaragoza, Spain: Universidad de Zaragoza.

Marti, P. & Lund, H.H., 2004. Novel tangible interfaces for physical manipulation, conceptual constructions and action composition. In *Proceedings of intelligent manipulation and grasping (IMG04)*.

Martin, B. & Ringham, F., 2000. *Dictionary of semiotics*, Casstell.

Marty, R., 2015. *76 Definitions of the sign by C. S. Peirce*, Available at: http://perso.numericable.fr/robert.marty/semiotique/76defeng.htm.

Mason, J., 1946. *Principles of chess in theory and practice*, Kingsport: Kingsport Press.

Matlin, M.W., 1988. *Sensation and perception* second., Allyn and Bacon, Inc.

Mazalek, A., 2001. *Tangible interfaces for interactive point-of-view narratives*, Masters thesis, University of Toronto.

Mazalek, A., 2005. *Media tables: An extensible method for developing multi-user media interaction platforms for shared spaces*, PhD thesis, Massachusetts Institute of Technology.

Mazalek, A., Davenport, G. & Ishii, H., 2002. Tangible viewpoints: A physical approach to multimedia stories. In *MULTIMEDIA ’02 proceedings of the tenth ACM international conference on multimedia*. Juan-les-Pins, France: ACM, pp. 153–160.

Mazalek, A., Wood, A. & Ishii, H., 2001. genieBottles: An interactive narrative in bottles. In *SIGGRAPH 2001: Sketches and applications*.

McCloud, S., 1994. *Understanding comics: The invisible art*, Harper Collins Publishers.

McNerney, T.S., 1999. *Tangible Programming Bricks: An approach to making programming accessible to everyone.* Masters thesis, Massachusetts Institute of Technology.

McNerney, T.S., 2004. From turtles to Tangible Programming Bricks: explorations in physical language design. *Personal Ubiquitous Comput.*, 8(5), pp.326–337.

Merleau-Ponty, M., 1962. *Phenomenology of perception*, Routledge & Kegan Paul Ltd.

Mernik, M., Heering, J. & Sloane, A.M., 2005. When and how to develop domain-specific languages. *ACM computing surveys*, 37(4), pp.316–344. Available at: http://doi.acm.org/10.1145/1118890.1118892.

Merrell, F., 1997. *Peirce, signs, and meaning*, University of Toronto Press.

Merrell, F., 2015. Semiology meets semiotics: A case of lingering linguicentrism? Available at: http://web.ics.purdue.edu/ fmerrell/linguicentrism.htm.

Merriam-Webster, Merriam-Webster dictionary. Available at: www.merriam-webster.com, accessed 20 January 2017.

Merrill, D., Kalanithi, J. & Maes, P., 2007. Siftables: towards sensor network user interfaces. In *Proceedings of the 1st international conference on Tangible and embedded interaction*. ACM, pp. 75–78.

Mick, D.G., 1986. Consumer research and semiotics: Exploring the morphology of signs, symbols, and significance. *Journal of consumer research*, pp.196–213.

Moggridge, B., 2006. *Designing interactions*, MIT Press.

Montemayor, J., 2001. Physical programming: software you can touch. In *CHI '01: CHI '01 extended abstracts on Human factors in computing systems*. New York, NY, USA: ACM Press, pp. 81–82.

Montemayor, J., 2003. *Physical programming: Tools for kindergarten children to author physical interactive environments*. PhD thesis, Department of Computer science, University of Maryland.

Montemayor, J., Druin, A., Chipman, G., Farber, A. & Guha, M.L., 2004. Tools for children to create physical interactive storyrooms. *ACM computers in entertainment*, 2(1), pp.12–12.

Montemayor, J., Druin, A., Farber, A., Simms, S., Churaman, W. & D'Amour, A., 2002. Physical programming: Designing tools for children to create physical interactive environments. In *Proceedings of the conference on human factors in computing systems*. Minneapolis, Minnesota, USA: ACM Press, pp. 299–306.

Morgado, L., 2006. *Framework for computer programming in preschool and kindergarten*. Department of Engenharias, Universidade de Trás-os-Montes e Alto Douro. Available at: http://home.utad.pt/ leonelm/papers/tese/teseleonelmorgado.pdf.

Morgado, L., Cruz, M. & Kahn, K., 2006. Radia Perlman - A pioneer of young children computer programming. *FORMATEX*.

Morris, C., 1964. *Signification and significance: A study of the relations of signs and values*, Cambridge: The MIT Press.

Mukherjee, A., Sharma, G. & Prakash, M., 2002. Programming using Stackable Bricks.

Nadin, M., 1988. Interface design: A semiotic paradigm. *Semiotica*, 69(3-4), pp.269–302.

Nam, Y. & Kim, J., 2010. A semiotic analysis of sounds in personal computers: Toward a semiotic model of human-computer interaction. *Semiotica*, 182(1/4), pp.269–284.

Nardi, B.A., 1993. *A small matter of programming*, Cambridge, Massachusetts, USA: MIT Press.

Newell, A., Shaw, J.C. & Simon, H.A., 1958. Elements of a theory of human problem solving. *Psychological review*, 65(3), pp.151–166.

Ngo, T.D. & Lund, H.H., 2004. Modular artefacts. In *ECOOP 2004 workshop: Components-oriented approaches to context-aware computing*. Oslo, Norway.

Niehaves, B., 2007. On epistemological diversity in design science: New vistas for a design-oriented is research? In *Twenty eighth international conference on information systems*.

Nielsen, J., 2002. *Intelligent bricks*. Masters thesis, Maersk McKinney Moller institute for production technology, University of Southern Denmark, Odense.

Nielsen, J., 2008. *User-configurable modular robotics: Design and use*. PhD thesis, The Maersk Mc-Kinney Moller Institute, University of Southern Denmark.

Nielsen, J. & Lund, H.H., 2008. Modular robotics as a tool for education and entertainment. *Computers in human behavior*, 24(2), pp.234–248.

Novikov, A.M. & Novikov, D.A., 2013. *From philosophy of science to research design*, Boca Raton: CRC Press.

O'Malley, C. & Fraser, D.S., 2004. *Literature review in learning with tangible technologies*, NESTA Futurelab.

Oh, H., Deshmane, A., Li, F., Han, J.Y., Stewart, M., Tsai, M., Xu, X. & Oakley, I., 2013. The Digital Dream Lab: Tabletop puzzle blocks for exploring programmatic concepts. In *Proceedings of the 7th international conference on tangible, embedded and embodied interaction*. TEI '13. Barcelona, Spain: ACM, pp. 51–56. Available at: http://doi.acm.org/10.1145/2460625.2460633.

Ohkubo, M., Ooide, Y. & Nojima, T., 2013. An interface composed of a collection of smart hairs. In *Proceedings of the second international workshop on smart material interfaces: Another step to a material future*. SMI '13. Sydney, Australia: ACM, pp. 23–26. Available at: http://doi.acm.org/10.1145/2534688.2534692.

Oizumi, T., Mikami, T., Sasada, S. & Ubukata, S., 2007. Diorama Table. In *Goodbye privacy*. Linz, Austria: Ars electronica, p. 51. Available at: http://90.146.8.18/en/festival2007/program/project.asp?iProjectID=13981.

Olivier, M.S., 2004. *Information technology research*, Van Schaik Publishers.

Overvliet, K.E., Krampe, R.T. & Wagemans, J., 2012. Perceptual grouping in haptic search: The influence of proximity, similarity, and good continuation. *Journal of experimental psychology: Human perception and performance*, 38(4), pp.817–821.

Palmer, S.E., 1980. What makes triangles point: Local and global effects in configurations of ambiguous triangles. *Cognitive psychology*, 12(3), pp.285–305. Available at: http://www.sciencedirect.com/science/article/pii/0010028580900122.

Palmer, S.E., 1999. *Vision science: Photons to phenomenology*, MIT Press.

Palmer, S.E., 2002a. Organizing objects and scenes. In D. J. Levitin., ed. *Foundations of cognitive psychology: Core readings*. Cambridge: The MIT Press, pp. 189–212.

Palmer, S.E., 2002b. Perceptual grouping: It's later than you think. *Current directions in psychological science*, 11(3), pp.101–106.

Palmer, S.E., 2003. Visual perception of objects. In A. F. Healy, R. W. Proctor, & I. B. Weiner, eds. John Wiley & Sons.

Palmer, S.E. & Rock, I., 1994. Rethinking perceptual organization: The role of uniform connectedness. *Psychonomic bulletin & review*, 1(1), pp.29–55.

Pangaro, G., Maynes-Aminzade, D. & Ishii, H., 2002. The actuated workbench: Computer-controlled actuation in tabletop tangible interfaces. In *UIST '02 proceedings of the 15th annual ACM symposium on user interface software and technology*. Paris, France: ACM Press, pp. 181–190.

Pangaro, G.A., 2003. *The Actuated Workbench: 2D actuation in tabletop tangible interfaces*. Masters thesis, School of Architecture and Planning, Massachusetts Institute of Technology.

Papert, S., 1980. *Mindstorms: children, computers, and powerful ideas*, New York, NY, USA: Basic Books, Inc.

Patten, J., Griffith, L. & Ishii, H., 2000. A tangible interface for controlling robotic toys. In *CHI '00: CHI '00 extended abstracts on Human factors in computing systems*. The Hague, The Netherlands: ACM Press, pp. 277–278.

Patten, J., Ishii, H., Hines, J. & Pangaro, G., 2001. Sensetable: a wireless object tracking platform for tangible user interfaces. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*. Seattle, Washington, United States: ACM Press, pp. 253–260.

Patten, J., Recht, B. & Ishii, H., 2002. Audiopad: A tag-based interface for musical performance. In *NIME '02 proceedings of the 2002 conference on new interfaces for musical expression*. Dublin, Ireland: National University of Singapore, pp. 1–6.

Patten, J., Recht, B. & Ishii, H., 2006. Interaction techniques for musical performance with tabletop tangible interfaces. In *ACE '06 proceedings of the 2006 ACM SIGCHI international conference on advances in computer entertainment technology*. Hollywood, California: ACM Press, p. 27.

Patten, J.M., 2005. *Mechanical constraints as common ground between people and computers*. PhD thesis, Massachusetts Institute of Technology.

Pattis, R.E., 1995. *Karel the robot: A gentle introduction to the art of programming*, John Wiley & Sons.

Patton, M.Q., 2002. *Qualitative research & evaluation methods* third., California: Sage Publications, Inc.

Patton, S.F., 1985. Zimbabwe contemporary stone sculpture. *African arts*, 19(1), pp.78–78. Available at: http://www.jstor.org/stable/3336389.

Patwell, J.M. ed., 1992. *The American heritage dictionary of the English language* third., Jonathan P. Latimer.

Pedersen, E.R., Sokoler, T. & Nelson, L., 2000. PaperButtons: Expanding a tangible user interface. In *DIS '00 proceedings of the conference on designing interactive systems*. New York City, New York, United States: ACM Press, pp. 216–223.

Pedersen, E.W. & Hornbæk, K., 2009. mixiTUI: A tangible sequencer for electronic live performances. In *Proceedings of the 3rd international conference on tangible and embedded Interaction*. TEI '09. Cambridge, United Kingdom: ACM, pp. 223–230. Available at: http://doi.acm.org/10.1145/1517664.1517713.

Peffers, K., Rothenberger, M., Tuunanen, T. & Vaezi, R., 2012. Design science research in information systems: Advances in theory and practice. In K. Peffers, M. Rothenberger, & B. Kuechler, eds. Lecture notes in computer science. Berlin: Springer-Verlag, pp. 398–410.

Peirce, C.S., 1935. *The collected papers of Charles Sanders Peirce* C. Hartshorne, P. Weiss, & A. W. Burks, eds., Cambridge: Harvard University Press.

Perlman, R., 1974. *TORTIS: Toddler's own recursive turtle interpreter system.* Technical report, MIT Artificial Intelligence Lab.

Perlman, R., 1976. *Using computer technology to provide a creative learning environment for preschool children.* Technical report, MIT Artificial Intelligence Lab.

Philipose, M., Fishkin, K.P., Perkowitz, M., Patterson, D.J., Fox, D., Kautz, H. & Hahnel, D., 2004. Inferring activities from interactions with objects. *Pervasive computing, IEEE*, 3(4), pp.50–57.

Piper, B., Ratti, C. & Ishii, H., 2002. Illuminating Clay: A 3-D tangible interface for landscape analysis. In *Proceedings of the SIGCHI conference on human factors in computing systems*. CHI '02. Minneapolis, Minnesota, USA: ACM, pp. 355–362. Available at: http://doi.acm.org/10.1145/503376.503439.

Poupyrev, I., Nashida, T., Maruyama, S., Rekimoto, J. & Yamaji, Y., 2004. Lumen: Interactive visual and shape display for calm computing. In *ACM SIGGRAPH 2004 emerging technologies*. SIGGRAPH '04. Los Angeles, California: ACM, p. 17–. Available at: http://doi.acm.org/10.1145/1186155.1186173.

Price, S., 2008. A representation approach to conceptualizing tangible learning environments. In *Proceedings of the 2nd international conference on tangible and embedded interaction*. ACM, pp. 151–158.

Purao, S., 2002. Design research in the technology of information systems: truth or dare.

Purao, S., 2013. Truth or dare: The ontology question in design science research. *Journal of database management*, 24(3), pp.51–66.

Raffle, H., 2008. *Sculpting behavior: A tangible language for hands-on play and learning*. PhD thesis, Tangible Media Group, MIT Media Lab.

Raffle, H., Ishii, H. & Yip, L., 2007. Remix and Robo: sampling, sequencing and real-time control of a tangible robotic construction system. In *IDC '07: Proceedings of the 6th international conference on Interaction design and children*. Aalborg, Denmark: ACM Press, pp. 89–96.

Raffle, H., Parkes, A., Ishii, H. & Lifton, J., 2006. Beyond record and play: backpacks: tangible modulators for kinetic behavior. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*. Montreal, Quebec, Canada: ACM Press, pp. 681–690.

Raffle, H.S., Parkes, A.J. & Hiroshi, I., 2004. Topobo: a constructive assembly system with kinetic memory. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA: ACM Press, pp. 647–654.

Rapaport, W.J., 2012. Semiotic systems, computers, and the mind: How cognition could be computing. *International journal of signs and semiotic systems*, 2(1), pp.32–71. Available at: http://dx.doi.org/10.4018/IJSSS.2012010102.

Raymond, E.S., 2000. *A brief history of hackerdom*, Thyrsus Enterprises. Available at: http://www.immagic.com/eLibrary/ARCHIVES/GENERAL/AUTHOR_P/R000825P.pdf.

Reas, C. & Fry, B., 2007. *Processing: A programming handbook for visual designers and artists*, The MIT Press.

Reas, C. & Fry, B., 2010. *Getting started with Processing*, O'Reilly Media, Inc.

Reitsma, E.S., 2011. *StoryBeads: For the preservation of indigenous stories*, Technical University of Eindhoven.

Reitsma, L., Smith, A.C. & Hoven, E. van den, 2013. StoryBeads: Preserving indigenous knowledge through tangible interaction design. In *International conference on culture and computing*. IEEE, pp. 79–85.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. & Kafai, Y., 2009. Scratch: Programming for all. *Communications of the ACM*, 52(11), pp.60–67.

Rock, I. & Palmer, S.E., 1990. The legacy of Gestalt psychology. *Scientific American*, 263, pp.84–90.

Roode, D., 1993. Implications for teaching of a process-based research framework for information systems. In L. Smith, ed. *Proceedings of the international academy for information management conference.* Orlando, Florida.

Rosner, D. & Ryokai, K., 2008. Weaving memories into handcrafted artifacts with Spyn. In *CHI '08 extended abstracts on human factors in computing systems*. CHI '08. Florence, Italy: ACM, pp. 2331–2336. Available at: http://doi.acm.org/10.1145/1358628.1358679.

Rosner, D.K. & Ryokai, K., 2009. Reflections on craft: Probing the creative process of everyday knitters. In *Proceedings of the seventh ACM conference on creativity and cognition*. Berkeley, California, USA: ACM, pp. 195–204. Available at: http://doi.acm.org/10.1145/1640233.1640264.

Rosner, D.K. & Ryokai, K., 2010. Spyn: Augmenting the creative and communicative potential of craft. In *CHI 2010 cooking, classrooms, and craft*. ACM.

Rossi, M. & Sein, M.K., 2003. Design research workshop: A proactive research approach. Available at: https://people.aalto.fi/index.html?profilepage=isfor#!matti_rossi, Accessed 20 January 2017.

Rossum, G. van, 2012. *The Python language reference* F. L. Drake, ed., Python Software Foundation.

Roy, P.V. & Haridi, S., 2004. *Concepts, techniques, and models of computer programming*, Cambridge, Massachusetts: The MIT Press.

Sabre, R.M., 2012. Peirce's ten trichotomies: Metaphor, hypothesis, and decision. *Semiotica*, pp.23–39.

Sáenz-Ludlow, A., 2007. Signs and the process of interpretation: Sign as an object and as a process. *Studies in philosophy and education*, 26(3), pp.205–223. Available at: http://dx.doi.org/10.1007/s11217-007-9028-4.

Sajaniemi, J., 2008. Psychology of programming: Looking into programmers' heads. *Human Technology*, 4(1), pp.4–8.

Sanders, E.B.N., 2000. Generative tools for co-designing. In *Collaborative design proceedings of CoDesigning 2000*. London: Springer, pp. 1–12.

Sarbo, J.J. & Farkas, J., 2013. Towards meaningful information processing: A unifying representation for Peirce's sign types. *Signs: International journal of semiotics*, 7, pp.1–41. Available at: http://vip.iva.dk/sis/index.php?journal=signs.

Saussure, F. de, 1916. *Cours de linguistique generale* C. Bailly & A. Sechehaye, eds., Lausanne: Libraire Payot & Cie.

Saussure, F. de, 1959. *Course in general linguistics* C. Bally & A. Sechehaye, eds., New York: Phylisophical Library, Inc.

Saussure, F. de, 2011. *Course in general linguistics* P. Meisel & H. Saussy, eds., New York: Columbia University Press.

Schapiro, M., 1998. The still life as a personal object: A note on Heidegger and van Gogh. In D. Preziosi, ed. *The art of art history: A critical anthology*. New York: Oxford University Press, pp. 427–431. Available at: http://dx.doi.org/10.1007/978-3-662-40265-8_14.

Schiettecatte, B. & Vanderdonckt, J., 2008. AudioCubes: A distributed cube tangible interface based on interaction range for sound design. In *Proceedings of the 2nd international conference on tangible and embedded Interaction*. TEI '08. Bonn, Germany: ACM, pp. 3–10. Available at: http://doi.acm.org/10.1145/1347390.1347394.

Schon, D.A., 1983. *The reflective practitioner: How professionals think in action*, Basic Books.

Schwandt, T.A., 2007. *The Sage dictionary of qualitative inquiry* third., California: Sage Publications, Inc.

Schweikardt, E. & Gross, M.D., 2006. roBlocks: a robotic construction kit for mathematics and science education. In *ICMI '06: Proceedings of the 8th international conference on Multimodal interfaces*. Banff, Alberta, Canada: ACM Press, pp. 72–75.

Schweikardt, E. & Gross, M.D., 2007. A brief survey of distributed computational toys. In *DIGITEL'07. The First IEEE international workshop on digital game and intelligent toy enhanced learning, 2007*. pp. 57–64.

ScienceUnlimited, ScienceUnlimited. Available at: http://www.scienceunlimited.co.za, accessed 28 May 2014.

Scifest, Scifest Africa. Available at: http://www.scifest.org.za, accessed 28 May 2014.

Sears, A. & Jacko, J.A. eds., 2009. *Human-computer interaction. Design issues, solutions, and applications*, London: CRC Press.

Sebeok, T.A., 2001. *Signs: An introduction to semiotics* second., Toronto: University of Toron to Press.

Sekuler, A.B. & Bennett, P.J., 2001. Generalized common fate: Grouping by common luminance changes. *Psychological Science*, 12(6), pp.437–444.

Shaer, O., Leland, N., Calvillo-Gamez, E.H. & Jacob, R.J.K., 2004. The TAC paradigm: Specifying tangible user interfaces. In *Personal and ubiquitous computing*. Springer-Verlag, pp. 359–369.

Sharlin, E., Itoh, Y., Watson, B., Kitamura, Y., Sutphen, S. & Liu, L., 2002. Cognitive cubes: A tangible user interface for cognitive assessment. In *CHI '02 proceedings of the SIGCHI conference on human factors in computing systems*. Minneapolis, Minnesota, USA: ACM Press, pp. 347–354.

Shepard, R.N. & Levitin, D.J., 2002. Foundations of cognitive psychology: Core readings. In D. J. Levitin, ed. The MIT Press, pp. 503–514.

Sheriff, J.K., 1989. *The fate of meaning: Charles Peirce, structuralism, and literature*, Princeton, N.J.: Princeton University Press.

Sherman, L., Druin, A., Montemayor, J., Farber, A., Platner, M., Simms, S., Porteous, J., Alborzi, H., Best, J., Hammer, J., Kruskal, A., Matthews, J., Rhodes, E., Cosans, C. & Lal, A., 2001. StoryKit: tools for children to build room-sized interactive experiences. In *CHI '01 extended abstracts on Human factors in computing systems*. Seattle, Washington: ACM Press, pp. 197–198.

Simmel, G., 2004. *The philosophy of money*, New York: Routledge.

Simon, H.A., 1996. *The sciences of the artificial* third., London: MIT Press.

Sipitakiat, A. & Nusen, N., 2012. Robo-Blocks: Designing debugging abilities in a tangible programming system for early primary school children. In *Interaction design and children*. ACM Press, pp. 98–105.

Smith, A.C., 2006. Tangible cubes as programming objects. In *Proceedings of the 16th international conference on artificial reality and telexistence–workshops (ICAT'06)*. Hangzhou, China: IEEE Conference Publications, pp. 157–161. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4089231.

Smith, A.C., 2007a. GameBlocks: An entry point to ICT for pre-school children. In *Meraka innovate conference*. CSIR Convention Centre, Pretoria South Africa. Available at: http://hdl.handle.net/10204/1778.

Smith, A.C., 2007b. Using magnets in physical blocks that behave as programming objects. In *TEI '07 proceedings of the 1st international conference on tangible and embedded interaction*. New York, NY, USA: ACM Press, pp. 147–150.

Smith, A.C., 2008a. A low-cost, low-energy tangible programming system for computer illiterates in developing regions. In *4th International workshop on technology for innovation and education in developing countries (TEDC)*. Available at: http://playpen.meraka.csir.co.za/ acdc/education/TEDC 2008 Proceedings - Technology for Innovation and Education in Developing Countries: 978-0-620-43087-6/Smith_08.pdf.

Smith, A.C., 2008b. Handcrafted physical syntax elements for illetterate children: Initial concepts. In *Proceedings of the 7th international conference on interaction design and children*. IDC '08.

Chicago, Illinois: ACM Press, pp. 157–160. Available at: http://doi.acm.org/10.1145/1463689.1463745.

Smith, A.C., 2008c. Programming without a computer: Introducing young children to computer programming. Available at: http://www.conf2008.school.za/PresentationData/118/programming without a computer ac smith innovate 2008 v1’0.pdf.

Smith, A.C., 2009a. Hand-crafted programming objects and visual perception. In *IST-Africa 2009 conference proceedings*. IIMC International Information Management Corporation.

Smith, A.C., 2009b. Leisure robotics: An African child’s gateway to programming. In *3rd Robotics & mechatronics symposium (ROBMECH 2009)*. Pretoria, South Africa. Available at: http://hdl.handle.net/10204/5360.

Smith, A.C., 2009c. Simple tangible language elements for young children. In *Proceedings of the 8th international conference on interaction design and children*. IDC ’09. Como, Italy: ACM, pp. 288–289. Available at: http://doi.acm.org/10.1145/1551788.1551860.

Smith, A.C., 2009d. Symbols for children’s tangible programming cubes: An explorative study. In *Proceedings of the 2009 annual conference of the Southern African computer lecturers’ association*. SACLA ’09. Eastern Cape, South Africa: ACM, pp. 105–109. Available at: http://doi.acm.org/10.1145/1562741.1562755.

Smith, A.C., 2009e. Visual perception skills testing: Preliminary results. In *Proceedings of the 3rd international conference on tangible and embedded Interaction*. TEI ’09. Cambridge, United Kingdom: ACM, pp. 207–208. Available at: http://doi.acm.org/10.1145/1517664.1517709.

Smith, A.C., 2010a. Dialando: Tangible programming for the novice with Scratch, Processing and Arduino. In *6th International workshop on technology for innovation and education in developing countries (TEDC)*. Available at: http://hdl.handle.net/10204/4048.

Smith, A.C., 2010b. Tangible interfaces for tangible robots. In E. Hall, ed. *Advances in robot manipulators*. Croatia: InTech, pp. 607–624. Available at: http://hdl.handle.net/10204/4351.

Smith, A.C., 2014a. Cluster-based tangible programming. In *Fourth international conference on digital information and communication technology and it’s applications (DICTAP)*. IEEE, pp. 405–410. Available at: http://ieeexplore.ieee.org/.

Smith, A.C., 2014b. Rock Garden Programming: programming in the physical world. In *Digital Information and Communication Technology and it’s Applications (DICTAP), 2014 Fourth International Conference on*. IEEE, pp. 430–434. Available at: http://ieeexplore.ieee.org/.

Smith, A.C., Foko, T. & Van Deventer, A., 2008. Quantifying the visual perception skills of pre-school testees using a novel tangible electronic test instrument. In *Science real and relevant: 2nd CSIR biennial conference*. CSIR international convention centre, Pretoria, p. 11. Available at: http://hdl.handle.net/10204/2554.

Smith, A.C. & Gelderblom, J.H., 2013a. The building is the program. In *Peripheral interaction: Embedding HCI in everyday life, A volume in the workshop proceedings series of the INTERACT 2013 conference*. Workshop at INTERACT 2013. Available at: http://www.peripheralinteraction.com/interact/paper/Workshop_PI_Smith.pdf.

Smith, A.C. & Gelderblom, J.H., 2013b. Towards a tangible web: Using physical objects to access and manipulate the Internet of Things. In *Proceedings of the 15th annual conference on world wide web applications*. Available at: http://hdl.handle.net/10204/7367.

Smith, A.C. & Gelderblom, J.H., 2016. End user programming with personally meaningful objects. In L. Church, ed. *Proceedings of the 27th annual workshop of the psychology of programming interest group - PPIG 2016*. St. Catharine's College, University of Cambridge, UK. Available at: https://drive.google.com/file/d/0B-7G3GOHucdiMTNjdEtyMC1qWjQ/view.

Smith, A.C. & Kotzé, P., 2010. Indigenous African artefacts: Can they serve as tangible programming objects? In *IST-Africa 2010 conference proceedings*. IEEE Conference Publications. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5753043.

Smith, A.C., Kotzé, P. & Gelderblom, H., 2011a. General design methodology applied to the research domain of physical programming for computer illiterates. In *Design, development & research conference*. Faculty of Informatics and Design, Cape Peninsula University of Technology. Available at: http://hdl.handle.net/10204/5423.

Smith, A.C., Reitsma, L., Hoven, E. van den, Kotzé, P. & Coetzee, L., 2011b. Towards preserving indigenous oral stories using tangible objects. In *Second international conference on culture and computing*. Kyoto, Japan: IEEE Conference Publications, pp. 86–91. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6103215.

Smith, A.C., Springhorn, H., Mulligan, S.B., Weber, I. & Norris, J., 2011c. tactusLogic: Programming using physical objects. In *IST-Africa 2011 conference proceedings*. IEEE Conference Publications. Available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6107337.

Smith, E.W., 1952. African symbolism. *The Journal of the Royal Anthropological Institute of Great Britain and Ireland*, 82(1), pp.13–37. Available at: http://www.jstor.org/stable/2844037.

Sonnenberg, C. & Brocke, J. vom, 2012. Design science research in information systems: Advances in theory and practice. In K. Peffers, M. Rothenberger, T. Tuunanen, R. Vaezi, K. Peffers, M. Rothenberger, & B. Kuechler, eds. Lecture notes in computer science. Berlin: Springer-Verlag, pp. 381–397.

Souza, C.S. de, 2005. *The semiotic engineering of human-computer interaction*, MIT Press.

Steinman, R.M., Pizlo, Z. & Pizlo, F.J., 2000. Phi is not beta, and why Wertheimer's discovery launched the Gestalt revolution. *Vision research*, 40(17), pp.2257–2264.

Stoakley, R., Conway, M.J. & Pausch, R., 1995. Virtual reality on a WIM: interactive worlds in miniature. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press/Addison-Wesley Publishing Co., pp. 265–272.

Streitz, N.A., Geissler, J., Holmer, T., Konomi, S., Müller-Tomfelde, C., Reischl, W., Rexroth, P., Seitz, P. & Steinmetz, R., 1999. i-LAND: an interactive landscape for creativity and innovation. In *CHI '99 proceedings of the SIGCHI conference on human factors in computing systems*. Pittsburgh, Pennsylvania, United States: ACM, pp. 120–127.

Suzuki, H. & Kato, H., 1994. AlgoBlock: A tangible programming language for collaborative learning. *Ed-Media'94*.

Suzuki, H. & Kato, H., 1995a. An educational tool for collaborative learning: AlgoBlock. *Cognitive Studies*, 2(1), pp.1_36–1_47.

Suzuki, H. & Kato, H., 1995b. Interaction-level support for collaborative learning: AlgoBlock - an open programming language. In *CSCL ’95: The first international conference on Computer support for collaborative learning*. Indiana Univ., Bloomington, Indiana, United States: Lawrence Erlbaum Associates, Inc., pp. 349–355.

Syropoulos, A., Tsolomitis, A. & Sofroniou., N., 2003. *Digital typography using LaTeX*, New York: Springer-Verlag.

Takahashi, K., 2007a. Diorama table. In *ACM SIGGRAPH 2007 art gallery*. SIGGRAPH ’07. San Diego, California: ACM, p. 221–. Available at: http://doi.acm.org/10.1145/1280120.1280178.

Takahashi, K., 2007b. Diorama table. In *Ars Electronica*. Ars Electronica, pp. 428–429. Available at: http://90.146.8.18/de/archiv_files/20071/081_FE_2007_Keiko_Takahashi.pdf.

Takahashi, K., 2007c. Diorama table video. Available at: http://www.th.jec.ac.jp/ keiko/vimeo/dioramaT_vimeo.html.

Takahashi, K. & Sasada, S., 2005. Diorama table. In *Proceedings of the 13th annual ACM international conference on multimedia*. MULTIMEDIA ’05. Hilton, Singapore: ACM, pp. 1077–1078.

Tanaka-Ishii, K., 2010. *The semiotics of programming*, Cambridge University Press.

Tarkan, G. S. Sazawal V. Druin A. Foss E. Golub E. Hatley L. Khatri T. Massey S. Walsh G. Torres, 2009. *Designing a Novice Programming Environment with Children.Technical Report 2009-03*, HCIL.

Terry, M., 2001. Task blocks: Tangible interfaces for creative exploration. In *CHI ’01 extended abstracts on human factors in computing systems*. Seattle, Washington: ACM Press, pp. 463–464.

Tonder, G.J.V. & Lyons, M.B.J., 2005. Visual perception in Japanese rock garden design. *Axiomathes*, 15, pp.353–371.

Touretzky, D.S., 1984. *LISP: A gentle introduction to symbolic computation*, New York: Harper & Row.

Tseng, T., Bryant, C. & Blikstein, P., 2011. Mechanix: An interactive display for exploring engineering design through a tangible interface. In *Proceedings of the fifth international conference on tangible, embedded, and embodied interaction*. TEI ’11. Funchal, Portugal: ACM, pp. 265–266. Available at: http://doi.acm.org/10.1145/1935701.1935757.

Tyler, C.W. ed., 2002. *Human symmetry perception and its computational analysis*, New Jersey: Lawrence Erlbaum Associates, Inc.

Ullmer, B., Ishii, H. & Glas, D., 1998. mediaBlocks: Physical containers, transports, and controls for online media. In *SIGGRAPH ’98 proceedings of the 25th annual conference on computer graphics and interactive techniques*. New York, NY, USA: ACM Press, pp. 379–386.

Ullmer, B.A., 1997. *Models and mechanisms for tangible user interfaces*. Massachusetts Institute of Technology. Available at: citeseer.ist.psu.edu/ullmer97model.html.

Ullmer, B.A., 2002. *Tangible interfaces for manipulating aggregates of digital information*. PhD thesis, Massachusetts Institute of Technology. Dept. of Architecture. Program in Media Arts and Sciences.

Ullmer, B.A. & Ishii, H., 2000. Emerging frameworks for tangible user interfaces. *IBM Systems Journal*, 39(3-4), pp.915–931.

Underkoffler, J.S., 1999a. Luminous Room ChessBottle 1999. Available at: http://vimeo.com/48602062, Accessed 19 January 2017.

Underkoffler, J.S., 1999b. *The I/O Bulb and the Luminous Room*. MIT School of Architecture and Planning.

Underkoffler, J.S. & Ishii, H., 1998. Illuminating light: An optical design tool with a luminous-tangible interface. In *CHI '98 proceedings of the SIGCHI conference on human factors in computing systems*. Los Angeles, California, United States: ACM Press/Addison-Wesley Publishing Co., pp. 542–549.

Underkoffler, J.S. & Ishii, H., 1999. Urp: A luminous-tangible workbench for urban planning and design. In *CHI '99 proceedings of the SIGCHI conference on human factors in computing systems*. Pittsburgh, Pennsylvania, United States: ACM Press, pp. 386–393.

Underkoffler, J.S., Ullmer, B. & Ishii, H., 1999. Emancipated pixels: Real-world graphics in the luminous room. In *SIGGRAPH '99 proceedings of the 26th annual conference on computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., pp. 385–392.

UNISA, 2007. *Policy on research ethics*, Pretoria: UNISA.

Vaishnavi, V. & Kuechler, W. eds., 2013. Design Science research in information systems. Available at: http://www.desrist.org/design-research-in-information-systems/.

Vaishnavi, V.K. & Kuechler, W., 2008. *Design science research methods and patterns: Innovating information and communication technology*, Auerbach Publications.

Vaishnavi, V.K. & Kuechler, W., 2015. *Design science research methods and patterns: Innovating information and communication technology*, CRC Press.

Venable, J., Pries-Heje, J. & Baskerville, R., 2016. FEDS: A framework for evaluation in design science research. *European journal of information systems*. Available at: http://espace.library.curtin.edu.au/R?func=dbin-jump-full&local_base=gen01-era02&object_id=203558.

Verstegen, I., 2005. *Arnheim, Gestalt and art: A psychological theory*, Springer.

Wagemans, J., Elder, J.H., Kubovy, M., Palmer, S.E., Peterson, M.A., Singh, M. & Heydt, R. von der, 2012. A century of Gestalt psychology in visual perception: I. Perceptual grouping and figure-ground organization. *Psychological bulletin*, 138(6), pp.1172–1217.

Wang, D., Zhang, C. & Wang, H., 2011. T-Maze: A tangible programming tool for children. In *IDC2011*. ACM Press.

Want, R., Fishkin, K.P., Gujar, A. & Harrison, B.L., 1999. Bridging physical and virtual worlds with electronic tags. In *CHI '99 proceedings of the SIGCHI conference on human factors in computing systems*. Pittsburgh, Pennsylvania, United States: ACM Press, pp. 370–377.

Watanabe, R., Itoh, Y., Asai, M., Kitamura, Y., Kishino, F. & Kikuchi, H., 2004a. The soul of ActiveCube: Implementing a flexible, multimodal, three-dimensional spatial tangible interface. *ACM computers in entertainment*, 2(4), pp.1–13.

Watanabe, R., Itoh, Y., Asai, M., Kitamura, Y., Kishino, F. & Kikuchi, H., 2004b. The soul of ActiveCube: Implementing a flexible, multimodal, three-dimensional spatial tangible interface. In *ACE '04 proceedings of the 2004 ACM SIGCHI international conference on advances in computer entertainment technology*. Singapore: ACM, pp. 173–180.

Watanabe, R., Itoh, Y., Kawai, M., Kitamura, Y., Kishino, F. & Kikuchi, H., 2004c. Implementation of ActiveCube as an intuitive 3D computer interface. In *Proceedings of 4th international symposium on smart-graphics*. LNCS. Heidelberg: Springer-Verlag, pp. 43–53.

Weinberg, G.M., 1998. *The psychology of computer programming*, New York: Van Nostrand Reinhold.

Weiten, W., 2011. *Psychology: Themes and variations, briefer version* eighth., Belmont: Wadsworth.

Welman, J. & Kruger, S., 2001. *Research methodology for the business and admonistrative sciences* second., Oxford: Oxford University Press Southern Africa.

Wertheimer, M., 1912. Experimentelle studien über das sehen von bewegung. *Zeitschrift für Psychologie*, 61, pp.161–265.

Wertheimer, M., 1938. A source book of Gestalt psychology. In W. Ellis, ed. London: Routledge & Kegan Paul, pp. 71–88. Available at: http://psychclassics.yorku.ca/Wertheimer/Forms/forms.htm.

Wootton, C., 2001. *JavaScript programmer's reference*, Wrox Press.

Wyeth, P., 2008. How young children learn to program with sensor, action, and logic blocks. *Journal of the learning sciences: a journal of ideas and their applications*, 17(4), pp.517–550.

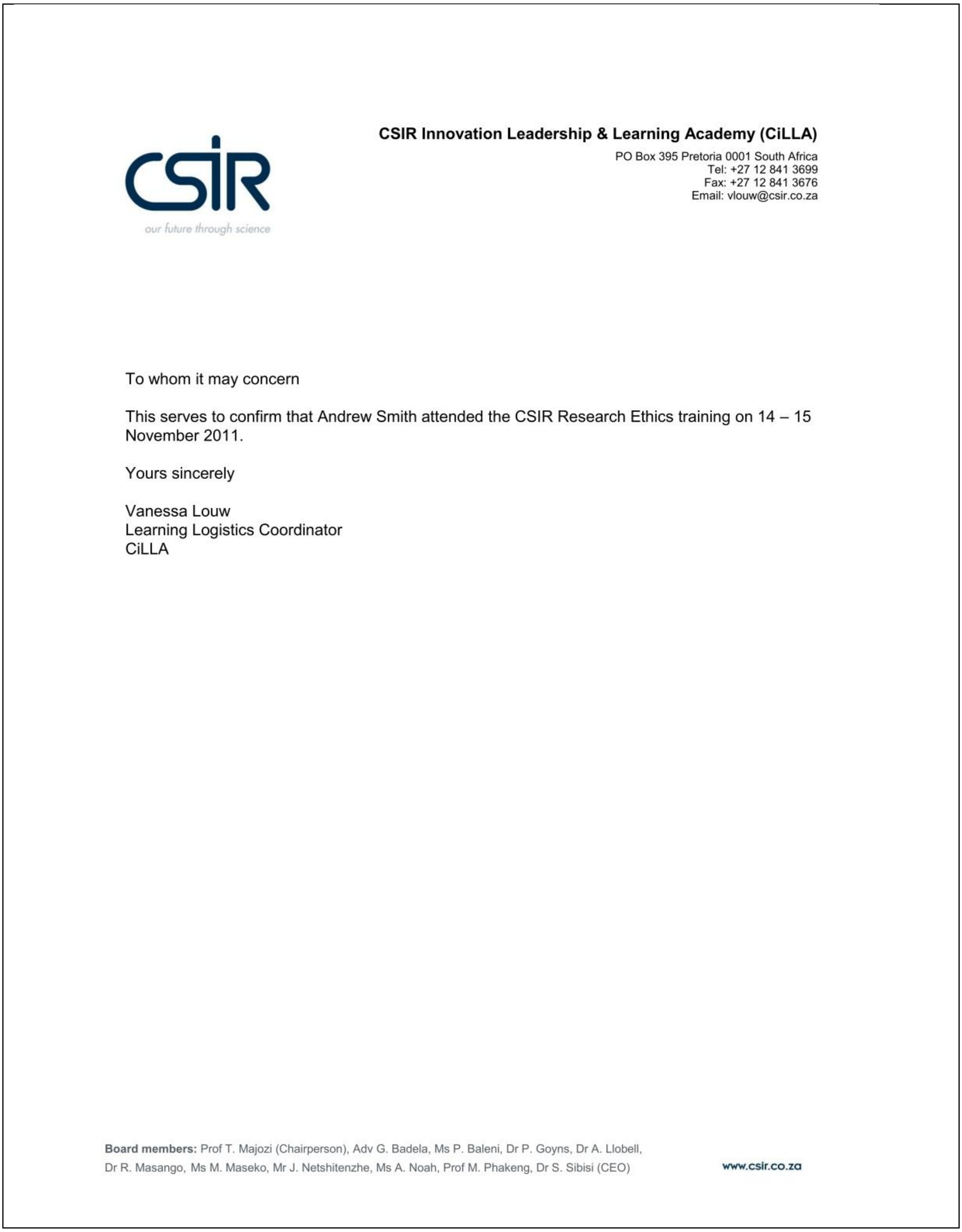
Wyeth, P. & Purchase, H.C., 2002. Tangible programming elements for young children. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*. Minneapolis, Minnesota, USA: ACM Press, pp. 774–775.

Wyeth, P. & Wyeth, G., 2001. Electronic Blocks: tangible programming elements for preschoolers. In *Proceedings of the eighth IFIP TC13 conference on Human-Computer Interaction (INTERACT 2001)*. IOS Press, pp. 496–503.

Young, H.D., Freedman, R.A. & Ford, L., 2007. *University Physics*, Addison Wesley.

Zuckerman, O., 2004. *System blocks: Learning about systems concepts through hands-on modeling and simulation*. Masters thesis, MIT Program in Media Arts & Sciences, School of Architecture & Planning.

APPENDIX A.. Research ethics training



CSIR Innovation Leadership & Learning Academy (CiLLA)

PO Box 395 Pretoria 0001 South Africa
Tel: +27 12 841 3699
Fax: +27 12 841 3676
Email: vlouw@csir.co.za

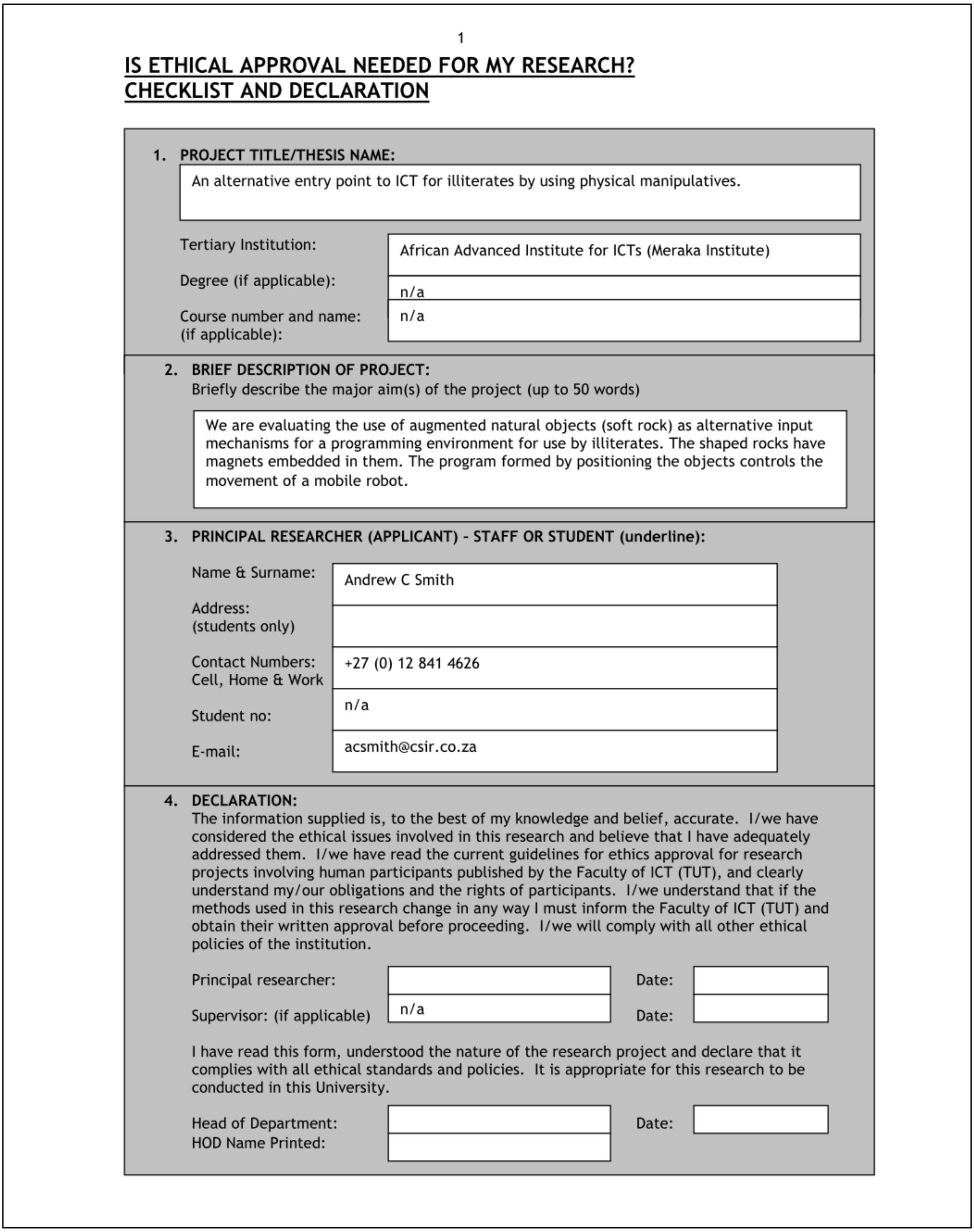
To whom it may concern

This serves to confirm that Andrew Smith attended the CSIR Research Ethics training on 14 – 15 November 2011.

Yours sincerely

Vanessa Louw
Learning Logistics Coordinator
CiLLA

APPENDIX B .... Second iteration ethical clearance form: SciFest Africa 2008



1

IS ETHICAL APPROVAL NEEDED FOR MY RESEARCH? CHECKLIST AND DECLARATION

1. PROJECT TITLE/THESIS NAME:

An alternative entry point to ICT for illiterates by using physical manipulatives.

Tertiary Institution: African Advanced Institute for ICTs (Meraka Institute)

Degree (if applicable): n/a

Course number and name: (if applicable): n/a

2. BRIEF DESCRIPTION OF PROJECT:
Briefly describe the major aim(s) of the project (up to 50 words)

We are evaluating the use of augmented natural objects (soft rock) as alternative input mechanisms for a programming environment for use by illiterates. The shaped rocks have magnets embedded in them. The program formed by positioning the objects controls the movement of a mobile robot.

3. PRINCIPAL RESEARCHER (APPLICANT) - STAFF OR STUDENT (underline):

Name & Surname: Andrew C Smith

Address:
(students only)

Contact Numbers: +27 (0) 12 841 4626
Cell, Home & Work

Student no: n/a

E-mail: acsmith@csir.co.za

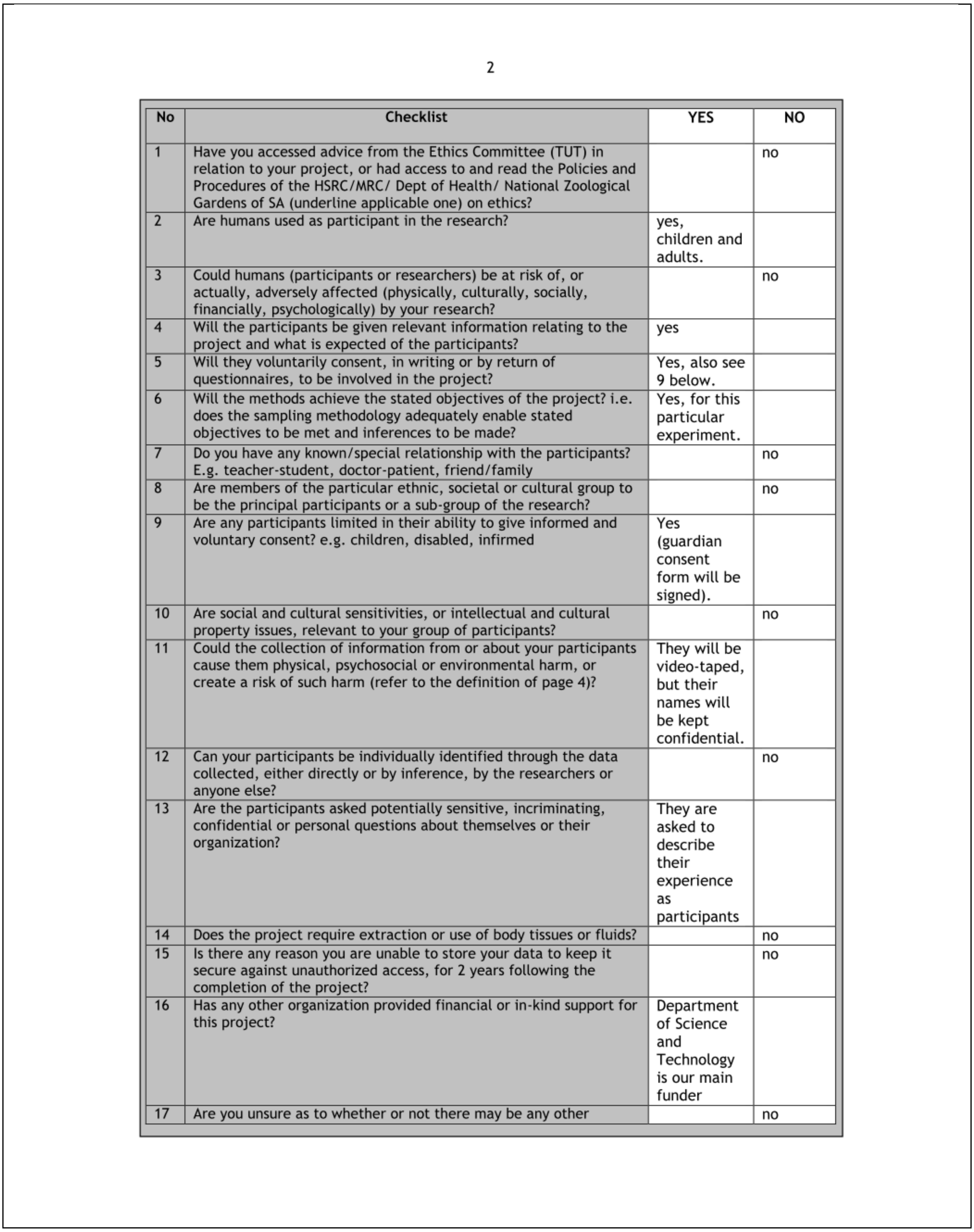
4. DECLARATION:
The information supplied is, to the best of my knowledge and belief, accurate. I/we have considered the ethical issues involved in this research and believe that I have adequately addressed them. I/we have read the current guidelines for ethics approval for research projects involving human participants published by the Faculty of ICT (TUT), and clearly understand my/our obligations and the rights of participants. I/we understand that if the methods used in this research change in any way I must inform the Faculty of ICT (TUT) and obtain their written approval before proceeding. I/we will comply with all other ethical policies of the institution.

Principal researcher: Date:

Supervisor: (if applicable) n/a Date:

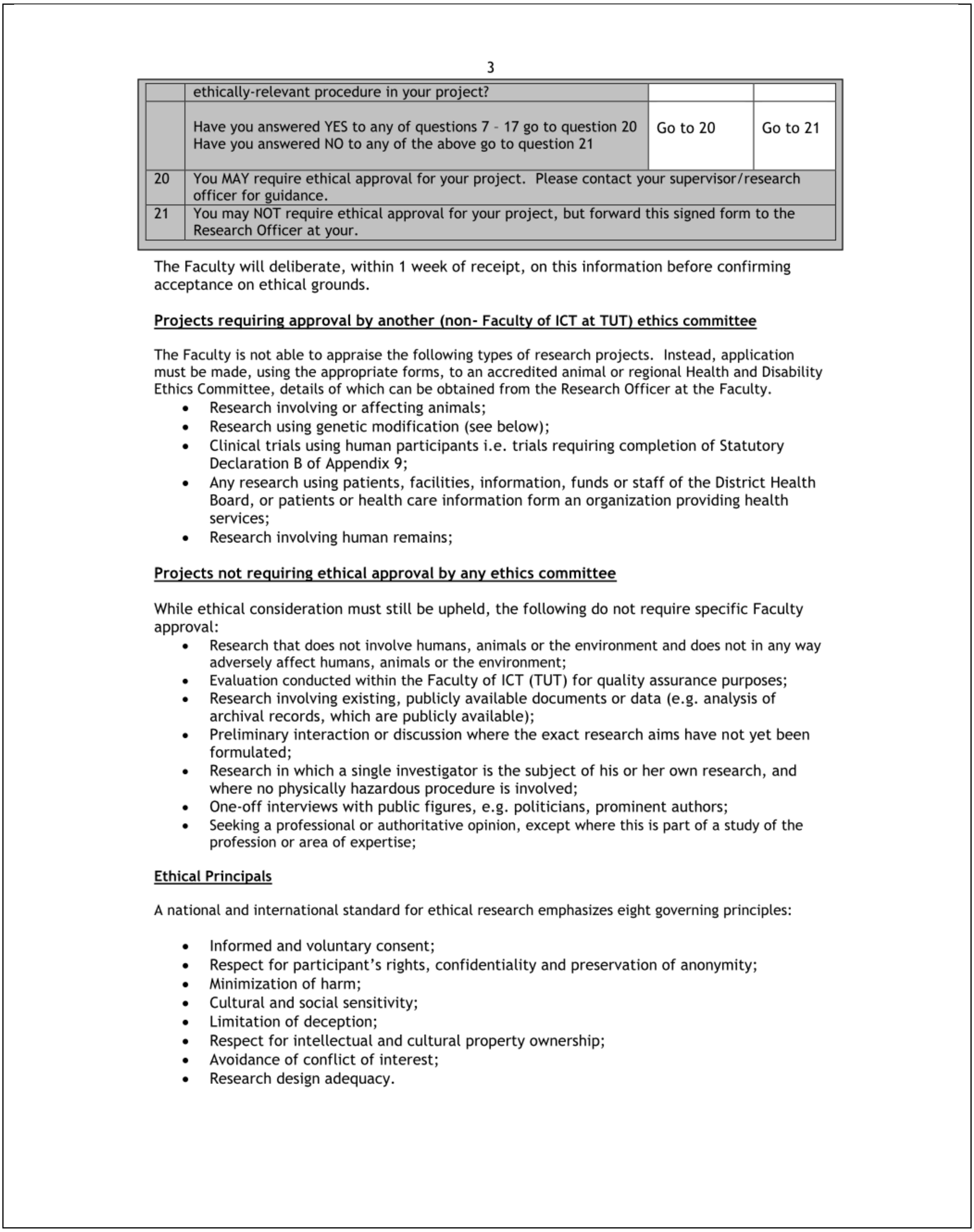
I have read this form, understood the nature of the research project and declare that it complies with all ethical standards and policies. It is appropriate for this research to be conducted in this University.

Head of Department: Date:
HOD Name Printed:



2

No	Checklist	YES	NO
1	Have you accessed advice from the Ethics Committee (TUT) in relation to your project, or had access to and read the Policies and Procedures of the HSRC/MRC/ Dept of Health/ National Zoological Gardens of SA (underline applicable one) on ethics?		no
2	Are humans used as participant in the research?	yes, children and adults.	
3	Could humans (participants or researchers) be at risk of, or actually, adversely affected (physically, culturally, socially, financially, psychologically) by your research?		no
4	Will the participants be given relevant information relating to the project and what is expected of the participants?	yes	
5	Will they voluntarily consent, in writing or by return of questionnaires, to be involved in the project?	Yes, also see 9 below.	
6	Will the methods achieve the stated objectives of the project? i.e. does the sampling methodology adequately enable stated objectives to be met and inferences to be made?	Yes, for this particular experiment.	
7	Do you have any known/special relationship with the participants? E.g. teacher-student, doctor-patient, friend/family		no
8	Are members of the particular ethnic, societal or cultural group to be the principal participants or a sub-group of the research?		no
9	Are any participants limited in their ability to give informed and voluntary consent? e.g. children, disabled, infirmed	Yes (guardian consent form will be signed).	
10	Are social and cultural sensitivities, or intellectual and cultural property issues, relevant to your group of participants?		no
11	Could the collection of information from or about your participants cause them physical, psychosocial or environmental harm, or create a risk of such harm (refer to the definition of page 4)?	They will be video-taped, but their names will be kept confidential.	
12	Can your participants be individually identified through the data collected, either directly or by inference, by the researchers or anyone else?		no
13	Are the participants asked potentially sensitive, incriminating, confidential or personal questions about themselves or their organization?	They are asked to describe their experience as participants	
14	Does the project require extraction or use of body tissues or fluids?		no
15	Is there any reason you are unable to store your data to keep it secure against unauthorized access, for 2 years following the completion of the project?		no
16	Has any other organization provided financial or in-kind support for this project?	Department of Science and Technology is our main funder	
17	Are you unsure as to whether or not there may be any other		no



	ethically-relevant procedure in your project?		
	Have you answered YES to any of questions 7 – 17 go to question 20 Have you answered NO to any of the above go to question 21	Go to 20	Go to 21
20	You MAY require ethical approval for your project. Please contact your supervisor/research officer for guidance.		
21	You may NOT require ethical approval for your project, but forward this signed form to the Research Officer at your.		

The Faculty will deliberate, within 1 week of receipt, on this information before confirming acceptance on ethical grounds.

Projects requiring approval by another (non- Faculty of ICT at TUT) ethics committee

The Faculty is not able to appraise the following types of research projects. Instead, application must be made, using the appropriate forms, to an accredited animal or regional Health and Disability Ethics Committee, details of which can be obtained from the Research Officer at the Faculty.

- Research involving or affecting animals;
- Research using genetic modification (see below);
- Clinical trials using human participants i.e. trials requiring completion of Statutory Declaration B of Appendix 9;
- Any research using patients, facilities, information, funds or staff of the District Health Board, or patients or health care information form an organization providing health services;
- Research involving human remains;

Projects not requiring ethical approval by any ethics committee

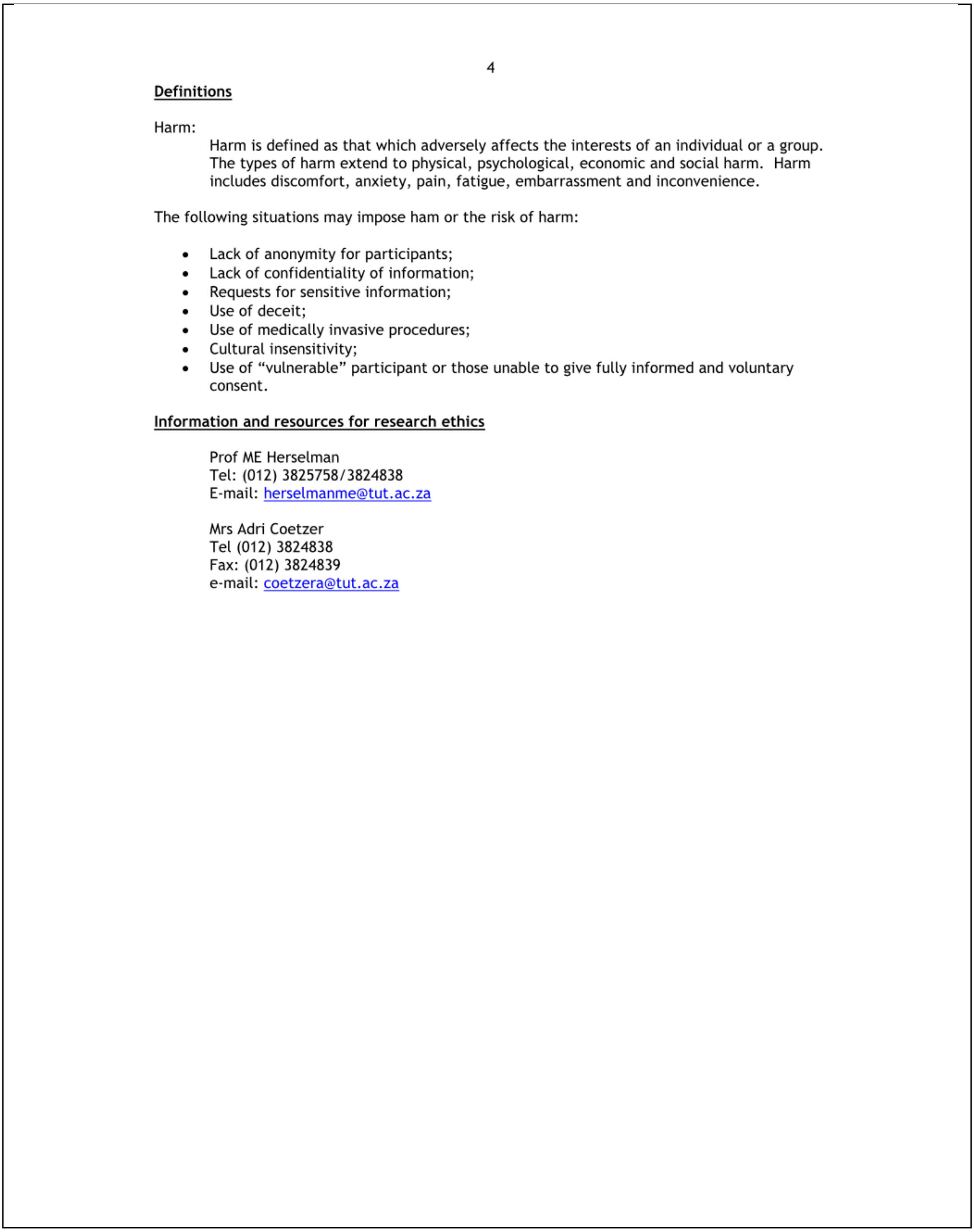
While ethical consideration must still be upheld, the following do not require specific Faculty approval:

- Research that does not involve humans, animals or the environment and does not in any way adversely affect humans, animals or the environment;
- Evaluation conducted within the Faculty of ICT (TUT) for quality assurance purposes;
- Research involving existing, publicly available documents or data (e.g. analysis of archival records, which are publicly available);
- Preliminary interaction or discussion where the exact research aims have not yet been formulated;
- Research in which a single investigator is the subject of his or her own research, and where no physically hazardous procedure is involved;
- One-off interviews with public figures, e.g. politicians, prominent authors;
- Seeking a professional or authoritative opinion, except where this is part of a study of the profession or area of expertise;

Ethical Principals

A national and international standard for ethical research emphasizes eight governing principles:

- Informed and voluntary consent;
- Respect for participant’s rights, confidentiality and preservation of anonymity;
- Minimization of harm;
- Cultural and social sensitivity;
- Limitation of deception;
- Respect for intellectual and cultural property ownership;
- Avoidance of conflict of interest;
- Research design adequacy.



Definitions

Harm:

Harm is defined as that which adversely affects the interests of an individual or a group. The types of harm extend to physical, psychological, economic and social harm. Harm includes discomfort, anxiety, pain, fatigue, embarrassment and inconvenience.

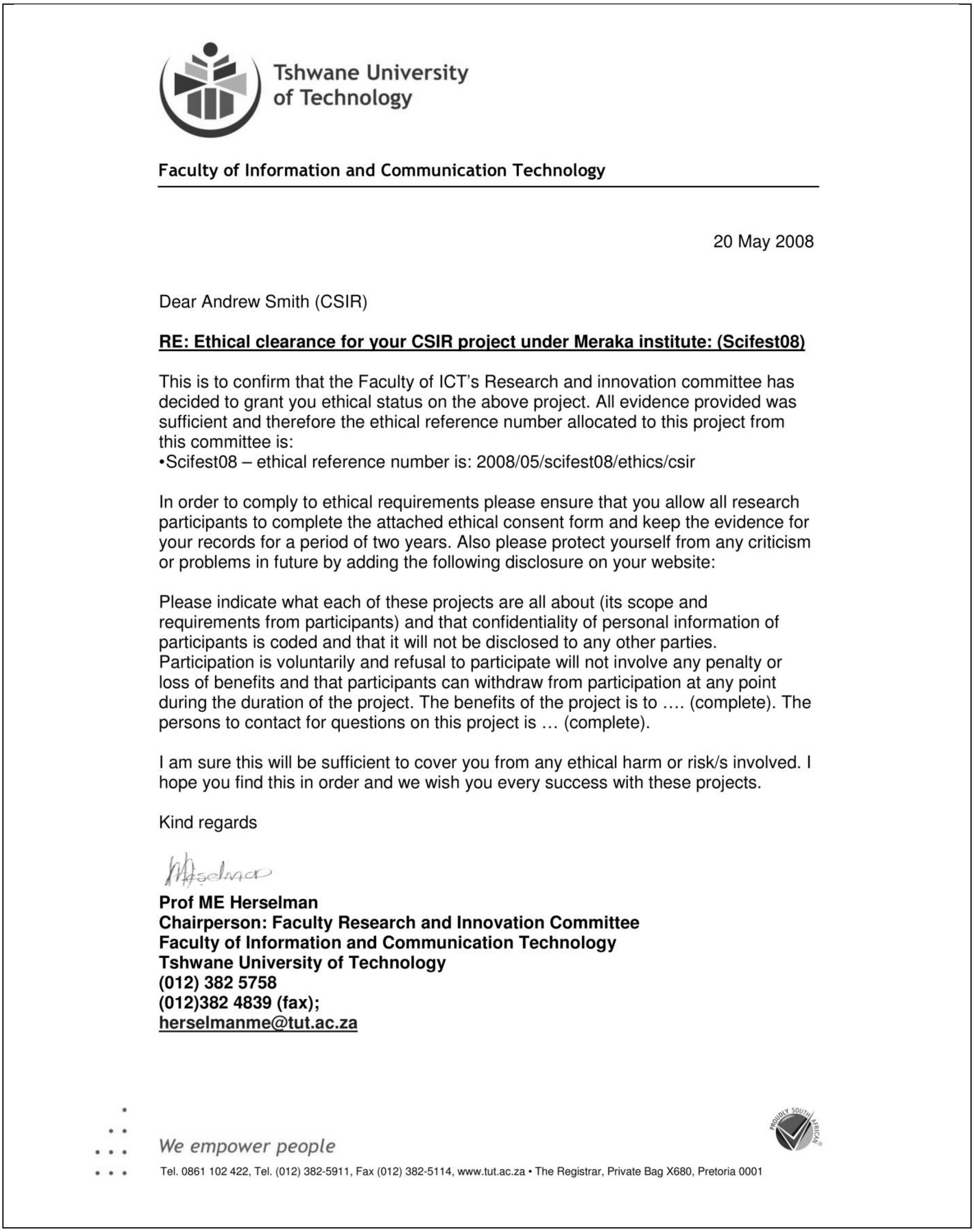
The following situations may impose ham or the risk of harm:

- Lack of anonymity for participants;
- Lack of confidentiality of information;
- Requests for sensitive information;
- Use of deceit;
- Use of medically invasive procedures;
- Cultural insensitivity;
- Use of "vulnerable" participant or those unable to give fully informed and voluntary consent.

Information and resources for research ethics

Prof ME Herselman
Tel: (012) 3825758/3824838
E-mail: herselmanme@tut.ac.za

Mrs Adri Coetzer
Tel (012) 3824838
Fax: (012) 3824839
e-mail: coetzera@tut.ac.za



Tshwane University of Technology

Faculty of Information and Communication Technology

20 May 2008

Dear Andrew Smith (CSIR)

RE: Ethical clearance for your CSIR project under Meraka institute: (Scifest08)

This is to confirm that the Faculty of ICT's Research and innovation committee has decided to grant you ethical status on the above project. All evidence provided was sufficient and therefore the ethical reference number allocated to this project from this committee is:
•Scifest08 – ethical reference number is: 2008/05/scifest08/ethics/csir

In order to comply to ethical requirements please ensure that you allow all research participants to complete the attached ethical consent form and keep the evidence for your records for a period of two years. Also please protect yourself from any criticism or problems in future by adding the following disclosure on your website:

Please indicate what each of these projects are all about (its scope and requirements from participants) and that confidentiality of personal information of participants is coded and that it will not be disclosed to any other parties. Participation is voluntarily and refusal to participate will not involve any penalty or loss of benefits and that participants can withdraw from participation at any point during the duration of the project. The benefits of the project is to …. (complete). The persons to contact for questions on this project is … (complete).

I am sure this will be sufficient to cover you from any ethical harm or risk/s involved. I hope you find this in order and we wish you every success with these projects.

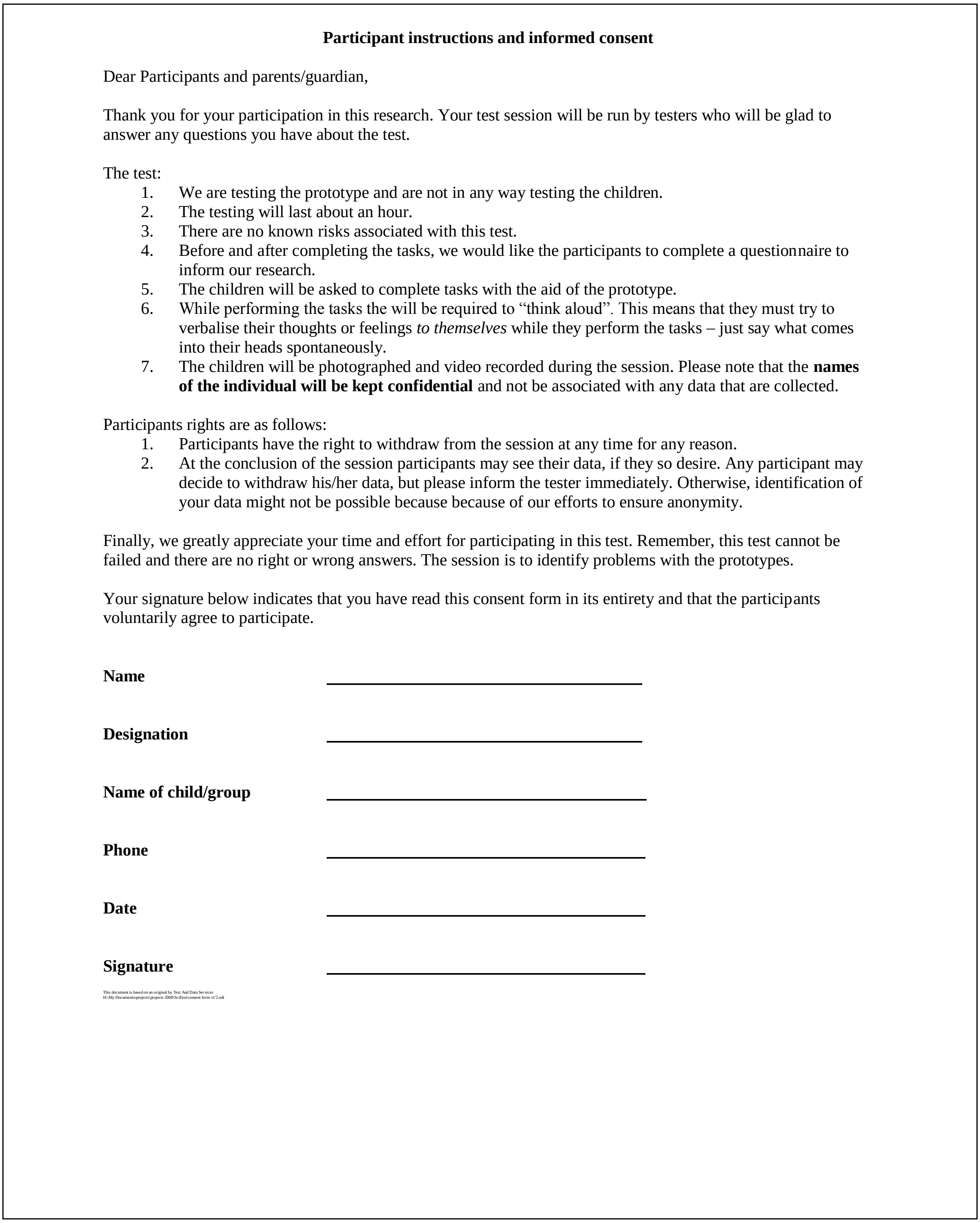
Kind regards

Prof ME Herselman
Chairperson: Faculty Research and Innovation Committee
Faculty of Information and Communication Technology
Tshwane University of Technology
(012) 382 5758
(012)382 4839 (fax);
herselmanme@tut.ac.za

We empower people

Tel. 0861 102 422, Tel. (012) 382-5911, Fax (012) 382-5114, www.tut.ac.za • The Registrar, Private Bag X680, Pretoria 0001

APPENDIX C .... Participant instructions and consent form: SciFest Africa 2008



Participant instructions and informed consent

Dear Participants and parents/guardian,

Thank you for your participation in this research. Your test session will be run by testers who will be glad to answer any questions you have about the test.

The test:

1. We are testing the prototype and are not in any way testing the children.
2. The testing will last about an hour.
3. There are no known risks associated with this test.
4. Before and after completing the tasks, we would like the participants to complete a questionnaire to inform our research.
5. The children will be asked to complete tasks with the aid of the prototype.
6. While performing the tasks the will be required to "think aloud". This means that they must try to verbalise their thoughts or feelings *to themselves* while they perform the tasks – just say what comes into their heads spontaneously.
7. The children will be photographed and video recorded during the session. Please note that the **names of the individual will be kept confidential** and not be associated with any data that are collected.

Participants rights are as follows:

1. Participants have the right to withdraw from the session at any time for any reason.
2. At the conclusion of the session participants may see their data, if they so desire. Any participant may decide to withdraw his/her data, but please inform the tester immediately. Otherwise, identification of your data might not be possible because because of our efforts to ensure anonymity.

Finally, we greatly appreciate your time and effort for participating in this test. Remember, this test cannot be failed and there are no right or wrong answers. The session is to identify problems with the prototypes.

Your signature below indicates that you have read this consent form in its entirety and that the participants voluntarily agree to participate.

Name ________________________________

Designation ________________________________

Name of child/group ________________________________

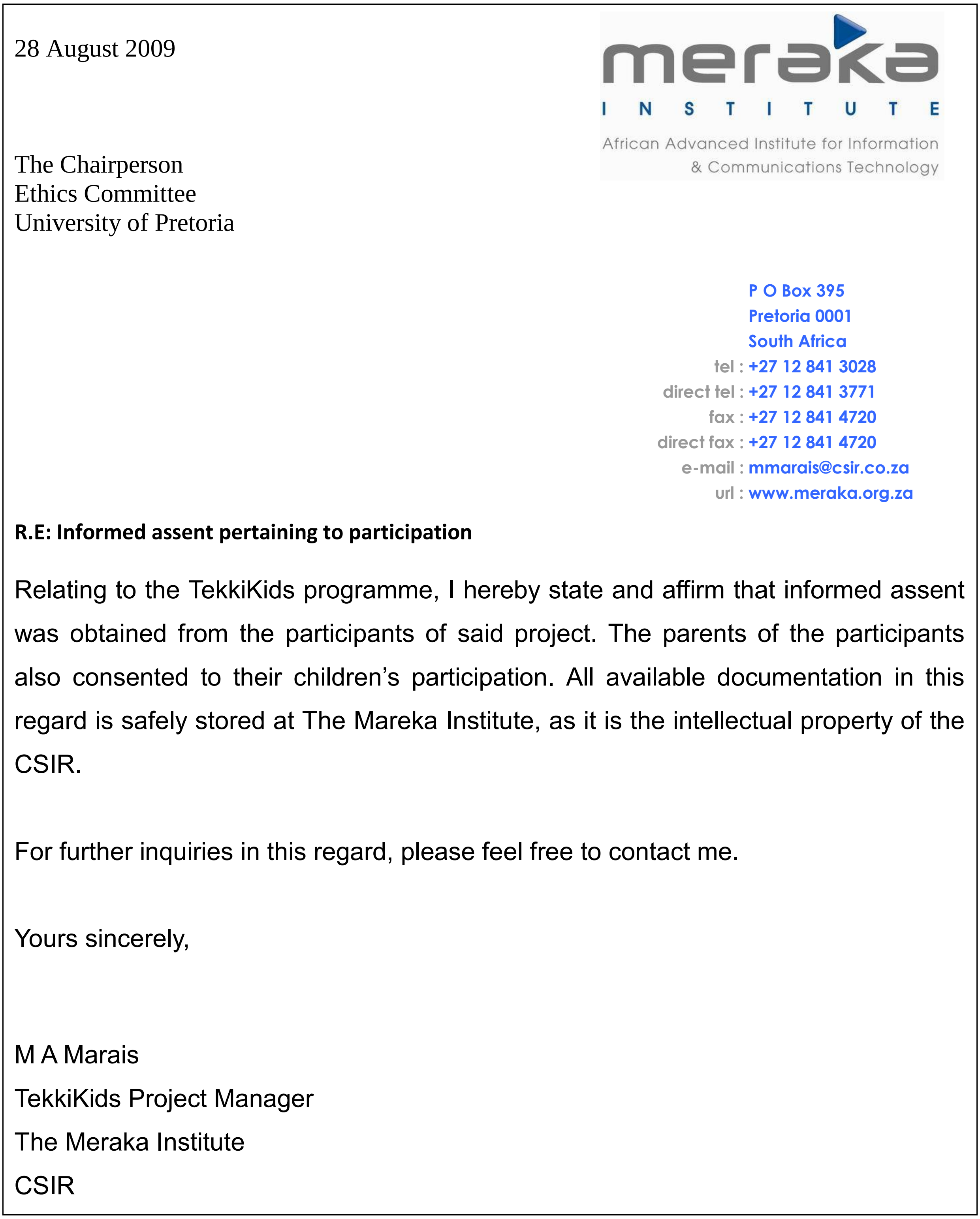
Phone ________________________________

Date ________________________________

Signature ________________________________

This document is based on an original by Test And Data Services
H:\My Documents\projects\projects 2008\SciFest\consent form v1'2.odt

APPENDIX D.. TekkiKids: Statement regarding informed assent and consent



28 August 2009

meraka
INSTITUTE
African Advanced Institute for Information
& Communications Technology

The Chairperson
Ethics Committee
University of Pretoria

P O Box 395
Pretoria 0001
South Africa
tel : +27 12 841 3028
direct tel : +27 12 841 3771
fax : +27 12 841 4720
direct fax : +27 12 841 4720
e-mail : mmarais@csir.co.za
url : www.meraka.org.za

R.E: Informed assent pertaining to participation

Relating to the TekkiKids programme, I hereby state and affirm that informed assent was obtained from the participants of said project. The parents of the participants also consented to their children's participation. All available documentation in this regard is safely stored at The Mareka Institute, as it is the intellectual property of the CSIR.

For further inquiries in this regard, please feel free to contact me.

Yours sincerely,

M A Marais
TekkiKids Project Manager
The Meraka Institute
CSIR

APPENDIX E .... TekkiKids: Invitation to participate and consent form

Arcadia Primary School Logo

meraka
INSTITUTE
African Advanced Institute for Information
& Communications Technology

P O Box 395
Pretoria 0001
South Africa
Tel. +27 12 841 3028
Direct Tel. +27 12 841 2952
Fax. +27 12 841 4720

E-mail. mmarais@csir.co.za
URL: www.meraka.org.za

26 August 2006

Dear Parent

The Meraka Institute (African Advanced Institute for ICT – managed by the CSIR) and Arcadia Primary School have selected your child to participate in the Kids' Club project which gives children the opportunity to experiment with science and technology in a fun and relaxed atmosphere. As mentioned in the invitation letter, the Kids' Club is a joint project between the Meraka Institute, the University of Pretoria and the University of Joensuu in Finland and is sponsored by the South African Department of Science and Technology and the Finnish Government. The project is a pilot which will be used for research in order to establish a much larger intervention. It is part of the Young Engineers Programme at the Meraka Institute.

Grade 5 and 6 learners are involved. The venue for the clubs will be at Arcadia Primary with some of the events being held at the CSIR. You will be informed beforehand of any events that take place outside the school premises. We plan to hold two-hour sessions every two weeks during the term, from 13h45 till 15h45 on Monday afternoons. The planned dates are:

Term 3 - 28 August and 11 September.
Term 4- 9 October, 23 October, 6 November, and 20 November.

We are in the process of constructing a website which will provide additional information about the sessions and where the learners' own reporting on their projects will be available.

The school and the Meraka Institute hereby request you to make a long-term commitment to the project for the learners to derive maximum benefit from participation. Your commitment will also maximize the research component over a period of two years (or until the end of grade 7). We intend to video record the sessions as part of the research. The research plan will be vetted by the ethics committee of the University of Pretoria. Publicity material such as videos and photographs may also be developed, for use by the school, Department of Science and Technology and the Meraka Institute, but the names of the children will not be revealed. The consent form is on the next page.

We look forward to having a great time together with your children, while at the same time conducting research that will benefit the children of South Africa.

Regards

Mario Marais
Researcher: ICTs in Education, Meraka Institute, CSIR, Pretoria, South Africa

Consent for a learner to participate in the Kids' Clubs project

I,……………………………..…………………………………………, the parent (legal guardian) of

……………………………………………………………………..……., hereby give my consent for the learner to participate in the Kids' Club project. I understand that any research to be done will be approved by the ethics committee of the University of Pretoria. The research results may be published. Publicity material may be developed, but the identity of the learners will be protected.

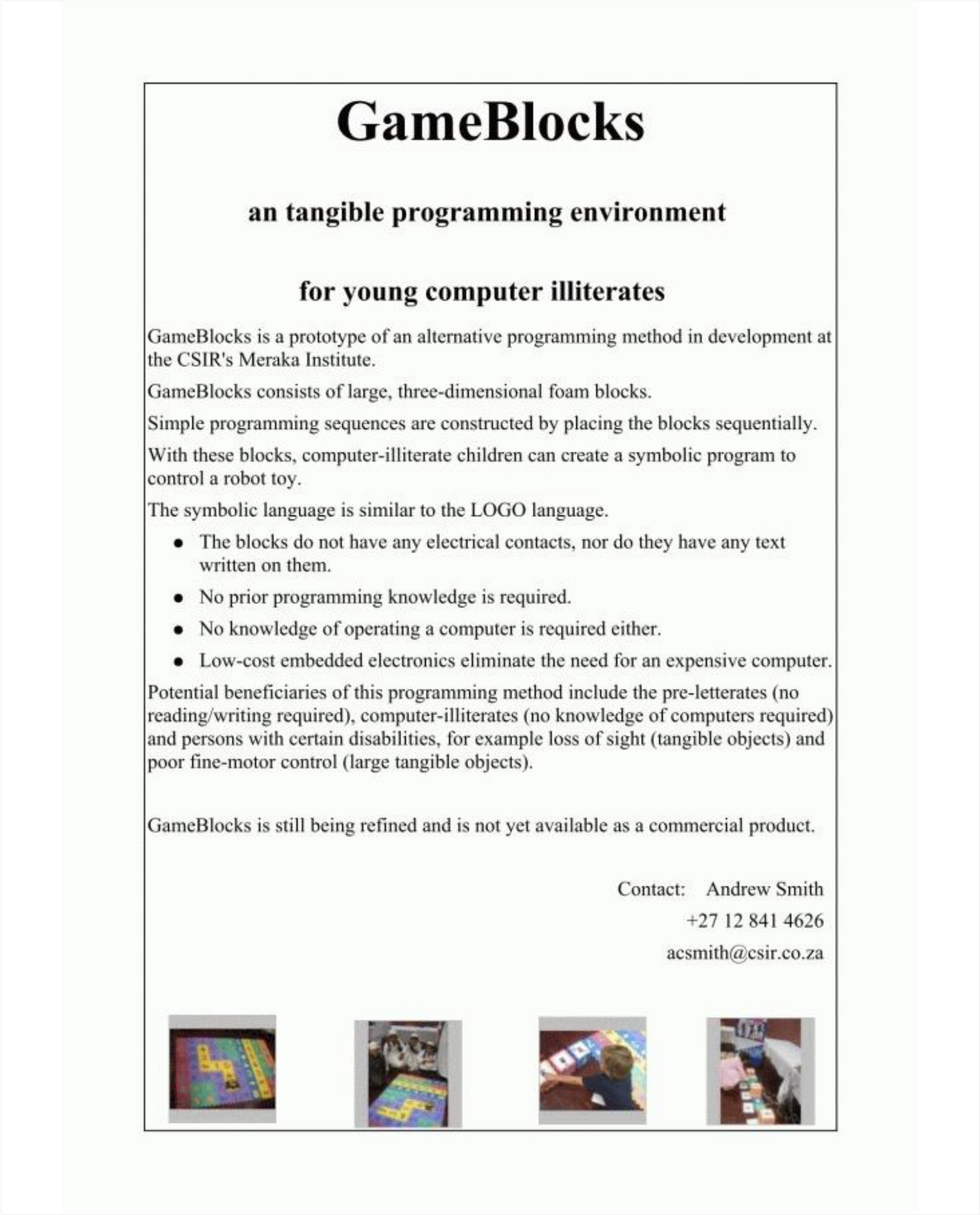
Signature:……………………………

Signed at ………………………………………..on this day of ………………………………………………..

Your contact Number(s):

Please indicate if there is any additional information that we must take note of (e.g. allergies, medication):

APPENDIX F .... Second iteration workshop invitation pamphlet: SciFest Africa 2008



GameBlocks

an tangible programming environment

for young computer illiterates

GameBlocks is a prototype of an alternative programming method in development at the CSIR's Meraka Institute.

GameBlocks consists of large, three-dimensional foam blocks.

Simple programming sequences are constructed by placing the blocks sequentially.

With these blocks, computer-illiterate children can create a symbolic program to control a robot toy.

The symbolic language is similar to the LOGO language.

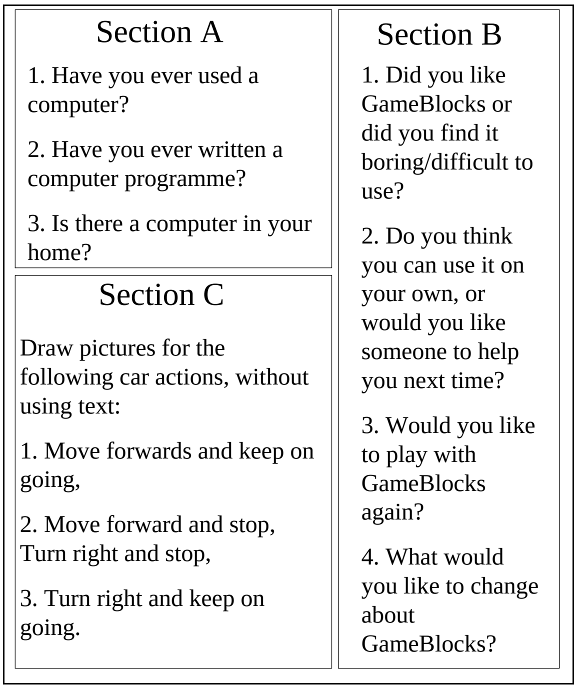
- The blocks do not have any electrical contacts, nor do they have any text written on them.
- No prior programming knowledge is required.
- No knowledge of operating a computer is required either.
- Low-cost embedded electronics eliminate the need for an expensive computer.

Potential beneficiaries of this programming method include the pre-letterates (no reading/writing required), computer-illiterates (no knowledge of computers required) and persons with certain disabilities, for example loss of sight (tangible objects) and poor fine-motor control (large tangible objects).

GameBlocks is still being refined and is not yet available as a commercial product.

Contact: Andrew Smith
+27 12 841 4626
acsmith@csir.co.za

APPENDIX G.. Second iteration partial evaluation form: Meraka Institute 2007



Section A

1. Have you ever used a computer?

2. Have you ever written a computer programme?

3. Is there a computer in your home?

Section C

Draw pictures for the following car actions, without using text:

1. Move forwards and keep on going,

2. Move forward and stop, Turn right and stop,

3. Turn right and keep on going.

Section B

1. Did you like GameBlocks or did you find it boring/difficult to use?

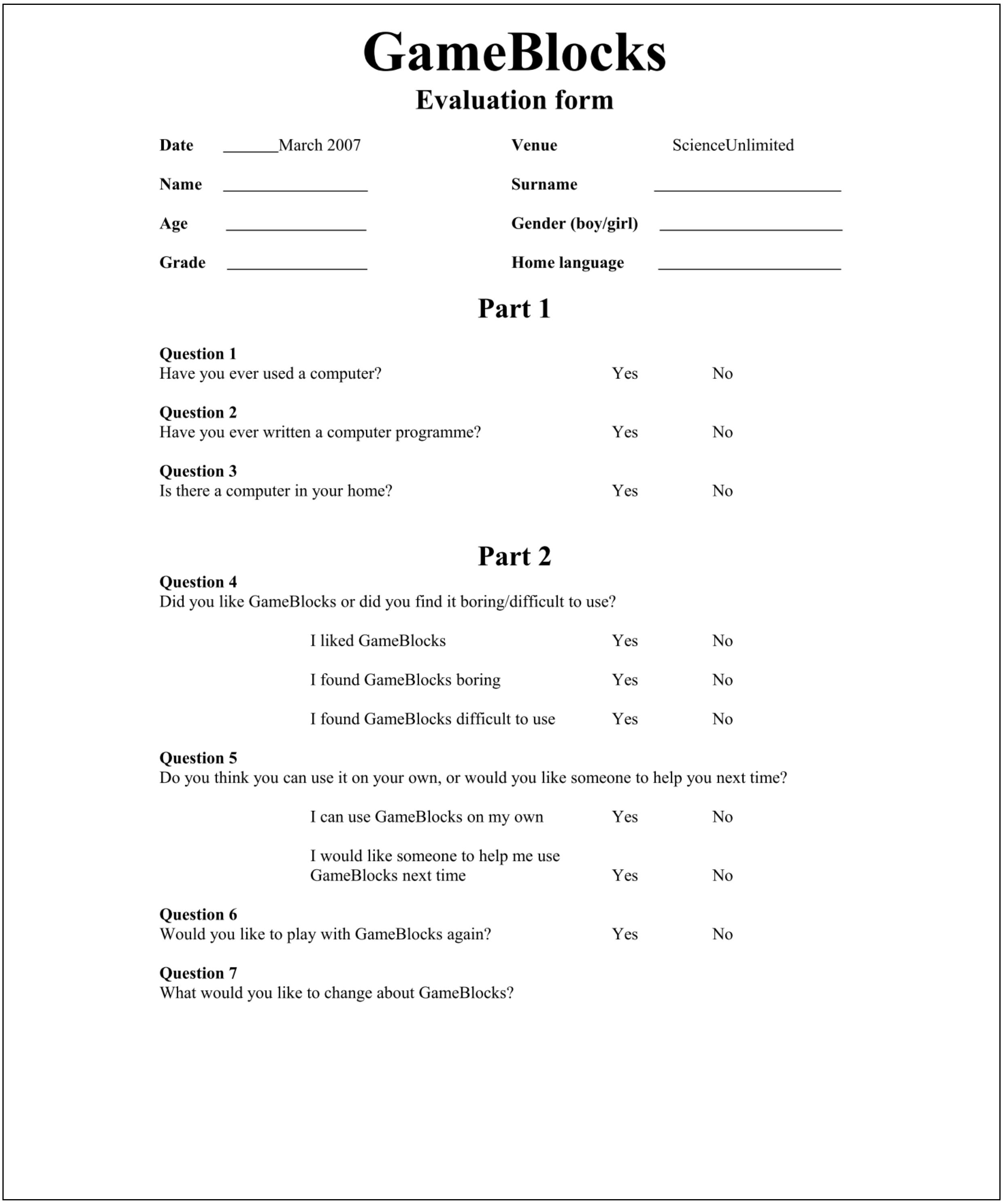
2. Do you think you can use it on your own, or would you like someone to help you next time?

3. Would you like to play with GameBlocks again?

4. What would you like to change about GameBlocks?

(Smith 2009d)

APPENDIX H…………………………………………………… Second iteration evaluation form: SciFest Africa and Science Unlimited 2007



GameBlocks

Evaluation form

Date ______March 2007 **Venue** ScienceUnlimited

Name ________________ **Surname** ____________________

Age ________________ **Gender (boy/girl)** ____________________

Grade ________________ **Home language** ____________________

Part 1

Question 1
Have you ever used a computer? Yes No

Question 2
Have you ever written a computer programme? Yes No

Question 3
Is there a computer in your home? Yes No

Part 2

Question 4
Did you like GameBlocks or did you find it boring/difficult to use?

I liked GameBlocks Yes No

I found GameBlocks boring Yes No

I found GameBlocks difficult to use Yes No

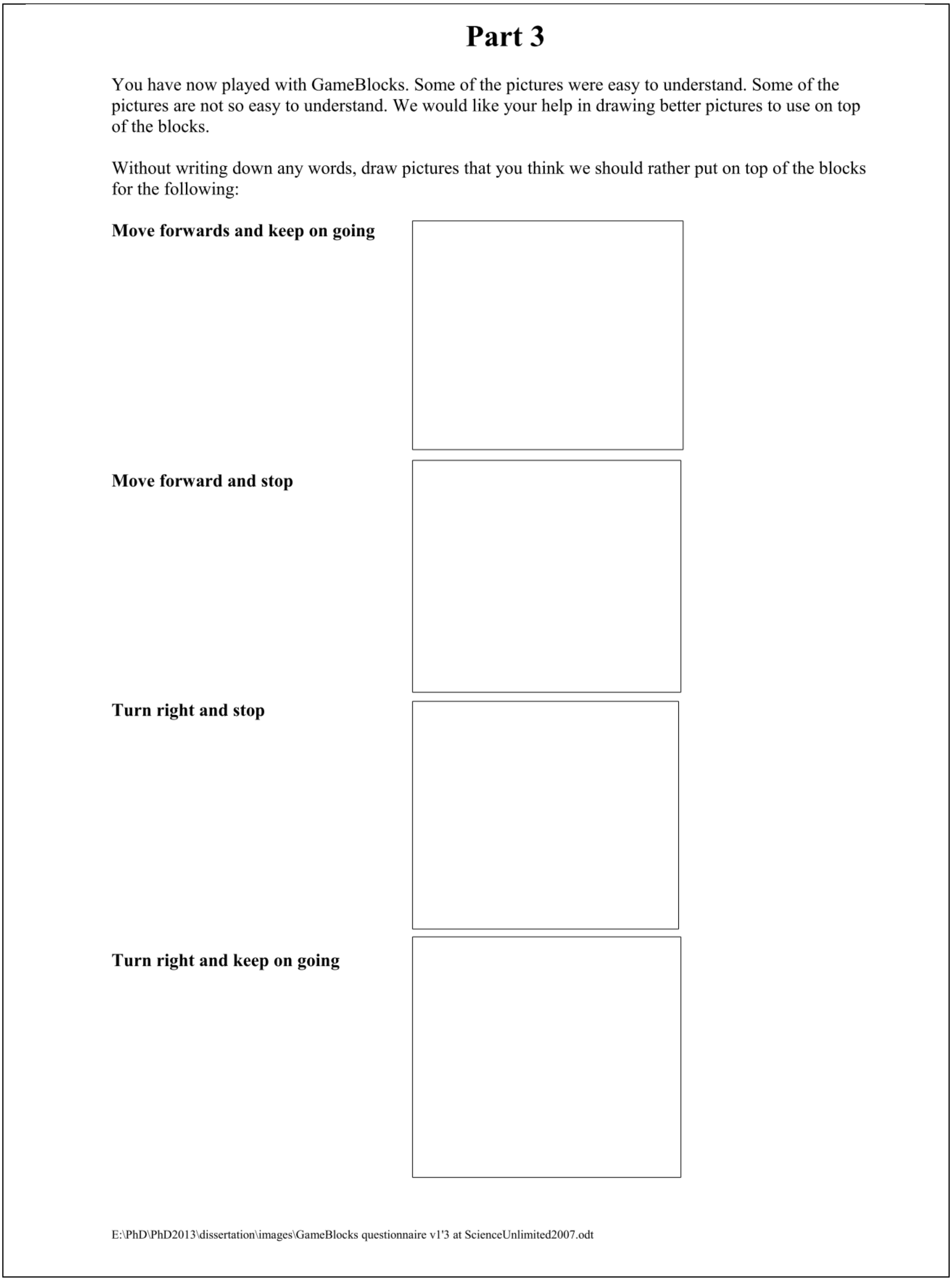
Question 5
Do you think you can use it on your own, or would you like someone to help you next time?

I can use GameBlocks on my own Yes No

I would like someone to help me use GameBlocks next time Yes No

Question 6
Would you like to play with GameBlocks again? Yes No

Question 7
What would you like to change about GameBlocks?



Part 3

You have now played with GameBlocks. Some of the pictures were easy to understand. Some of the pictures are not so easy to understand. We would like your help in drawing better pictures to use on top of the blocks.

Without writing down any words, draw pictures that you think we should rather put on top of the blocks for the following:

Move forwards and keep on going

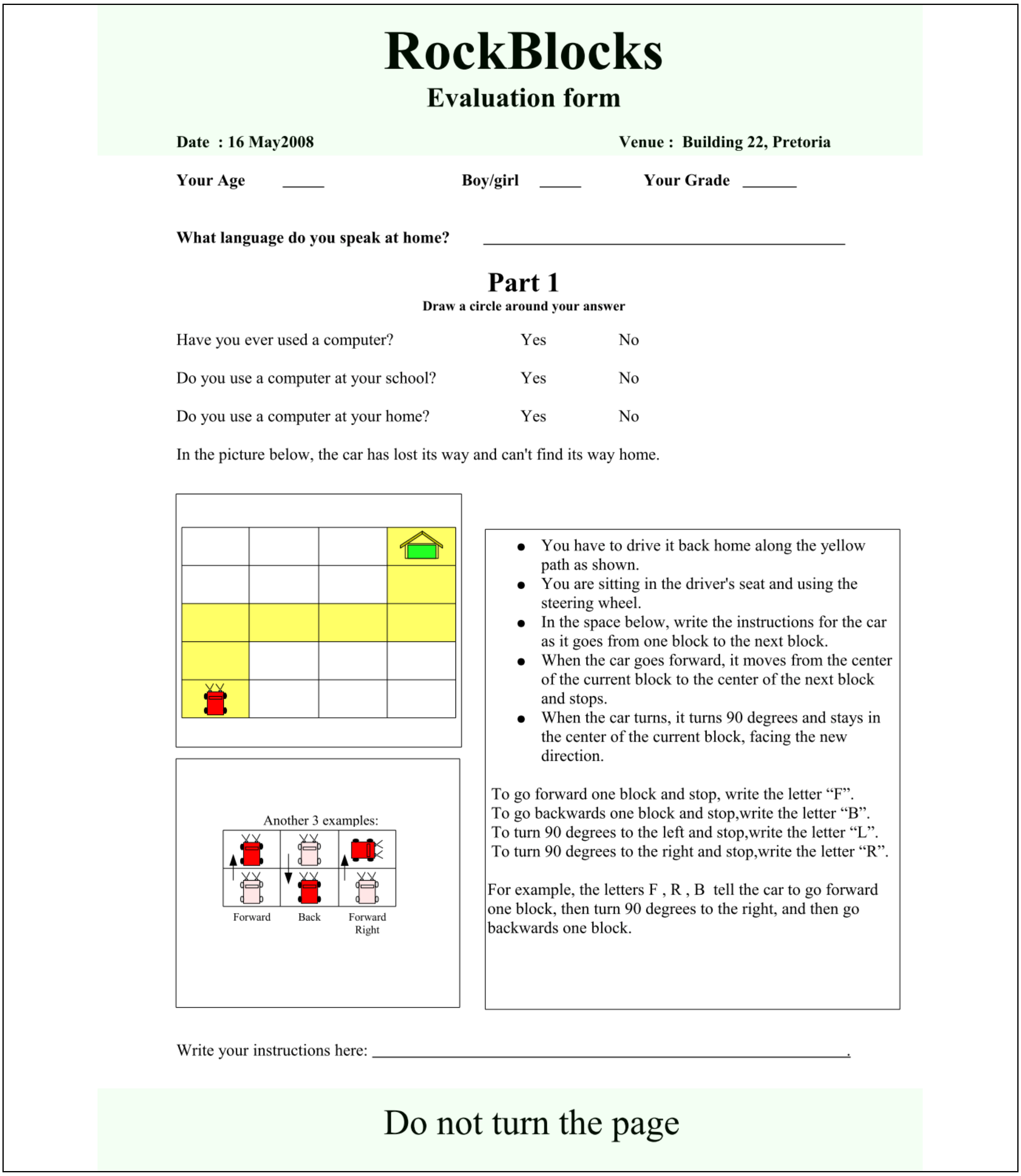
Move forward and stop

Turn right and stop

Turn right and keep on going

E:\PhD\PhD2013\dissertation\images\GameBlocks questionnaire v1'3 at ScienceUnlimited2007.odt

APPENDIX I .... Third iteration user evaluation forms: CSIR and SciFest Africa 2008



RockBlocks

Evaluation form

Date : 16 May2008 **Venue : Building 22, Pretoria**

Your Age _____ **Boy/girl** _____ **Your Grade** ______

What language do you speak at home? ____

Part 1

Draw a circle around your answer

Have you ever used a computer?	Yes	No
Do you use a computer at your school?	Yes	No
Do you use a computer at your home?	Yes	No

In the picture below, the car has lost its way and can't find its way home.

Another 3 examples:
Forward
Back
Forward Right

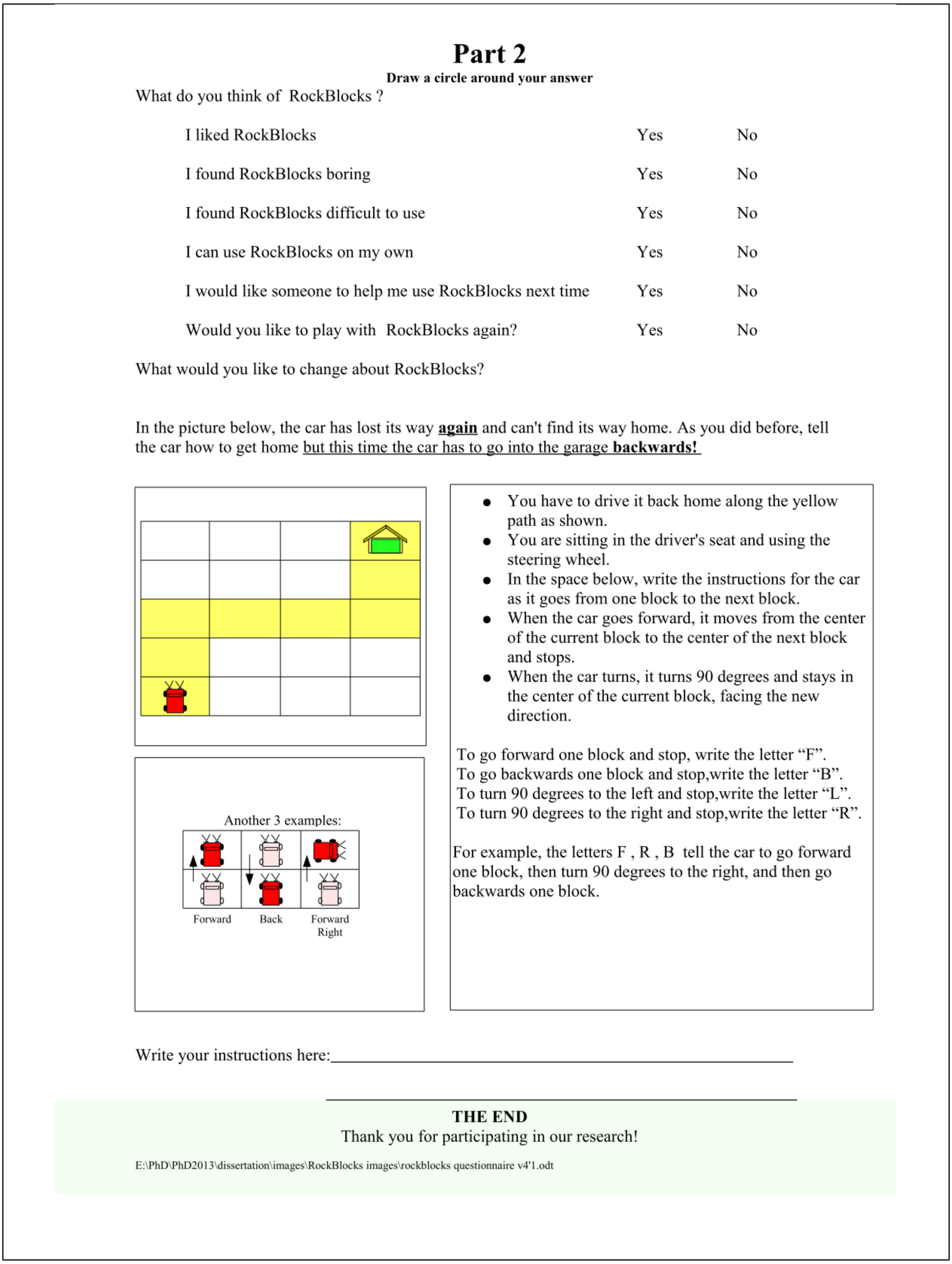
- You have to drive it back home along the yellow path as shown.
- You are sitting in the driver's seat and using the steering wheel.
- In the space below, write the instructions for the car as it goes from one block to the next block.
- When the car goes forward, it moves from the center of the current block to the center of the next block and stops.
- When the car turns, it turns 90 degrees and stays in the center of the current block, facing the new direction.

To go forward one block and stop, write the letter “F”.
To go backwards one block and stop,write the letter “B”.
To turn 90 degrees to the left and stop,write the letter “L”.
To turn 90 degrees to the right and stop,write the letter “R”.

For example, the letters F , R , B tell the car to go forward one block, then turn 90 degrees to the right, and then go backwards one block.

Write your instructions here: ____.

Do not turn the page



Part 2

Draw a circle around your answer

What do you think of RockBlocks ?

I liked RockBlocks	Yes	No
I found RockBlocks boring	Yes	No
I found RockBlocks difficult to use	Yes	No
I can use RockBlocks on my own	Yes	No
I would like someone to help me use RockBlocks next time	Yes	No
Would you like to play with RockBlocks again?	Yes	No

What would you like to change about RockBlocks?

In the picture below, the car has lost its way **again** and can't find its way home. As you did before, tell the car how to get home but this time the car has to go into the garage **backwards!**

- You have to drive it back home along the yellow path as shown.
- You are sitting in the driver's seat and using the steering wheel.
- In the space below, write the instructions for the car as it goes from one block to the next block.
- When the car goes forward, it moves from the center of the current block to the center of the next block and stops.
- When the car turns, it turns 90 degrees and stays in the center of the current block, facing the new direction.

To go forward one block and stop, write the letter "F".
To go backwards one block and stop,write the letter "B".
To turn 90 degrees to the left and stop,write the letter "L".
To turn 90 degrees to the right and stop,write the letter "R".

For example, the letters F , R , B tell the car to go forward one block, then turn 90 degrees to the right, and then go backwards one block.

Another 3 examples:

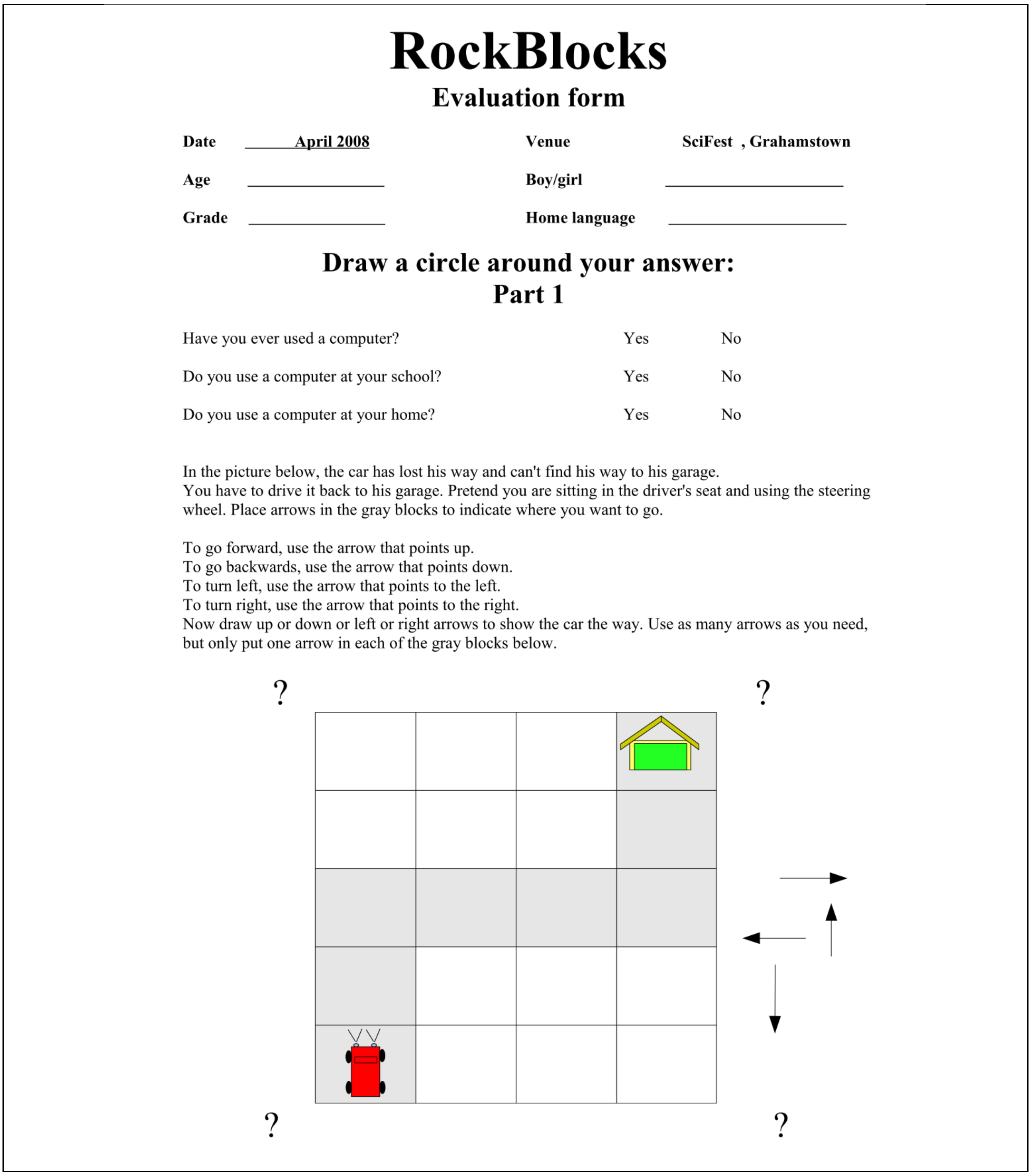
Write your instructions here:____

____

THE END

Thank you for participating in our research!

E:\PhD\PhD2013\dissertation\images\RockBlocks images\rockblocks questionnaire v4'1.odt



RockBlocks

Evaluation form

Date April 2008 **Venue** SciFest , Grahamstown

Age ________________ **Boy/girl** ____________________

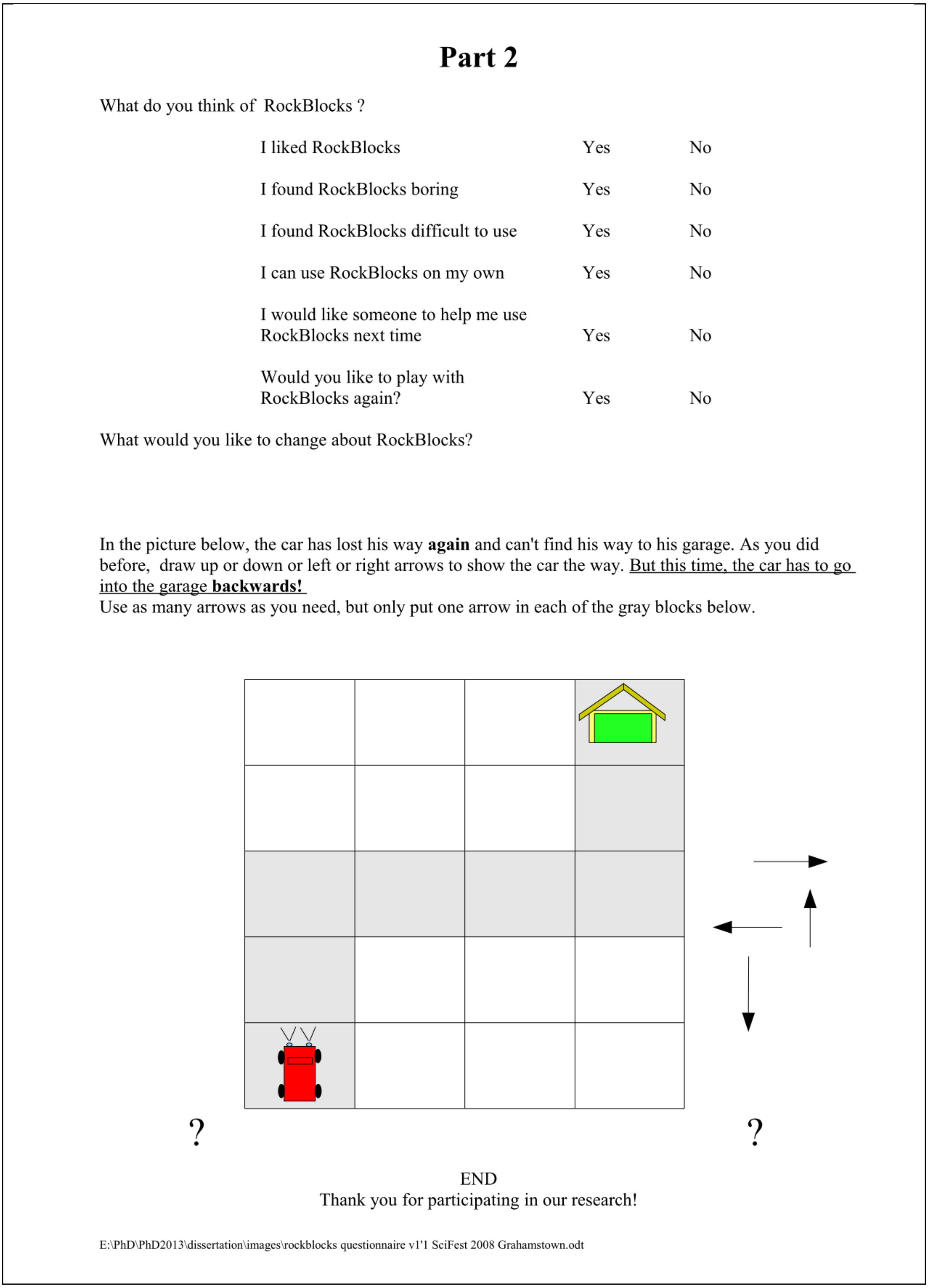
Grade ________________ **Home language** ____________________

Draw a circle around your answer:
Part 1

Have you ever used a computer?	Yes	No
Do you use a computer at your school?	Yes	No
Do you use a computer at your home?	Yes	No

In the picture below, the car has lost his way and can't find his way to his garage.
You have to drive it back to his garage. Pretend you are sitting in the driver's seat and using the steering wheel. Place arrows in the gray blocks to indicate where you want to go.

To go forward, use the arrow that points up.
To go backwards, use the arrow that points down.
To turn left, use the arrow that points to the left.
To turn right, use the arrow that points to the right.
Now draw up or down or left or right arrows to show the car the way. Use as many arrows as you need, but only put one arrow in each of the gray blocks below.



Part 2

What do you think of RockBlocks ?

I liked RockBlocks	Yes	No
I found RockBlocks boring	Yes	No
I found RockBlocks difficult to use	Yes	No
I can use RockBlocks on my own	Yes	No
I would like someone to help me use RockBlocks next time	Yes	No
Would you like to play with RockBlocks again?	Yes	No

What would you like to change about RockBlocks?

In the picture below, the car has lost his way **again** and can't find his way to his garage. As you did before, draw up or down or left or right arrows to show the car the way. But this time, the car has to go into the garage **backwards!**
Use as many arrows as you need, but only put one arrow in each of the gray blocks below.

? ?

END
Thank you for participating in our research!

E:\PhD\PhD2013\dissertation\images\rockblocks questionnaire v1'1 SciFest 2008 Grahamstown.odt